AWS Static Website Hosting

NEGIN HEZARJARIBI

Cloud Programming (DLBSEPCP01_E)

Studies: Bachelor in Applied Artificial Intelligence

IU INTERNATIONAL UNIVERSITY OF APPLIED SCIENCES (IU) | 18 MARCH 2025

Table of Contents

Introduction

In this project, we aim to deploy a secure, scalable, and globally optimized static website using AWS cloud services. The primary objective is to ensure high availability, low latency, and robust security by leveraging key AWS components such as Amazon S3 for storage, Amazon CloudFront as a Content Delivery Network (CDN), and Origin Access Identity (OAI) for secure access control. Additionally, Terraform is utilized as an Infrastructure as Code (IaC) tool to automate the provisioning and management of cloud resources, ensuring consistency and efficiency.

By integrating these technologies, we establish a cloud-based hosting solution that enhances performance, security, and scalability. This document provides an overview of the implemented architecture, the deployment process, and key technical details. We also highlight the benefits of using Terraform for automated deployments and explain the essential AWS configurations used in this project.

1. How to Deploy a Static Website on AWS

The deployment process involves several key steps to ensure a secure and efficient static website hosting setup. The workflow includes the following phases:

Setting up AWS Credentials: Creating an IAM user with programmatic access and assigning necessary permissions, including AmazonS3FullAccess and CloudFrontFullAccess.

Configuring Amazon S3: Creating a private S3 bucket to store website files and restricting public access while allowing CloudFront to fetch content securely.

Setting up CloudFront as a CDN: Deploying a CloudFront distribution to cache and serve content globally, reducing latency and improving load times.

Implementing Origin Access Identity (OAI): Ensuring secure access to the S3 bucket by restricting direct public access and allowing CloudFront as the only authorized entity.

Automating Infrastructure with Terraform: Using Terraform scripts to define and provision AWS resources, making the deployment process repeatable and manageable.

By following this structured approach, we successfully deployed a highly available and optimized static website while adhering to AWS best practices for security and performance. In the following sections, we will explore the key components and resources used in this project, starting with Terraform

2. Terraform

Terraform is an Infrastructure as Code (IaC) tool that allows us to define, provision, and manage cloud infrastructure in a declarative way. It enables automation, consistency, and version control when deploying resources in AWS. By using Terraform, we can ensure that infrastructure is easily replicable and can be modified efficiently when needed.

Additionally, Terraform provides key advantages, including automation, eliminating the need for manual configuration, ensuring consistency across environments, and offering scalability to adjust infrastructure as project requirements evolve. While this project focuses on AWS, Terraform supports multiple cloud providers, making it a versatile tool for infrastructure management.

2.1 Terraform in This Project

In this project, Terraform plays a critical role in automating the deployment of AWS resources. By defining infrastructure as code, we minimize human error, streamline resource provisioning, and maintain a structured, repeatable approach to cloud infrastructure deployment. Terraform enables us to create and manage the S3 bucket, configure CloudFront for content delivery, and enforce access control policies efficiently. Using Terraform ensures that all infrastructure components are provisioned efficiently and can be easily modified or replicated in the future. The use of Infrastructure as Code (IaC) also improves maintainability and version control, making the deployment process structured and repeatable.

2.2 Terraform Files and Their Responsibilities

For this project, Terraform is structured into multiple files, each responsible for different aspects of the deployment:

provider.tf: Configures the AWS provider and defines the region.

variables.tf: Declares variables such as the AWS region and S3 bucket name.

main.tf: Defines the S3 bucket and its access policies.

cloudfront.tf: Sets up the CloudFront distribution and Origin Access Identity (OAI).

outputs.tf: Specifies output values such as the CloudFront domain name.

terraform.tfvars: Stores default values for variables.

versions.tf : Defines required Terraform and AWS provider versions for compatibility.

2.3 Setting Up AWS Credentials for Terraform

Before using Terraform, we need to configure AWS credentials:

Create an IAM User: The user should have programmatic access.

Attach Policies: Assign the necessary permissions (AmazonS3FullAccess, CloudFrontFullAccess).

Save Credentials: Store the AWS Access Key and Secret Key securely for Terraform authentication.

