Host a Website

on AWS with Terraform (Infrastructure as Code)

HashiCorp
Terraform

Lucas Nebelung
42304499

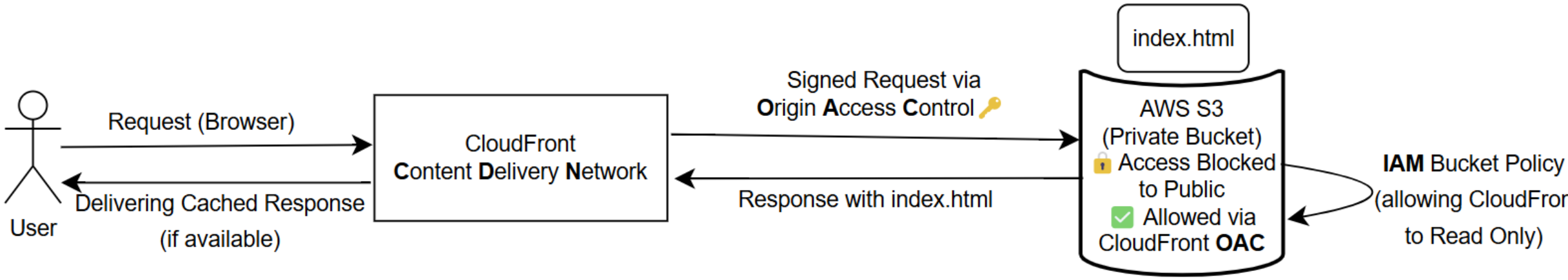
Overview

- **Goal**: Host a static "Hello World" webpage on *AWS* using *IaC*

- **Requirements**:
  - Global availability
  - Low latency
  - Autoscaling

  (optional) secure, cost-efficient, modular

Solution: Architecture



Why this Architecture?

CloudFront + S3 REST API + OAC (Private Bucket)

- More secure
- BUT: More complex
  -> requires OAC, bucket policy and CloudFront Config

CloudFront + S3 Website Hosting (Public Bucket)

- Less secure
  -> Bucket must be public for CloudFront to fetch content
- Simpler

Why S3 + CloudFront?

- **S3**: Simple, cost-effective storage for static content.
- **CloudFront**: Provides a global edge network (CDN) for low-latency delivery.
- **Private Bucket + OAC**: Ensures the bucket is *not* publicly accessible; only CloudFront can read objects.
- **Autoscaling**
  S3 and CloudFront automatically scale based on traffic load. No manual configuration needed for extra servers.

Why Terraform?

- **Infrastructure as Code (IaC)**: All resources defined in *.tf* files
- **Version Control**: Easily track changes and collaborate.
- **Consistency**: The same configuration can be applied across environments
- **Cloud-Agnostic**: Could adapt to other clouds in the future if needed.

Code Explanation I: S3 Bucket

- 1) selecting AWS region to be Frankfurt (Germany)
- 2)
  - Creates a private S3 bucket named *iu-test-bucket (*No ACL or website hosting block so the bucket defaults to private)

  - Uploads local *index.html* into the newly created private S3 bucket.

```
############################################################
###### Hosting a simple webpage on AWS using Terraform #######
############################################################


#####################################################
# 1) Terraform Configuration & Provider Settings
#####################################################

provider "aws" {
  region = "eu-central-1"
}


#####################################################
# 2) create a private S3 Bucket
#####################################################

# create an S3 bucket with name
resource "aws_s3_bucket" "private_bucket" {
  bucket = "iu-test-bucket"
}


# upload the index.html file to the S3 bucket
resource "aws_s3_object" "index" {
  bucket = aws_s3_bucket.private_bucket.id
  key    = "index.html"       # name of the file in S3
  source = "index.html"       #local file
  content_type = "text/html"
}
```

Code Explanation II: OAC

- Defining **O**rigin **A**ccess **C**ontrol (OAC) for the CloudFront Distribution
- Setting "*always*" and "*sigv4*" for maximum security

```
################################################
# 3) Origin Access Control (OAC)
################################################

resource "aws_cloudfront_origin_access_control" "oac" {
  name                              = "my-s3-oac"
  origin_access_control_origin_type = "s3"
  signing_behavior                  = "always"
  signing_protocol                  = "sigv4"
}
```

