Project Report – Cloud Computing

Shortify

(URL shortener browser extension with serverless architecture in AWS)

Project By:

Supragya Sharma
E22CSEU0796
Course – B. TECH
Branch – Computer Science and Engineering
Batch – 27
Term/Year – 3rd Year / 6th Semester

Under the guidance of ***Dr. Naween Kumar***

I. Introduction

This report details the development and implementation of Shortify, a serverless URL shortening service designed to provide users with a fast and convenient way to create and manage short links. The project leverages a suite of Amazon Web Services (AWS) to build a robust, scalable, and cost-effective backend, coupled with a user-friendly browser extension for seamless interaction. The primary goal was to create a self-hosted URL shortener under a custom domain (*thexeon.tech* in this implementation) that offers high performance for redirects through the use of a Content Delivery Network (CDN).

The traditional approach to URL shortening often involves monolithic servers, which can be expensive and complex to manage. Shortify adopts a modern serverless architecture, abstracting away infrastructure management and allowing the system to scale automatically based on demand. The integration of a browser extension enhances usability, enabling users to shorten links directly from their active browser tab without needing to visit a separate website. Performance is a key consideration, and by placing the redirect mechanism behind a global CDN, Shortify ensures that users experience minimal latency when accessing shortened links, regardless of their geographic location.

II. AWS Services Used

The Shortify project is built upon several key AWS services, each playing a specific role in the overall architecture. The serverless nature of these services contributes to the project's scalability, reliability, and cost-efficiency.

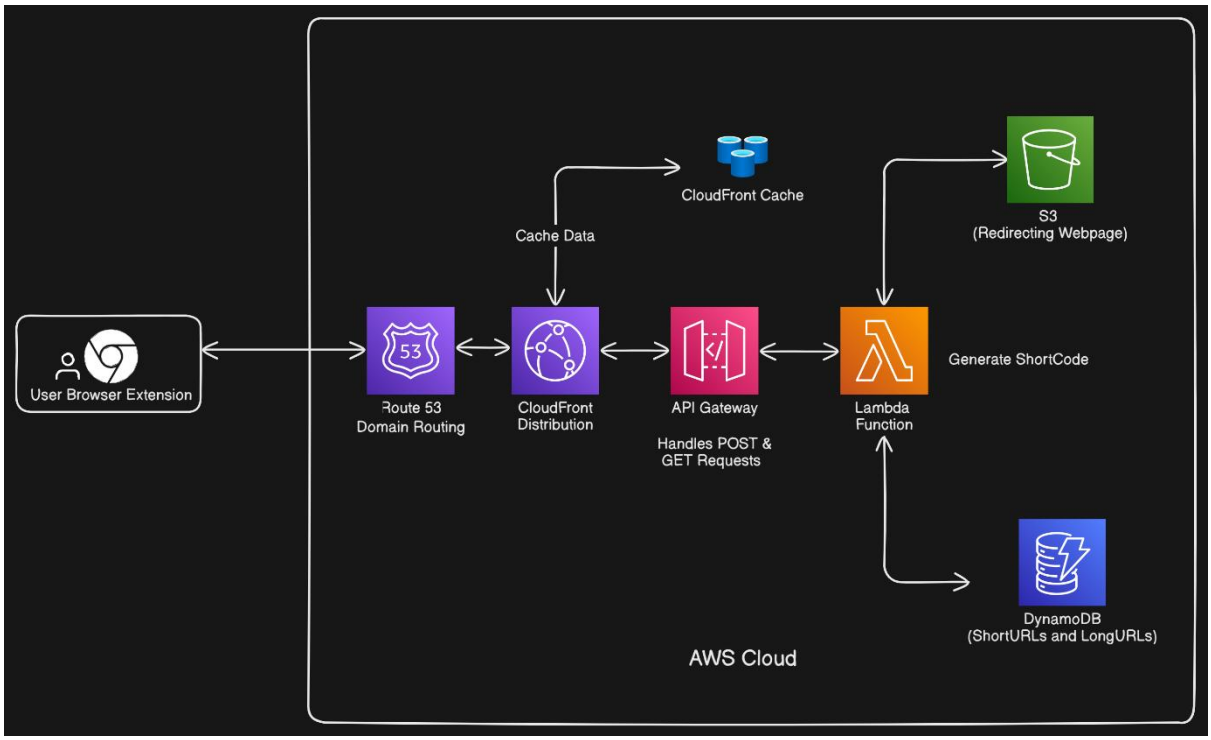

Figure 1 AWS Architecture Diagram

- **AWS Lambda:**
  - **Role:** The core compute service that executes the backend logic.
  - **Functionality:** Handles two primary tasks: generating unique short codes and storing URL mappings for incoming shortening requests (POST /shorten), and looking up the original long URL and preparing the redirect response for incoming short link clicks (GET /{short_code}).
  - **Benefit:** Provides a pay-per-execution model, automatically scales with the number of requests, and requires no server management.