With these configurations in place, Terraform allows us to efficiently deploy and manage our static website hosting infrastructure on AWS.

3. Amazon S3

Amazon Simple Storage Service (S3) is a highly scalable, durable, and secure object storage service designed for various use cases, including static website hosting. In this project, S3 serves as the primary storage solution for website files, ensuring high availability and seamless integration with AWS services.

One of the key reasons for selecting Amazon S3 is its cost-effective storage, which offers built-in security, redundancy, and durability. By using S3, website files are stored across multiple availability zones, ensuring reliability and data persistence. Additionally, S3 natively integrates with Amazon CloudFront, enabling optimized content delivery with reduced latency (Amazon Web Services, 2024).

Another critical aspect of using Amazon S3 is access control. To maintain a secure architecture, the S3 bucket is configured to be private, preventing direct public access. Instead, CloudFront retrieves content from S3 securely through Origin Access Identity (OAI), enforcing controlled access while improving

performance. This ensures that content is securely delivered to users without exposing the S3 bucket to public access.

3.1 Terraform Implementation for Amazon S3

Terraform is utilized to define and manage the S3 bucket in a structured and automated manner. The Terraform configuration ensures that:

- The S3 bucket is created with a globally unique name.
- Public access to the bucket is restricted, preventing unauthorized access.
- A bucket policy is applied, allowing only CloudFront to fetch objects from S3.
- Versioning and encryption can be enabled (if required) to enhance security and durability.

By implementing these configurations using Terraform, we achieve a scalable, repeatable, and secure storage solution for hosting the static website.

3.2 Key Terraform Code for Amazon S3

```
# Create an S3 bucket for static website hosting

resource "aws_s3_bucket" "static_site" {

  bucket = var.bucket_name

}

# Restrict public access to the S3 bucket

resource "aws_s3_bucket_public_access_block" "static_site" {

  bucket                  = aws_s3_bucket.static_site.id

  block_public_acls       = true

  block_public_policy     = true

  ignore_public_acls      = true

  restrict_public_buckets = true

}
```

```
# Define an S3 bucket policy to allow access only from CloudFront OAI

resource "aws_s3_bucket_policy" "static_site" {

  bucket = aws_s3_bucket.static_site.id

  policy = <<POLICY

{

  "Version": "2012-10-17",

  "Statement": [

    {

      "Effect": "Allow",

      "Principal": {

        "AWS": "${aws_cloudfront_origin_access_identity.oai.iam_arn}"

      },

      "Action": "s3:GetObject",

      "Resource": "arn:aws:s3:::${aws_s3_bucket.static_site.bucket}/*"

    }

  ]

}

POLICY

}
```

This configuration ensures that:

- The S3 bucket is created and kept private.
- Only CloudFront (via OAI) has permission to access website files.
- Public access is entirely blocked to prevent security risks.

In the next section, we will discuss Amazon CloudFront and its role in optimizing website performance and security.

4. Amazon CloudFront

Amazon CloudFront is a **Content Delivery Network (CDN)** that helps improve the performance, security, and scalability of web applications by caching content at edge locations around the world. This reduces latency and enhances the user experience when accessing static websites (Amazon Web Services, 2023).

4.1 Amazon CloudFront as a CDN

A CDN is a distributed network of servers that delivers content based on the geographical location of users. Instead of serving content directly from the Amazon S3 bucket, CloudFront caches it at edge locations, reducing load times and minimizing latency for users worldwide (Gill & Li, 2022).

Key benefits of using CloudFront as a CDN:

Lower Latency : Content is delivered from the nearest edge location.

Improved Security : AWS WAF (Web Application Firewall) integration and secure access to origin servers.

Cost Optimization : Reduces S3 data transfer costs by serving cached content.

Better Availability : Automatically routes requests to the best-performing edge location.

