AWS Static Website Hosting

Cloud Architecture Implementation for Static Website

Negin Hezarjaribi

INTERNATIONAL UNIVERSITY OF APPLIED SCIENCES

Course Name: Cloud Programming

Course Code: DLBSEPCP01_E

Date: 13 March 2025

TABLE OF CONTENTS

How Deploy a Static Website on AWS ?

Introduction

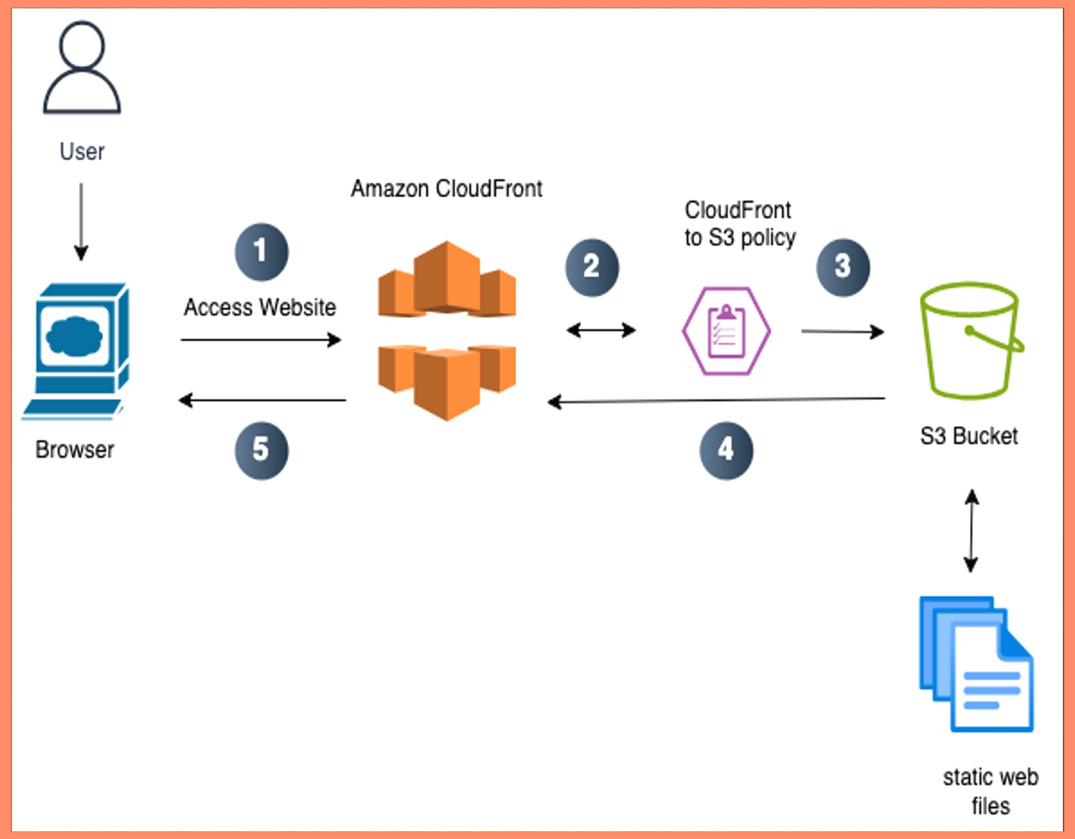
In this project, we aim to deploy a static website on AWS (Amazon Web Services) using a cloud architecture that ensures high availability, scalability, and security. This means the website will be accessible worldwide with minimal delay, handle traffic spikes efficiently, and follow AWS security best practices.

To achieve this, we utilize the following resources:

- **Amazon S3**: Stores the static website files (HTML, CSS).
- **Amazon CloudFront (with Origin Access Identity):** Distributes content globally via a CDN and restricts direct access to the S3 bucket, enhancing security.
- **Terraform:** An Infrastructure as Code (IaC) tool to automate resource provisioning.

In this presentation, we will explore the components of this project, their architecture, and the step-by-step implementation process.

Architecture Diagram: Request Flow in CloudFront + OAI



1-> User requests "index.html" from CloudFront.

2,3 -> CloudFront forwards the request to S3 using OAI.

4-> S3 checks if "index.html" exists. If yes, it returns the file to CloudFront.

5-> CloudFront caches the response and serves it to the user.

The browser receives "index.html" and renders the webpage.

Project Implementation Steps

1. Set Up AWS Credentials

- Create an IAM user with **programmatic access**.
- Attach necessary policies :

  AmazonS3FullAccess, CloudFrontFullAccess.
- Save the **Access Key** and **Secret Key** for Terraform.

