Task 1: Host a simple webpage on AWS

Course name – Cloud Programming (DLBSEPCP01_E)

A course of Study – Bachelor of Science in Applied Artificial Intelligence

Author Name – Fotimakhon Gulamova

Matriculation Number – 92116230

Tutor's Name – Georgi Dimchev

Table of Content

Introduction

Problem Statement

Design and implement a highly available, low-latency website on AWS using Amazon S3 bucket for hosting and CloudFront for content delivery using Infrastructure as Code (IaC) tools like Terraform for reproducibility. The goal of the task is to host an “index” HTML file, guaranteeing high availability, minimizing latency, and enabling auto-scaling for optimal performance.

Key Considerations

- Host a simple “index” HTML file on AWS services.
- Ensure high availability of the website.
- Low latency for global visitors accessing the website.
- Implement auto-scaling capabilities for the website backend to handle increased visitor traffic efficiently.

Approach

- Amazon S3 bucket for web hosting and Amazon CloudFront for content delivery and global distribution.
- Develop an Infrastructure as Code (IaC) script using Terraform to create the AWS infrastructure required for hosting the webpage.
- Implement CloudFront to distribute content globally and reduce latency for visitors accessing the webpage from different geographical locations.
- Configure auto-scaling settings to dynamically adjust resources based on the incoming traffic to maintain optimal performance and responsiveness.

Step 1: AWS Set Up

Before deploying the infrastructure you must set up your AWS credentials as an environmental variable:

- **AWS Access Key:** Identify an account
- **AWS Secret Key:** The corresponding secret key pairs with an access key

For Windows:

```
setx AWS_ACCESS_KEY_ID "AWS_ACCESS_KEY_ID"
setx AWS_SECRET_ACCESS_KEY "AWS_SECRET_ACCESS_KEY"
```

For macOS / Linux:

```
export AWS_ACCESS_KEY_ID="AWS_ACCESS_KEY_ID"
export AWS_SECRET_ACCESS_KEY="AWS_SECRET_ACCESS_KEY"
```

Then I set up an AWS (Amazon Web Services) provider configuration.

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

region = "us-east-1": This line specifies the AWS region where resources were created or managed. In this case, I set it to **"us-east-1"**. AWS regions represent geographically distinct locations where AWS data centers are situated.

Step 2: AWS S3 Bucket

S3 Bucket

This code is responsible for creating an AWS S3 Bucket with a specified name and assigning tags based on the variables provided (**var.bucket_name** for the bucket name and **var.bucket_tags** for the tags).

```
resource "aws_s3_bucket" "s3bucket" {
  bucket = var.bucket_name

  tags = var.bucket_tags
}
```

S3 Bucket Configuration

This code sets a static website hosting configuration for an existing S3 bucket by setting the default document in the variable **var.index_document_suffix.**

```
resource "aws_s3_bucket_website_configuration" "static_website_bucket" {
  bucket = aws_s3_bucket.s3bucket.id

  index_document {
    suffix = var.index_document_suffix
  }
}
```

The Terraform code below sets ownership controls for an existing S3 bucket, specifying the desired object ownership configuration through the **var.object_ownership** variable. Also, this feature helps define who should own the objects uploaded to the S3 bucket.

```
resource "aws_s3_bucket_ownership_controls" "static_website_bucket" {
  bucket = aws_s3_bucket.s3bucket.id

  rule {
    object_ownership = var.object_ownership
  }
}
```

This code sets a configuration of the public access settings for an AWS S3 bucket using the **aws_s3_bucket_public_access_block** resource. I set all settings to **false**, meaning public access is not blocked.

```
resource "aws_s3_bucket_public_access_block" "static_website_bucket" {
  bucket = aws_s3_bucket.s3bucket.id

  block_public_acls       = false
  block_public_policy     = false
  ignore_public_acls      = false
  restrict_public_buckets = false
}
```

