INTERNATIONAL
UNIVERSITY OF
APPLIED SCIENCES

Phase 3: Host a simple webpage on AWS

Course name – Cloud Programming (DLBSEPCP_E)

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

Author Name – Zukhrakhon Gulomova

Matriculation Number – 92118181

Tutor's Name – Georgi Dimchev

Table of Content

Introduction

Purpose

The purpose of this document is to provide a comprehensive description and Infrastructure as Code (IaC) using Terraform for deploying a highly available and globally distributed static website on AWS. The infrastructure includes an S3 bucket for storage, CloudFront for content delivery, and associated configurations.

Scope

This project covers the creation of an S3 bucket configured for static website hosting and a CloudFront distribution to ensure global availability and low-latency content delivery. The infrastructure is defined using Terraform to meet high availability requirements, global latency avoidance, and scalability.

Infrastructure

Provider Setup

Before deploying the infrastructure using Terraform, you must set up your AWS credentials.

- **AWS Access Key ID:** Your AWS access key ID, which identifies your account.
- **AWS Secret Access Key:** The corresponding secret key pairs with your access key.

Setting Up AWS Credentials

- Open a terminal or PowerShell window.
- Set your AWS access key ID and secret access key as environment variables.

```
PS C:\Users\User\AWS-Project\Project>
$env:AWS_ACCESS_KEY_ID="YOUR_ACCESS_KEY_ID"
PS C:\Users\User\AWS-Project\Project>
$env:AWS_SECRET_ACCESS_KEY="YOUR_SECRET_ACCESS_KEY"
```

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

S3 Bucket Configuration

```
resource "aws_s3_bucket" "bucket1" {
  bucket         = "zukhra-tf-bucket"
  force_destroy = true

  tags = {
    Name          = "My bucket"
    Environment = "Dev"
  }
}
```

`aws_s3_bucket`

- Creates an S3 bucket for storing static website files.
- Enables the `force_destroy` option to allow for the removal of all objects when deleting the bucket.

```
resource "aws_s3_object" "files" {
  bucket        = aws_s3_bucket.bucket1.id
  for_each      = fileset("website/", "**/*.*")
  key           = each.value
  source        = "website/${each.value}"
  content_type = each.value
}
```

`aws_s3_object`

- Uploads files to the S3 bucket, facilitating the deployment of the static website.

```
resource "aws_s3_bucket_ownership_controls" "ownership" {
  bucket = aws_s3_bucket.bucket1.id
  rule {
    object_ownership = "BucketOwnerPreferred"
  }
}
```

`aws_s3_bucket_ownership_controls`

- Configures object ownership controls for the S3 bucket.

```
resource "aws_s3_bucket_public_access_block" "public_access_block" {
  bucket = aws_s3_bucket.bucket1.id

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

`aws_s3_bucket_public_access_block`

- Configures public access block settings for the S3 bucket.

```
resource "aws_s3_bucket_acl" "s3_bucket_acl" {
  bucket = aws_s3_bucket.bucket1.id
  acl    = "public-read"

  depends_on = [
    aws_s3_bucket_ownership_controls.ownership,
    aws_s3_bucket_public_access_block.public_access_block,
  ]
}
```

`aws_s3_bucket_acl`

- Sets the S3 bucket ACL to allow public read access.

```
resource "aws_s3_bucket_policy" "bucket_policy" {
  bucket = aws_s3_bucket.bucket1.bucket
  policy = jsonencode(
    {
      "Version" : "2012-10-17",
      "Statement" : [
        {
          "Sid" : "PublicReadGetObject",
          "Effect" : "Allow",
          "Principal" : "*",
          "Action" : "s3:GetObject",
          "Resource" : "${aws_s3_bucket.bucket1.arn}/*"
        }
      ]
    }
  )}
```

`aws_s3_bucket_policy`

- Defines a policy allowing public read access to objects in the S3 bucket.

```
resource "aws_s3_bucket_website_configuration"
"bucket_website_configuration" {
  bucket = aws_s3_bucket.bucket1.id

  index_document {
    suffix = "index.html"
  }
}
```

`aws_s3_bucket_website_configuration`

- Configures the S3 bucket to act as a static website, defining the default index document.

CloudFront Distribution

```
locals {
  s3_origin_id = "myS3Origin"
}

resource "aws_cloudfront_distribution" "distribution" {
  enabled             = true
  is_ipv6_enabled     = true
  default_root_object = "index.html"

  origin {
    domain_name = aws_s3_bucket.bucket1.bucket_regional_domain_name
    origin_id   = local.s3_origin_id
  }

  viewer_certificate {
    cloudfront_default_certificate = true
  }

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

  default_cache_behavior {
    cache_policy_id        = "4135ea2d-6df8-44a3-9df3-4b5a84be39ad"
    viewer_protocol_policy = "redirect-to-https"
    allowed_methods        = ["DELETE", "GET", "HEAD", "OPTIONS", "PATCH", "POST", "PUT"]
    cached_methods         = ["GET", "HEAD"]
    target_origin_id       = local.s3_origin_id
  }
}
```

`aws_cloudfront_distribution`

- Creates a CloudFront distribution to globally distribute and serve the static content with low latency.
- Uses the S3 bucket as the origin for CloudFront.
- Configures a default cache behavior to redirect HTTP to HTTPS and allows specified HTTP methods.
- Uses the default CloudFront SSL certificate.
- Enables IPv6

Outputs

```
output "website_url" {
  description = "Website URL (HTTPS)"
  value       = aws_cloudfront_distribution.distribution.domain_name
}

output "s3_url" {
  description = "S3 hosting URL (HTTP)"
  value       = aws_s3_bucket_website_configuration.bucket_website_configuration.website_endpoint
}
```

- Provides the URLs for the static website, both through CloudFront (HTTPS) and S3 (HTTP).

Terraform

Run the following command to initialize Terraform configuration:

- **terraform init**

Run the following command to apply the Terraform configuration and deploy the infrastructure:

- **terraform apply --auto-approve**

Run the following command to destroy the resources:

- **terraform destroy**

Meeting Requirements:

High Availability

- Using Amazon S3 for static content hosting is highly available by design.
- I've also integrated Amazon CloudFront for content delivery, which enhances availability by distributing content globally.

Global Latency Avoidance

- CloudFront helps reduce latency by caching content at edge locations worldwide. This is a good approach for serving content with low latency to visitors from different geographic locations.

Autoscaling for Increased Visitors:

- The current setup focuses on the frontend (S3 and CloudFront) and does not include any backend or server-side processing. For static websites, this is sufficient.

Infrastructure as Code

- Terraform script fulfills the requirement of using Infrastructure as Code.

Conclusion

This Terraform script deploys a robust and scalable architecture for hosting a static website on AWS. Using S3 for storage and CloudFront for content delivery, ensures high availability, global reach, and low-latency access. The infrastructure-as-code approach enhances reproducibility, scalability, and ease of management for the static website infrastructure on AWS.