- **Amazon API Gateway (HTTP API):**
  - **Role:** Acts as the front door for the backend logic running on Lambda.
  - **Functionality:** Exposes the Lambda function as a set of HTTP endpoints (POST /shorten and GET /{short_code}). It handles routing incoming HTTP requests to the correct Lambda function trigger.
  - **Benefit:** Simplifies the creation of RESTful APIs, handles request/response mapping, manages CORS, and integrates seamlessly with Lambda. HTTP API offers lower cost and complexity compared to REST API for this use case.

- **Amazon DynamoDB:**
  - **Role:** The NoSQL database used to persistently store the mapping between generated short codes and their corresponding original long URLs.
  - **Functionality:** Stores records where the short_code is the primary key and long_url is an attribute. It provides fast, low-latency data retrieval based on the short code.
  - **Benefit:** A fully managed, highly available, and scalable database service that offers consistent performance at any scale.

- **Amazon S3 (Simple Storage Service):**
  - **Role:** Hosts the static HTML file (redirect.html) used for client-side redirects.
  - **Functionality:** Serves the redirect.html template file, which is fetched by the Lambda function during a GET /{short_code} request. S3's static website hosting capabilities are used for efficient delivery of this static asset.
  - **Benefit:** Highly durable, available, and cost-effective object storage. Static website hosting provides a simple way to serve static files directly.

- **Amazon CloudFront:**
  - **Role:** Functions as a Content Delivery Network (CDN) positioned in front of the API Gateway endpoint.
  - **Functionality:** Caches the HTML redirect pages (GET /{short_code}) at edge locations worldwide, significantly reducing latency for users accessing short links. It forwards the API calls for shortening (POST /shorten) to the API Gateway origin. It also handles SSL termination and redirects HTTP traffic to HTTPS.
  - **Benefit:** Improves performance by caching content closer to users, reduces load on the origin, enhances security (DDoS protection), and provides HTTPS at the edge.

- **Amazon Route 53:**
  - **Role:** The DNS web service used to manage the custom domain (thexeon.tech).
  - **Functionality:** Configured with Alias records to point the custom domain to the CloudFront distribution. This allows users to access the shortener and short links using the project's branded domain name.
  - **Benefit:** Provides highly available and scalable DNS management, integrates tightly with other AWS services like CloudFront and API Gateway for easy alias configuration.

- **AWS Identity and Access Management (IAM):**
  - **Role:** Manages access control and permissions for AWS resources.
  - **Functionality:** An IAM Role was created for the Lambda function, granting it the necessary permissions to read from the S3 bucket (s3:GetObject) and read/write to the DynamoDB table (dynamodb:GetItem, dynamodb:PutItem). IAM ensures that the Lambda function can only interact with the specific resources it needs, following the principle of least privilege.

III. Methodology

The development of Shortify followed a phased approach, building the core backend, exposing it via an API, implementing the redirect mechanism, configuring the custom domain, optimizing with a CDN, and finally developing the browser extension.

1. **Backend Core:** Set up DynamoDB for storage and developed the Lambda function for short code generation, mapping storage, and URL retrieval.

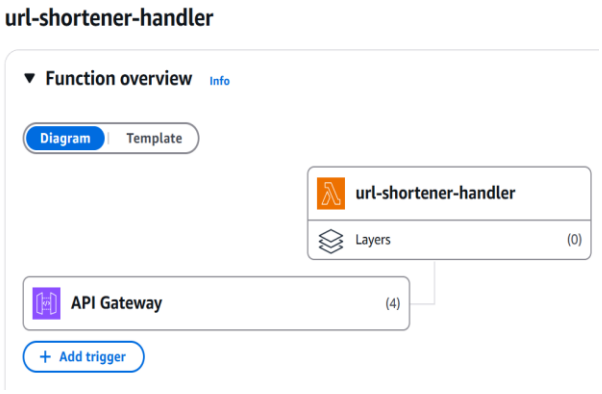

Figure 2: Lambda Fuction to handle API calls

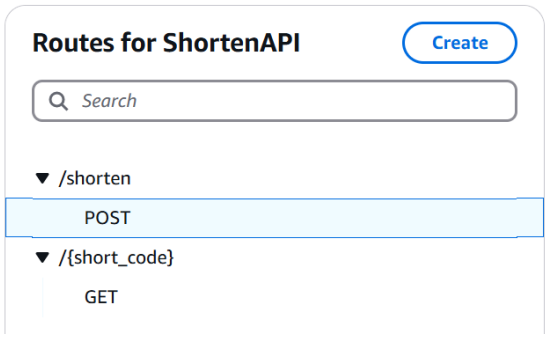

Figure 3: API Gateway to handle HTTP requests

2. **API Exposure:** Created an API Gateway HTTP API with /shorten (POST) and /{short_code} (GET) routes, integrating them with the Lambda function and configuring CORS.