This Terraform code defines an AWS S3 bucket ACL (Access Control List) configuration using the **aws_s3_bucket_acl** resource for an existing S3 bucket based on the specified in the variable **var.acl_setting**, ensuring that the ACL configuration is applied only after certain dependencies (**aws_s3_bucket_ownership_controls** and **aws_s3_bucket_public_access_block**).

```
resource "aws_s3_bucket_acl" "static_website_bucket" {
  depends_on = [
    aws_s3_bucket_ownership_controls.static_website_bucket,
    aws_s3_bucket_public_access_block.static_website_bucket,
  ]
```

```
  bucket = aws_s3_bucket.s3bucket.id
  acl    = var.acl_setting
}
```

S3 Bucket Policy

This Terraform code creates an S3 bucket policy allowing public read access to objects within the specified bucket using the **aws_s3_bucket_policy** resource.

```
resource "aws_s3_bucket_policy" "s3_bucket_policy" {
  bucket = aws_s3_bucket.s3bucket.bucket

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Sid       = "PublicReadGetObject"
        Effect    = "Allow"
        Principal = "*"
        Action    = "s3:GetObject"
        Resource = [
          aws_s3_bucket.s3bucket.arn,
          "${aws_s3_bucket.s3bucket.arn}/*"
        ]
      }
    ]
  })
}
```

Upload Website Files

The code below uploads website files to an AWS S3 bucket using the **aws_s3_object** resource.

```
resource "aws_s3_object" "website_files" {
  bucket       = aws_s3_bucket.s3bucket.id
  for_each     = fileset("website files/", "**/*.*")
  key          = each.value
  source       = "website files/${each.value}"
  content_type = each.value
}
```

Step 3: AWS CloudFront Distribution

CloudFront Distribution

The code below is responsible for setting up an AWS CloudFront distribution that uses an S3 bucket as its origin, configuring various settings such as distribution behavior, SSL certificate, cache behavior, and geo-restriction, based on the provided variables and default configurations.

```
locals {
  s3_origin_id = var.s3_origin_id
}

resource "aws_cloudfront_distribution" "s3_distribution" {
  depends_on = [aws_s3_bucket.s3bucket]

  origin {
    domain_name = aws_s3_bucket.s3bucket.bucket_regional_domain_name
    origin_id   = local.s3_origin_id
  }

  enabled             = true
  is_ipv6_enabled     = var.ipv6_enabled
  default_root_object = var.default_root_object

  viewer_certificate {
    cloudfront_default_certificate = true
  }

  restrictions {
    geo_restriction {
      restriction_type = "none"
      locations        = []
    }
  }

  default_cache_behavior {
    cache_policy_id        = var.cache_behavior.cache_policy_id
    viewer_protocol_policy = var.cache_behavior.viewer_protocol_policy
    allowed_methods        = var.cache_behavior.allowed_methods
    cached_methods         = var.cache_behavior.cached_methods
    target_origin_id       = local.s3_origin_id
  }
}
```

Step 4: AWS Outputs

Outputs

The code illustrates the outputs that are used to provide useful information after deploying the infrastructure. They display the CloudFront distribution ID, the CloudFront distribution URL, and the S3 hosting URL.

```
output "cloudfront_id" {
  description = "Cloudfront ID"
  value       = aws_cloudfront_distribution.s3_distribution.id
}

output "cloudfront_url" {
  description = "Cloudfront distribution URL (HTTPS)"
  value       =
"https://${aws_cloudfront_distribution.s3_distribution.domain_name}"
}

output "s3_url" {
  description = "S3 hosting URL"
  value       =
aws_s3_bucket_website_configuration.static_website_bucket.website_endpoint
}
```

Conclusion

The goal of this project was to create a high-performing website on AWS using Amazon S3 and CloudFront using Terraform. It ensured global accessibility, low latency, and auto-scaling features for optimal performance. The steps included setting up S3 buckets, configuring CloudFront distribution, and providing the necessary output information after deployment. Overall, it successfully achieved the goal of establishing a highly available, low-latency website on AWS.