Finalization Phase: Hosting a Simple Webpage on AWS

Abstract

This project successfully deploys a globally accessible and highly available webpage using AWS. Amazon S3 stores the webpage's static content, while Amazon CloudFront distributes the content globally to minimize latency and enhance performance. Terraform was used as the Infrastructure as Code (IaC) tool to automate the resource setup, ensuring consistency and scalability. Key features include caching optimization, secure delivery with Origin Access Identity (OAI), and HTTPS enforcement. Feedback led to simplifications such as removing Route 53 for DNS, further improving cost-efficiency and ease of use. The project demonstrates AWS's capability to create cost-effective, scalable, and secure solutions for static website hosting, with potential future enhancements for custom domains, dynamic content, and monitoring.

Keywords: AWS, Amazon S3, CloudFront, Terraform, Infrastructure as Code, scalability, high availability, security, global accessibility.

Introduction

The purpose of this project is to create a webpage that is accessible worldwide, highly available, and able to handle growing traffic demands. The webpage, which displays a restaurant menu, serves as a starting point for exploring AWS's capabilities in hosting and delivering static content globally.

AWS was chosen for its:

- **Scalability**: Automatically handles increased traffic without manual intervention (AWS, 2024a).
- **Global Reach**: Ensures low-latency access through its global infrastructure (AWS, 2024a).
- **Cost-Efficiency**: Offers free-tier options for lightweight use cases, making it suitable for projects like this (AWS, 2024a).

Terraform, an IaC tool, automates the setup and management of AWS resources (HashiCorp, n.d.). By using Terraform, this project avoids manual configuration errors and ensures consistent resource provisioning (HashiCorp, n.d.).

This final report highlights the three phases of the project:

1. **Conception Phase**: Planning and designing the architecture.
2. **Development Phase**: Implementing the architecture and testing functionality.
3. **Finalization Phase**: Refining the setup, incorporating feedback, and documenting the project.

Phases Overview

1. Conception Phase

In the first phase, the project scope and goals were defined:

- **Goal**: Create a webpage that is:
  - Always available.
  - Accessible worldwide with minimal delay.
  - Scalable to handle increasing visitors.
- **AWS Services Chosen**:
  - **Amazon S3**: A reliable service for hosting static files (AWS, 2024a).
  - **Amazon CloudFront**: Provides fast, global delivery through caching (AWS, 2024b).
  - **Route 53**: Initially planned for DNS management but removed later to simplify the architecture (AWS, 2024c).

2. Development Phase

This phase involved implementing the planned architecture:

- **Terraform Deployment**: Terraform was used to create and configure the S3 bucket, CloudFront distribution, and related resources.
- **Security**: Origin Access Identity (OAI) was applied to restrict access to the S3 bucket, ensuring that only CloudFront could fetch content.
- **Simplifications**: Route 53 was removed to save costs and rely on CloudFront's default DNS capabilities.

3. **Finalization Phase**

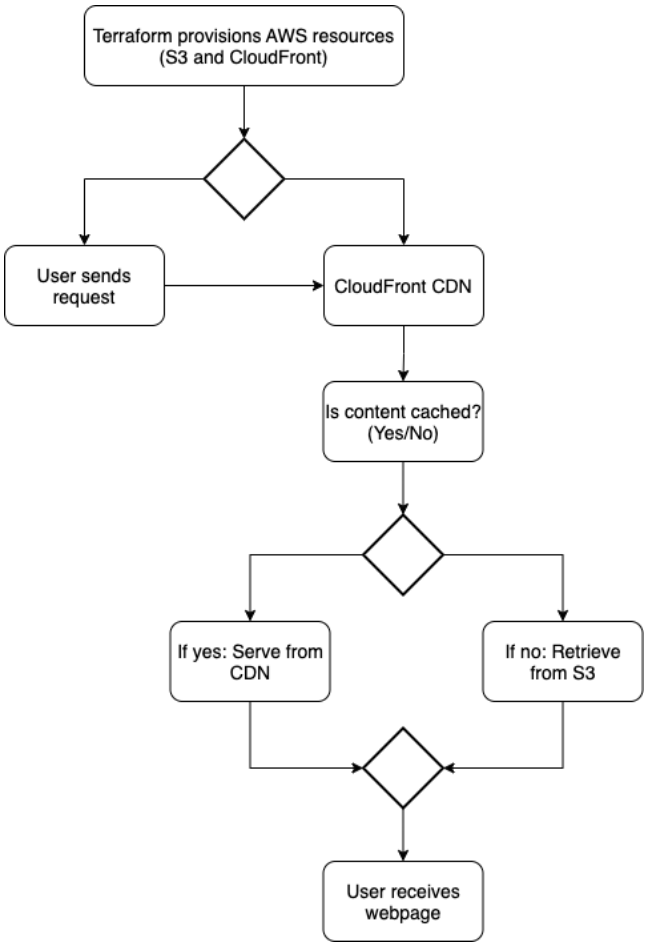
During this phase:

1. **Improvements**: Feedback was incorporated to optimize caching policies and enhance security.
2. **Testing**: The setup was thoroughly tested to ensure reliability and performance.
3. **Documentation**: Detailed diagrams, configurations, and results were prepared for final submission.