3. **S3 Redirect:** Created an S3 bucket for static hosting, uploaded the redirect.html template, and modified the Lambda to fetch and populate this template for GET requests. Updated Lambda IAM role for S3 access.
4. **Custom Domain & SSL:** Requested and validated an ACM certificate for the custom domain in us-east-1. Configured the custom domain in API Gateway and initially pointed Route 53 to the API Gateway endpoint.

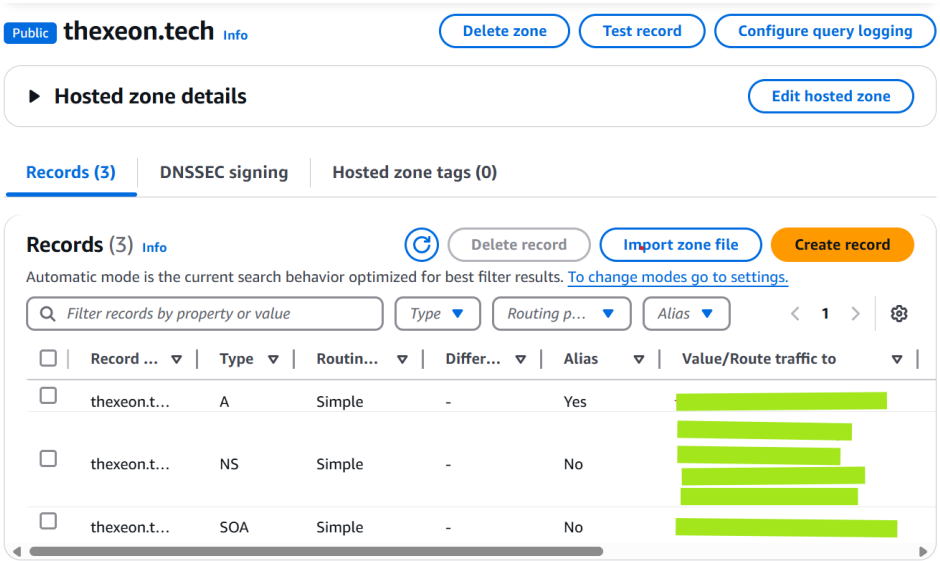

Figure 4: Route 53 records after configuration

5. **CloudFront Optimization:** Created a CloudFront distribution with the API Gateway custom domain as the origin. Configured cache behaviors and policies to disable caching for POST /shorten and enable caching for GET /{short_code} redirects. Updated the Lambda's GET response with Cache-Control headers. Updated Route 53 to point the custom domain to the CloudFront distribution.
6. **Browser Extension:** Developed the extension's HTML, CSS, and JavaScript (manifest.json, popup.html, popup.js) to get the current URL, call the CloudFront endpoint for shortening, display the result, and provide copy functionality. Tested by loading the unpacked extension.

