Making of: Cloud Architecture for simple webpage on AWS

This report outlines my journey of designing, implementing, and refining a simple, globally accessible website on Amazon Web Services (AWS) using Terraform as an Infrastructure-as-Code (IaC) tool. Although the project's goal—hosting a static "Hello World" page—appears straightforward, I quickly discovered the complexities and decisions involved in configuring AWS securely, cost-effectively, and according to best practices.

1. Project Background & Motivation

Initially, I set out to learn about Terraform, an open-source IaC tool, by working through videos, tutorials, and documentation to understand how to install and run it locally. This involved setting up the AWS CLI, generating IAM credentials, and integrating Terraform with Visual Studio Code. Along the way, I also explored the AWS Management Console, which can be overwhelming due to the sheer number of services. However, I decided to keep things minimal by focusing on two key AWS components for my static website:

- **Amazon S3** (Simple Storage Service)
- **Amazon CloudFront** (Content Delivery Network)

The simplicity of S3 (for object storage) and CloudFront (for global distribution) makes them ideal for hosting static content. Both services inherently scale to handle traffic spikes, meet global performance requirements, and align with the "pay-as-you-go" model that helps avoid large upfront costs.

2. Technical Approach & Challenges

2.1. Getting Started with Terraform

I installed Terraform locally and initialized a small project folder that included my index.html file.

My first trials involved creating S3 buckets and experimenting with terraform apply to see how changes reflect in the AWS console.

I encountered minor hurdles with Git due to large .terraform directories, which taught me the importance of a proper .gitignore file.

2.2. Selecting a Secure Architecture

From the outset, I realized there were multiple ways to connect CloudFront and S3:

- S3 Website Endpoint (Public): Simpler, but the bucket is publicly accessible.
- S3 REST Endpoint + Origin Access Control (OAC): A more modern approach that keeps the bucket private and grants CloudFront permission to pull content securely.

I opted for the REST Endpoint + OAC strategy, aiming for a private S3 bucket with a bucket policy that only allows access from CloudFront. This approach is considered best practice, and although it introduced more complexity (e.g., IAM policy documents, 403 “Access Denied” errors to troubleshoot), it offered stronger security and forced me to learn about AWS policies and Terraform’s syntax for data references.

2.3. Overcoming Errors

Like many first-time AWS users, I encountered 403 Forbidden errors when CloudFront attempted to retrieve objects from my bucket. By exploring the AWS documentation and cross-referencing best practices, I discovered:

- The bucket must have a policy explicitly granting s3:GetObject permissions to CloudFront’s service principal (cloudfront.amazonaws.com).
- The Origin Access Control (OAC) ID must be referenced in the s3_origin_config or origin block in Terraform.
- The code also required a forwarded_values block to avoid a CloudFront “InvalidArgument: The parameter ForwardedValues is required” error.

Resolving these issues deepened my understanding of AWS authentication and Terraform’s modular structure.

3. “Making Of” & Key Takeaways

- Installation & Setup: Learned how to install Terraform, the AWS CLI, and connect them to my AWS account through IAM credentials.
- Exploring AWS Services: Familiarized myself with S3, CloudFront, and IAM—realizing each service has multiple configuration patterns (public vs. private, OAC vs. OAI, etc.).
- Security & Budget: Enabled a budget alert on AWS to prevent unforeseen charges beyond the free tier.
- Chose a private bucket approach for stronger security, though it introduced more complexity and a lot more 403 Error Codes, which were tricky to solve at first.

- Terraform Trials: Wrote .tf files incrementally, overcame Git issues, and discovered the importance of a clean .gitignore.

4. Conclusion

This project emphasized that even a “simple” static website can involve numerous settings—from bucket policies and origin access controls to Terraform’s code structure. Nonetheless, it demonstrated how quickly AWS and Terraform can scale and secure a global site once properly configured. The experience sharpened my understanding of Infrastructure-as-Code principles, AWS best practices, and the interplay between a local development environment and cloud infrastructure. Going forward, I feel more confident handling more complex AWS deployments!