AWS Architecture

- **Architecture Components**
  1. **Amazon S3**:
     - Hosts the static files, including HTML and images.
     - Configured with private access, linked to CloudFront via OAI for secure delivery (AWS, 2024a).
  2. **Amazon CloudFront**:
     - Acts as a Content Delivery Network (CDN) to deliver content globally with low latency (AWS, 2024b).
     - Improves user experience by caching files at edge locations closer to users (AWS, 2024b).
  3. **Terraform**:
     - Automates resource provisioning and ensures consistency (HashiCorp, n.d.).
- **Architecture Flow**
  1. The user requests the webpage through their browser.
  2. CloudFront routes the request to the nearest edge location.
  3. If the content is cached, CloudFront serves it directly; otherwise, it retrieves it from the S3 bucket.
  4. The webpage is displayed to the user.

Diagram:



Terraform Configuration

Terraform automates the deployment of AWS resources. Below are key configurations:

1. AWS Provider

The provider specifies the AWS region (eu-central-1) where resources will be created:

```
# Define the AWS Provider
provider "aws" {
  region = "eu-central-1"
}
```

2. S3 Bucket

The S3 bucket stores static files securely. It is configured with a private access policy:

```
# Create the S3 Bucket
resource "aws_s3_bucket" "restaurant_menu_webpage" {
  bucket = "restaurant-menu-webpage"
  acl    = "private"  # Set to private since access will be controlled via CloudFront
}
```

3. CloudFront Distribution

CloudFront delivers content globally, using HTTPS to secure communications:

```
# Configure CloudFront Distribution for Global Access
resource "aws_cloudfront_distribution" "restaurant_menu_distribution" {
  origin {
    domain_name = aws_s3_bucket.restaurant_menu_webpage.bucket_regional_domain_name
    origin_id   = "S3-restaurant-menu-webpage"

    s3_origin_config {
      origin_access_identity = aws_cloudfront_origin_access_identity.restaurant_menu_identity.cloudfront_access_identity_path
    }
  }

  enabled             = true
  is_ipv6_enabled     = true
  default_root_object = "index.html"

  # Default Cache Behavior for CloudFront
  default_cache_behavior {
    target_origin_id       = "S3-restaurant-menu-webpage"
    viewer_protocol_policy = "redirect-to-https"

    allowed_methods = ["GET", "HEAD"]
    cached_methods  = ["GET", "HEAD"]

    forwarded_values {
      query_string = false
      cookies {
        forward = "none"
      }
    }

    min_ttl     = 0
    default_ttl = 3600
    max_ttl     = 86400
  }

  # Viewer Certificate for HTTPS (using default CloudFront certificate)
  viewer_certificate {
    cloudfront_default_certificate = true
  }

  restrictions {
    geo_restriction {
      restriction_type = "none"
    }
  }
```

4. Outputs

Outputs provide important information such as the CloudFront URL:

```
# Output CloudFront Distribution URL for easy access
output "cloudfront_url" {
  value       = "https://dnbv8i18z5791.cloudfront.net"
  description = "The CloudFront distribution URL for the website"
}
```

Complete Code: The full Terraform configuration is available in the GitHub repository: GitHub Repository.

Security Measures

1. **Origin Access Identity (OAI)**

   The **Origin Access Identity (OAI)** ensures that the S3 bucket is not directly accessible by users (AWS, 2024a). Instead, it allows only CloudFront to retrieve the content from the bucket securely (AWS, 2024a). This configuration prevents users from bypassing CloudFront and accessing the content directly from S3 (AWS, 2024a).

   - **Key Benefit**: Protects the S3 bucket from unauthorized access while ensuring that CloudFront can deliver the content efficiently (AWS, 2024a).

   For example, if someone attempts to access the S3 bucket URL directly, they will receive an "Access Denied" error unless the request comes through CloudFront.

2. **HTTPS Enforcement**

   HTTPS is enforced in the CloudFront configuration to ensure secure communication between users and the webpage (AWS, 2024b). By redirecting all HTTP requests to HTTPS, the system guarantees that data is transmitted securely (AWS, 2024b).

   - **Key Benefit**: HTTPS protects user data from being intercepted, especially over public networks (AWS, 2024b). It also improves user trust by showing the "secure" lock symbol in the browser (AWS, 2024b).