2. Configure AWS CLI and Install Terraform

Install AWS CLI and configure it with IAM credentials:

```
aws configure
```

Install Terraform and verify installation:

```
terraform version
```

3. Create and Configure S3 Bucket

- Create an S3 bucket
- (e.g., aws-static-site-2025-NeginHZ).
- Block all public access to enhance security.
- Do NOT enable static website hosting.
- Configure S3 bucket policy to allow access only from CloudFront OAI.
- Defines error and index documents.

Project Implementation Steps

4. Set Up OAI

- Create an Origin Access Identity (OAI) for CloudFront.
- Attach the OAI to the CloudFront distribution.
- Modify S3 bucket policy to grant read access to CloudFront OAI only.

5. Configure CloudFront as CDN

- Create a CloudFront distribution.
- Set the S3 bucket (via OAI) as the origin.
- Define caching and security settings and custom error responses.

6. Deploy Infrastructure Using Terraform

- Initialize Terraform:

  terraform init

- Validate configuration:

  terraform validate

- Preview changes:

  terraform plan

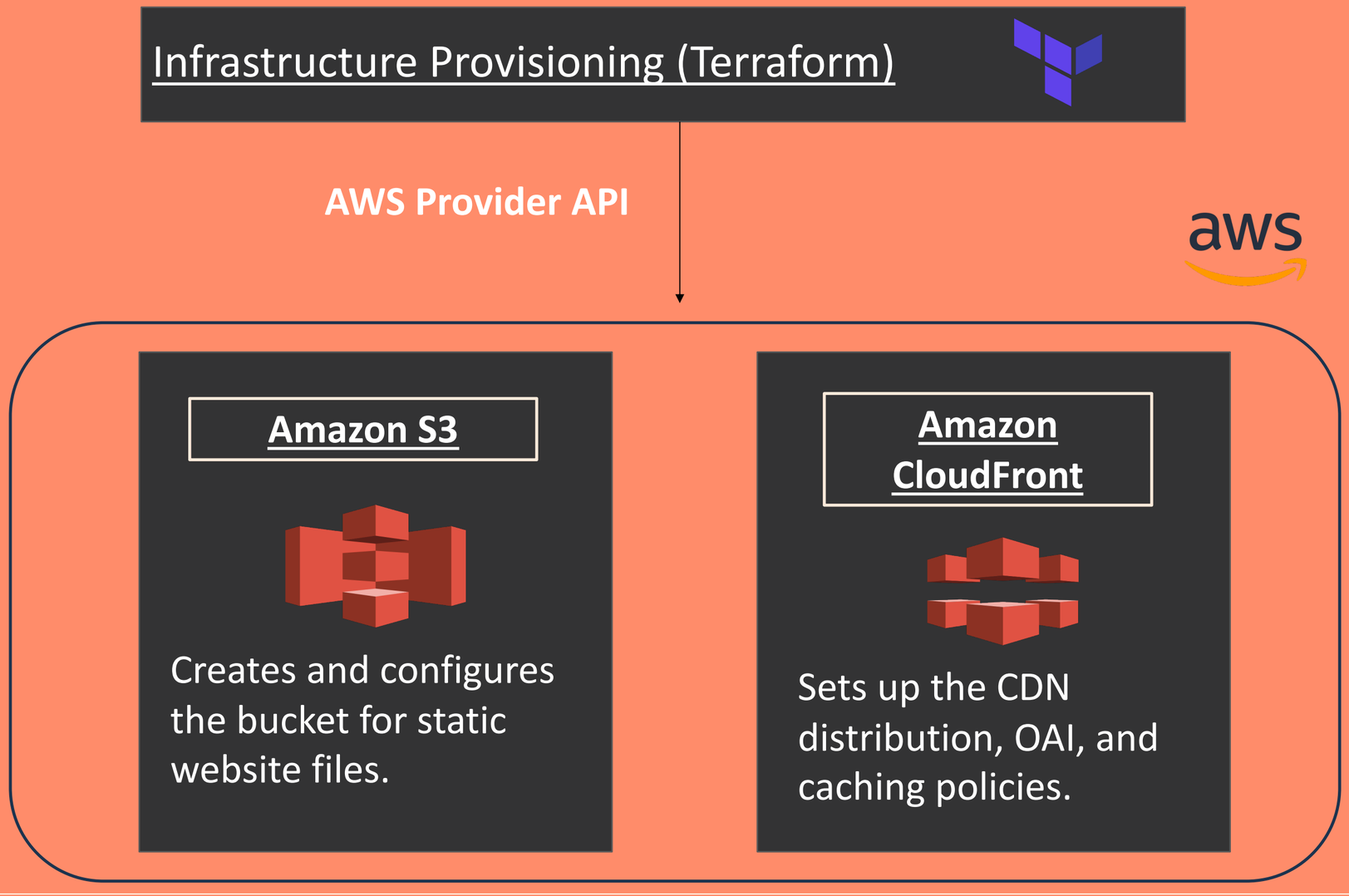
- Deploy infrastructure:

  terraform apply -auto-approve

- Retrieve output values:

  terraform output

Terraform manages the provisioning of key AWS resources



Terraform Code Structure

Our Project directory is structured as follows:

```
aws-static-site/
├── terraform/                # Terraform configuration files
│   ├── provider.tf           # AWS provider setup
│   ├── variables.tf          # Variables for AWS region and S3 bucket name
│   ├── main.tf               # S3 bucket setup (private, OAI access only)
│   ├── cloudfront.tf         # CloudFront distribution setup
│   ├── outputs.tf            # Output values for website URLs
│   ├── versions.tf           # Required Terraform and AWS provider versions
│   └── terraform.tfvars      # Default values for variables (e.g., bucket name, AWS region)
├── website/                  # Static website files
│   ├── index.html            # Homepage
│   ├── style.css             # Website styles
│   └── error.html            # Custom error page (CloudFront response)
```

Key Terraform Configuration Files

main.tf (S3 Bucket Configuration)

This file contains the main Terraform configurations for your infrastructure:

- **S3 Bucket Creation:** It creates an S3 bucket to store your static website files.
- **Public Access Block:** It disables public access to the S3 bucket to enhance security.
- **Bucket Policy:** It sets a policy to allow access to the bucket only from CloudFront's Origin Access Identity (OAI), ensuring secure content delivery.

cloudfront.tf (CDN Configuration)

This file defines the CloudFront distribution:

- **OAI Creation:** It creates an Origin Access Identity to securely access the S3 bucket.
- **CloudFront Distribution:** It sets up the CloudFront distribution, linking it to the S3 bucket and configuring settings like caching, SSL certificates, and error responses.

provider.tf (AWS Provider Configuration)

This file configures the AWS provider for Terraform, specifying the AWS region where your resources will be created.

Key Terraform Configuration Files

terraform.tfvars (User-Defined Variables)

This file provides default values for the variables defined in variables.tf, such as the AWS region and S3 bucket name, allowing for easy customization of your deployment.

variables.tf (Configuration Management)

This file defines the variables used in your Terraform configurations, such as the AWS region and the S3 bucket name.

version.tf (Terraform Versionin)

This file specifies the required Terraform version and provider versions to ensure compatibility and consistency in your infrastructure deployments.

outputs.tf (Displaying Deployment Results)

This file specifies the outputs of your Terraform configuration, such as the CloudFront distribution URL, which can be used to access your website.

Deployment Results & Outputs

Terraform Outputs:

❖CloudFront Distribution URL: output "cloudfront_url"

After successful deployment, the Terraform output provides the following URLs:

cloudfront_url = d3n1b7jgbt32ke.cloudfront.net

These URLs can be used to access the deployed static website.

Challenges and Architectural Decisions

Initial Plan: IAM + CloudFront

We planned to secure our static website using IAM policies to control access, allowing only CloudFront to fetch content from the S3 bucket. However, this approach left the S3 bucket public, meaning users could still access files directly via S3 URLs.

Issues with IAM Policies

- ❖Security Risk: Direct S3 access bypassed CloudFront.
- ❖Limited Control: IAM policies couldn’t fully block direct S3 access.
- ❖Public Exposure: S3 URLs remained accessible.

Challenges and Architectural Decisions

Final Solution: CloudFront + OAI

We switched to CloudFront with Origin Access Identity (OAI), which ensures only CloudFront can access the S3 bucket. This made the bucket private, blocking all direct access.

Benefits of OAI

- Enhanced Security: S3 is now private; direct access is blocked.
- Full Access Control: Only CloudFront can fetch content.
- Improved Performance: CloudFront caches content globally.
- Cost Efficiency: Reduced direct S3 requests lower costs.

By replacing IAM policies with CloudFront + OAI, we achieved a secure, scalable, and optimized architecture, ensuring only CloudFront can access the S3 bucket.