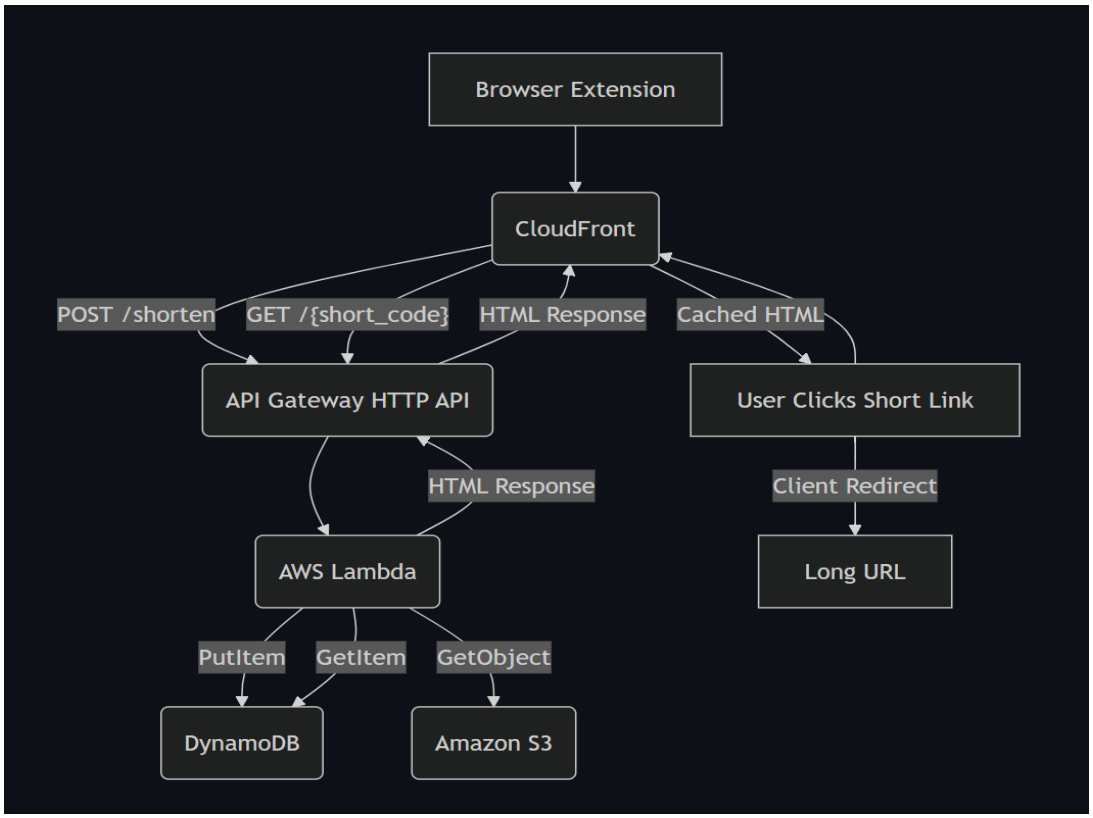

Figure 5: Flowchart of all the communications between services

IV. Result

The development of Shortify successfully yielded a functional and performant serverless URL shortening solution with a user-friendly browser extension.

The AWS backend is fully operational, capable of receiving requests via the custom domain (https://thexeon.tech). The Lambda function correctly generates unique short codes, stores them in DynamoDB, and retrieves original URLs. The integration with S3 for the redirect page template allows for a consistent and customizable redirect experience.

The CloudFront distribution effectively caches the redirect HTML pages for GET /{short_code} requests. Performance testing by observing the X-Cache header confirms that subsequent requests for the same short code are served directly from CloudFront edge locations, resulting in significantly lower latency compared to hitting the API Gateway and Lambda for every request. The /shorten endpoint

correctly bypasses the CloudFront cache, ensuring that each shortening request triggers the Lambda function to generate a new, unique code.

The browser extension provides a convenient interface for users to interact with the service. It successfully retrieves the current tab's URL, sends it to the backend via the CloudFront endpoint, and displays the generated short URL. The copy-to-clipboard functionality works as expected. The improved UI provides clear status messages and a better user experience.

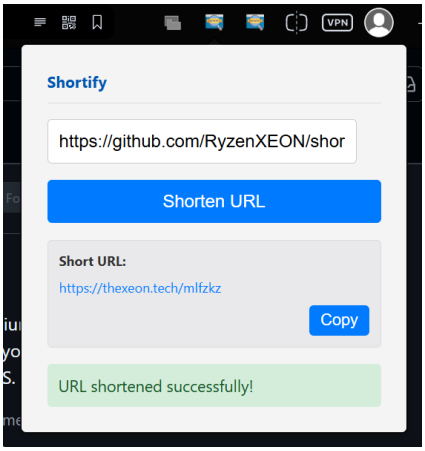

Figure 6: Extension UI

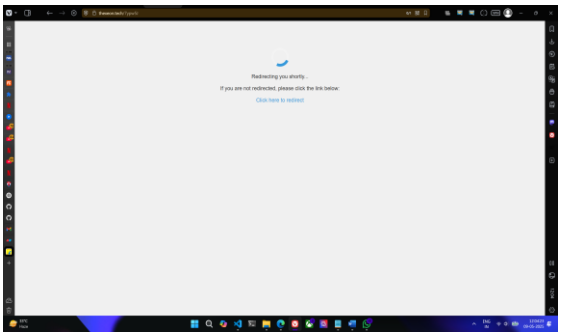

Figure 7: Redirection Page

The serverless architecture ensures that the system can handle varying levels of traffic efficiently and cost-effectively, scaling up automatically during peak usage and scaling down during idle periods. The use of a custom domain provides a branded and trustworthy appearance for the shortened links.

Overall, Shortify demonstrates a practical application of serverless technologies on AWS to build a real-world application that is both functional and performant.