3. **S3 Bucket Policy**

   A custom **S3 bucket policy** was applied to further secure access. This policy explicitly blocks public access to the bucket and grants read-only access to the CloudFront OAI (AWS, 2024a).

   - **Key Benefit**: The policy ensures that only CloudFront can fetch the content stored in the bucket, reducing the risk of data leaks or unauthorized downloads (AWS, 2024a).

Testing and Results

- **Deployment Steps:**
  1. Used terraform apply to provision AWS resources.
  2. Verified the infrastructure via AWS Management Console and Terraform outputs.
- **Testing Methodology:**
  1. Tested webpage accessibility through the CloudFront URL from multiple geographic regions.
  2. Monitored cache performance by updating content and observing propagation delays.
- **Results:**
  - Global accessibility achieved with low latency.
  - Cache hit ratio increased after content updates.
  - Security configurations successfully restricted direct access to S3.

CloudFront URL: https://dnbv8i18z5791.cloudfront.net

Reflection and Improvements

- **Feedback Implementation:**
  1. **Simplified Architecture by Removing Route 53**:
  - At first, Route 53 was included for managing DNS and allowing a custom domain name. However, it was removed to save costs and reduce complexity. Instead, the default CloudFront domain was used. This change made the setup easier to manage and cheaper.
  - **Result**: The architecture became simpler, especially for small projects, and it fit better within the AWS Free Tier limits (AWS, 2024a).
  2. **Improved Caching Policies**:
  - The caching settings in CloudFront were updated to make the website faster and lower costs. By increasing the time-to-live (TTL) for cached files, fewer requests were sent to the S3 bucket (AWS, 2024b).
  - **Result**: This reduced waiting time for users and saved costs by lowering the load on S3 (AWS, 2024b).

- **Lessons Learned:**
  1. **Automation with Terraform is Helpful**:
  - Using Terraform to create and manage resources made the process faster and less error-prone. Writing the setup as code made it easy to repeat and update (HashiCorp, n.d.).
  - **What I Learned**: Terraform is very useful when building cloud systems, especially if changes need to be made later (HashiCorp, n.d.).
  2. **Security is Very Important**:
  - Setting up an Origin Access Identity (OAI) and restricting S3 access showed how important security is in cloud projects. Without these, the S3 bucket could have been accessed by anyone, leading to data leaks (AWS, 2024a; AWS, 2024b).
  - **What I Learned**: Security measures like OAI and HTTPS protect the content and build trust with users (AWS, 2024a; AWS, 2024b).

3. **Balancing Cost and Performance**:
   - Removing Route 53 and improving caching policies taught me how to balance saving money and improving performance. Sometimes, a simpler setup is better for small projects.
4. **AWS Services are Flexible**:
   - AWS offers many options, making it easy to adjust the setup based on project needs. This flexibility made it possible to start small but plan for future growth.

Future Scope

While the webpage is fully functional, there are ways to improve it in the future to make it more professional.

1. **Custom Domain Integration**

   Adding a custom domain using Amazon Route 53 would make the webpage look more professional and easier to remember (e.g., www.restaurantmenu.com). Route 53 can link the domain to the CloudFront distribution, and HTTPS can secure the connection (AWS, 2024b).

   - **Benefit**: Improves branding and user trust.

2. **Enhanced Monitoring**

   AWS CloudWatch can track performance and usage, helping to detect errors or traffic spikes (AWS, 2024b). This would allow proactive maintenance to keep the webpage running smoothly (AWS, 2024b).

   - **Benefit**: Provides insights to improve performance and reliability (AWS, 2024b).

3. **Dynamic Features**

   AWS Lambda can be used to add interactivity, such as a contact form or reservation system (AWS, 2024d). This would make the webpage more engaging for users (AWS, 2024d).

   - **Benefit**: Adds useful features without needing a dedicated server (AWS, 2024d).

Packaging and Submission

Included Files:

- Terraform scripts (main.tf).
- Static website files (index.html, images).
- Architecture diagram.
- Final report (PDF).

Deliverables:

- Terraform scripts.
- Static website files.
- Architecture diagram.
- Testing results and final report.

References:

Amazon Web Services (AWS). (2024a). *What is Amazon S3?*. Retrieved from https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

Amazon Web Services (AWS). (2024b). *What is Amazon CloudFront?*. Retrieved from https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html

Amazon Web Services (AWS). (2024c). *What is Route 53?*. Retrieved from https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/Welcome.html

Amazon Web Services (AWS). (2024d). *What is AWS Lambda?*. Retrieved from https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

HashiCorp. (n.d.). *What is Terraform?*. Retrieved from https://developer.hashicorp.com/terraform/intro