In this project, CloudFront is configured as a CDN for our static website hosted in an Amazon S3 bucket. Below is the Terraform implementation of the CloudFront distribution:

```
# Create Origin Access Identity (OAI) for CloudFront

resource "aws_cloudfront_origin_access_identity" "oai" {

  comment = "OAI for accessing S3 bucket securely"

}
```

```
# CloudFront distribution

resource "aws_cloudfront_distribution" "cdn" {

  origin {

    domain_name = aws_s3_bucket.static_site.bucket_regional_domain_name

    origin_id   = "S3Origin"

    s3_origin_config {

      origin_access_identity =
aws_cloudfront_origin_access_identity.oai.cloudfront_access_identity_path

    }

  }



  enabled             = true

  default_root_object = "index.html"



  default_cache_behavior {

    target_origin_id       = "S3Origin"

    viewer_protocol_policy = "redirect-to-https"

    allowed_methods        = ["GET", "HEAD"]

    cached_methods         = ["GET", "HEAD"]

    compress              = true

    forwarded_values {

      query_string = false

      cookies {

        forward = "none"

      }
```

```
    }
  }

  price_class = "PriceClass_100"

  custom_error_response {
    error_code            = 404
    response_page_path    = "/error.html"
    response_code         = 404
    error_caching_min_ttl = 300
  }

  restrictions {
    geo_restriction {
      restriction_type = "none"
    }
  }

  viewer_certificate {
    cloudfront_default_certificate = true
  }
}
```

4.2 Origin Access Identity (OAI) and Security

By default, an Amazon S3 bucket is private, and to allow CloudFront to retrieve content from it, we use an **Origin Access Identity (OAI)**. OAI ensures that only CloudFront can access the objects stored in S3, preventing direct public access (Krishnan, 2021).

To enforce this restriction, we define an S3 bucket policy that grants CloudFront OAI permission to read objects from the bucket:

```
# Set an S3 bucket policy to allow access only from CloudFront OAI

resource "aws_s3_bucket_policy" "static_site" {

  bucket = aws_s3_bucket.static_site.id

  policy = <<POLICY

{

  "Version": "2012-10-17",

  "Statement": [

    {

      "Effect": "Allow",

      "Principal": {

        "AWS": "${aws_cloudfront_origin_access_identity.oai.iam_arn}"

      },

      "Action": "s3:GetObject",

      "Resource": "arn:aws:s3:::${aws_s3_bucket.static_site.bucket}/*"

    }

  ]

}

POLICY
```

```
}
```

By implementing **OAI and CloudFront**, the static website hosted in Amazon S3 is securely delivered via the CloudFront CDN, ensuring both performance improvements and restricted access to the S3 bucket.

5. Results & Outputs

After successfully deploying the static website using **Terraform**, we verified the following outputs:

The Amazon S3 bucket was created and configured to store website files securely.

The CloudFront distribution was established, ensuring efficient content delivery through edge locations.

The **CloudFront URL** was generated, allowing users to access the website.

Security measures were correctly implemented, restricting direct access to the S3 bucket while allowing CloudFront to serve content securely.

The **Terraform output** confirms the successful deployment:

```
output "cloudfront_url" {

description = "CloudFront Distribution URL"

value = aws_cloudfront_distribution.cdn.domain_name }
```

By accessing this **CloudFront URL**, users can view the static website, benefiting from low latency and optimized performance due to caching at edge locations. In this project CloudFront URL is (https://d3n1b7jgbt32ke.cloudfront.net).

Conclusion

This project demonstrated how to deploy a secure and scalable static website using Amazon S3, CloudFront, and Terraform. By making use of CloudFront as a CDN, we significantly improved website performance while ensuring controlled access through OAI. The use of Infrastructure as Code (IaC) via Terraform streamlined the deployment, making it reproducible and manageable. This implementation

provides a cost-effective, scalable, and secure solution for hosting static websites on AWS, suitable for both personal and business applications.

References

1. Amazon Web Services. (2024). *Amazon CloudFront: Content delivery made fast and secure*. https://aws.amazon.com/cloudfront/
2. Amazon Web Services. (2023). *Amazon CloudFront Documentation*. Retrieved from https://docs.aws.amazon.com/cloudfront/
3. Amazon Web Services. (2024). *Amazon S3 features*. AWS Documentation. Retrieved from https://aws.amazon.com/s3/features/
4. Gill, P., & Li, W. (2022). *CDN Performance Optimization: A Comparative Study*. IEEE Transactions on Networking.
5. Krishnan, M. (2021). *Cloud Security Principles: Protecting Web Applications*. Addison-Wesley.