Code Explanation III: CloudFront Distribution

- Creating **CloudFront Distribution** and setting *index.html* as default
- Specifying origin to our "*private-s3-origin*" and assigning previously specified **OAC**
- Setting *default_cache_behavior* to ["GET", "HEAD"] as sufficient for our use case
- Forcing all traffic to use HTTPS, improving security
- Using simple settings for *forwarded_values*, *restrictions* and *viewer_certificate* (so website available via *.cloudfront.net)

```
################################################
# 4) Cloudfront Distribution with S3 Bucket as Origin
################################################

#create a cloudfront distribution first
resource "aws_cloudfront_distribution" "cdn" {
  enabled = true
  default_root_object = "index.html"
  is_ipv6_enabled = true
  wait_for_deployment = true

# Where does Cloudfront pull content from?
  origin {
    domain_name = aws_s3_bucket.private_bucket.bucket_regional_domain_name
    origin_id = "private-s3-origin"
    origin_access_control_id = aws_cloudfront_origin_access_control.oac.id
  }
#set cache behaviors
  default_cache_behavior {
    target_origin_id = "private-s3-origin"
    viewer_protocol_policy = "redirect-to-https"
    allowed_methods = ["GET", "HEAD", ]
    cached_methods = ["GET", "HEAD",]
    compress = true

# simple default setting for query string, cookies, and headers
    forwarded_values {
      query_string = false
      cookies {
        forward = "none"
      }
    }
  }

#no geo or specific HTTPS, SSL/TLS encryption restrictions
  restrictions {
    geo_restriction {
      restriction_type = "none"
    }
  }
   viewer_certificate {
    cloudfront_default_certificate = true
  }

}
```

Code Explanation IV: IAM Bucket Policy

- Fetching current AWS account
- Creating **IAM** policy document
  -> allows CloudFront to READ (GET) only
  -> allows CloudFront to access all objects inside specified bucket
- *Principals* ensures only CloudFront (and no other Service) can access
- *Condition* enforces requests come from our AWS Account
- Assigning created Bucket Policy to our Bucket

```
################################################
# 5) Bucket Policy to Allow CloudFront Read
################################################

# get the current AWS account identity and assign it to "curre
data "aws_caller_identity" "current" {}

# create a IAM policy document
data "aws_iam_policy_document" "bucket_policy_doc" {
  statement {
    sid = "AllowCloudFrontServicePrincipal"  #Statement ID
    effect = "Allow"
    actions = ["s3:GetObject"]               #Cloudfront can on
    resources = [
      "${aws_s3_bucket.private_bucket.arn}/*"    #allows Cloud
    ]

    # only allow CloudFront to access the S3 private bucket
    principals {
      type        = "Service"
      identifiers = ["cloudfront.amazonaws.com"]
    }

    # only allow access from the same AWS account
    condition {
      test     = "StringEquals"
      variable = "AWS:SourceAccount"
      values   = [data.aws_caller_identity.current.account_id]
    }
  }
}

# apply the policy to the bucket
resource "aws_s3_bucket_policy" "bucket_policy" {
  bucket = aws_s3_bucket.private_bucket.id     #specifies whic
  policy = data.aws_iam_policy_document.bucket_policy_doc.json
}
```

Code Explanation V: Output

- Ensuring Terraform states the *domain* of Website and corresponding bucket after deployment

```
################################################
# 6) Outputs
################################################

# Output the CloudFront domain name
output "cloudfront_domain_name" {
  description = "The Website can be accesed with this URL"
  value       = aws_cloudfront_distribution.cdn.domain_name
}

# Output the S3 bucket name
output "bucket_name" {
  value = aws_s3_bucket.private_bucket.bucket
}
```

Frameworks used …

- **AWS** Documentation
  https://docs.aws.amazon.com/
  -> link to all relevant services including S3, CloudFront, IAM
- **Terraform** Documentation
  https://developer.hashicorp.com/terraform/docs
  -> link to Terraform Installation, commands etc