Conclusion

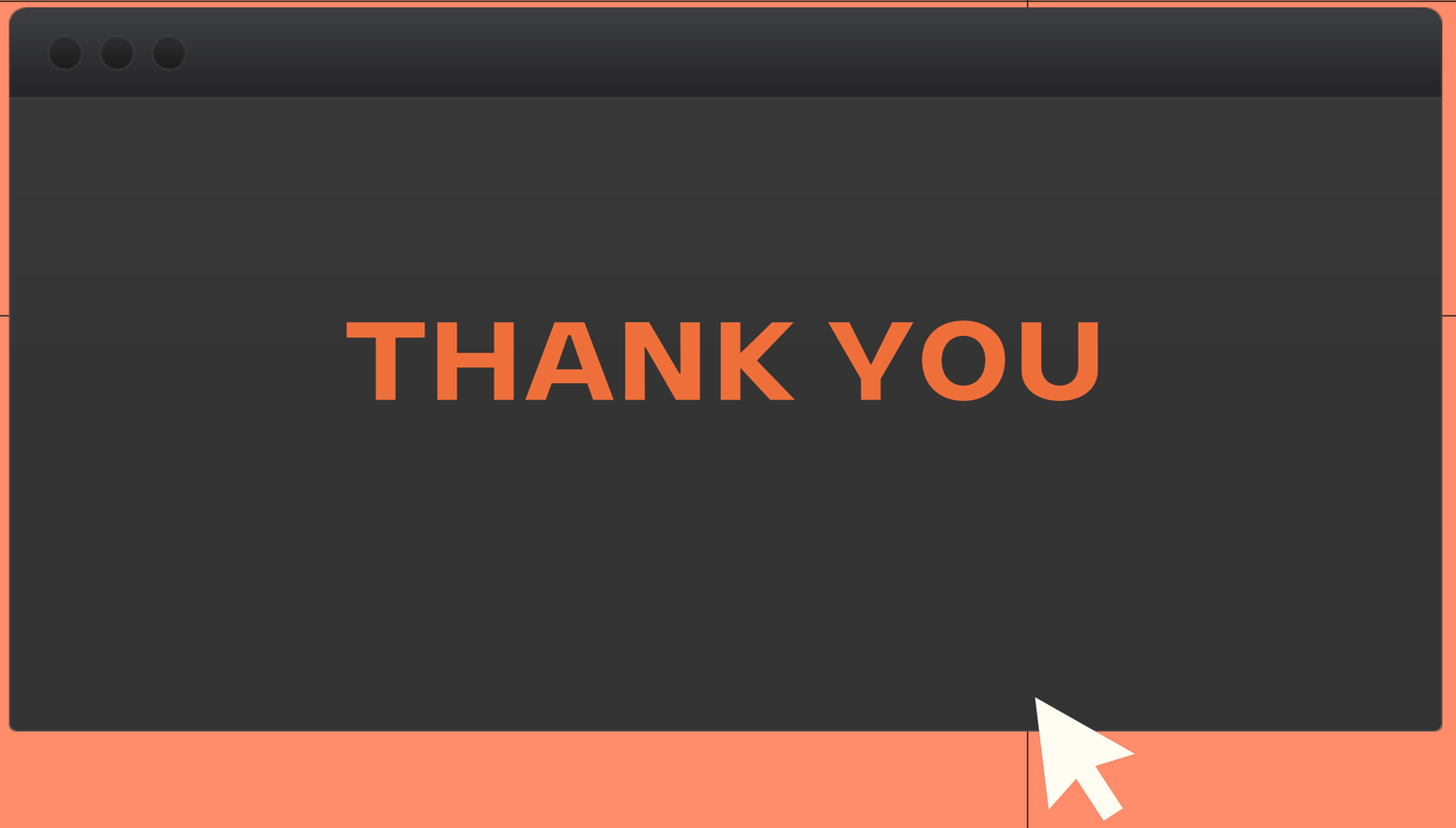
In this project, we successfully deployed a secure, highly available, and globally optimized static website using AWS services. By integrating Amazon S3 for storage, CloudFront as a Content Delivery Network (CDN), and Origin Access Identity (OAI) for security, we ensured restricted access to S3 and improved content delivery performance.

Using Terraform, we automated the infrastructure deployment, making it easily replicable and manageable. This approach enhances security, reduces latency, and provides a scalable solution for hosting static websites.

In summary, our architecture achieves:

- Stronger security by preventing direct S3 access.
- Improved performance through CloudFront caching.
- Efficient scalability to handle varying traffic loads.

This solution aligns with AWS best practices, ensuring a reliable and cost-effective approach for static website hosting.


THANK YOU

References

- Amazon Web Services. (n.d.). *Amazon S3*. Retrieved March 5, 2025, from https://aws.amazon.com/s3/
- Amazon Web Services. (n.d.). *Amazon CloudFront*. Retrieved March 5, 2025, from https://aws.amazon.com/cloudfront/
- Amazon Web Services. (n.d.). *AWS Identity and Access Management (IAM)*. Retrieved March 5, 2025, from https://aws.amazon.com/iam/
- HashiCorp. (n.d.). *Terraform by HashiCorp*. Retrieved March 5, 2025, from https://www.terraform.io/
- Amazon Web Services. (n.d.). *AWS Documentation*. Retrieved March 5, 2025, from https://docs.aws.amazon.com/
- diagrams.net. (n.d.). *diagrams.net - Free Online Diagram Software*. Retrieved March 5, 2025, from https://app.diagrams.net/
- Amazon Web Services. (n.d.). *AWS Architecture Icons*. Retrieved March 5, 2025, from https://aws.amazon.com/architecture/icons/