URL Shortener Project Documentation

Overview

This project entails the development and deployment of a scalable and reliable URL shortener system. The architecture leverages modern cloud-native technologies to ensure high availability, low latency, and the ability to handle significant traffic with zero error rates. Below is a detailed explanation of the technologies used and the reasons behind these choices.

Architecture Components

Frontend

- **Technology**: React
- **Deployment**: AWS Amplify
- **Purpose**:
  - Provides a UI interface for shortening URLs, managing them, and viewing analytics.
  - AWS Amplify was chosen for its simplicity in managing frontend hosting and CI/CD capabilities, as well as its ability to scale automatically based on traffic.
  - URL: Staging Frontend

Backend

- **Technologies**:
  - **AWS API Gateway**: For routing and managing API endpoints.
  - **AWS Lambda**: For handling backend logic through serverless functions.
  - **DynamoDB (NoSQL)**: For storing URL data.
- **Reasoning**:
  - **API Gateway and Lambda**:
    - This serverless architecture ensures automatic scaling, reduced maintenance overhead, and cost efficiency by charging only for actual computation time only!.
    - Functions are separated for better maintainability, with each handling a distinct part of the system:
      - **Shorten URL**: Generates a shortened URL from a long one (working explaination below).
      - **Redirect URL**: Redirects users to the original URL using the shortened link (working explaination below).
      - **Analytics**: Tracks and provides data on URL access (working explaination below).
  - **DynamoDB**:
    - Chosen for its scalability, flexibility, and ability to handle high traffic efficiently.
    - NoSQL databases are well-suited for handling unstructured or semi-structured data, and they scale horizontally, making them ideal for this application, where data access patterns are primarily key-value lookups.

Database Schema

- **Table Name**: `ShortenedUrls`
- **Primary Key**: `shortUrl` (String)
- **Attributes**:
  - `longUrl` (String)

- `clickCount` (Number)
- `createdAt` (String)
- `lastAccessedAt` (String)

Functional Requirements Fulfillment

1. Shorten URL

- **Lambda Function:** `ShortenUrlFunction`
  - **Purpose**: Accepts a long URL and returns a shortened URL.
  - **Workflow**:
    i. The function receives the long URL from the frontend via API Gateway.
    ii. It generates a unique identifier (e.g., an 8-character string) (Hashing Process below).
    iii. The function constructs a shortened URL using this identifier.
    iv. It stores the mapping (`shortUrl` to `longUrl`) in **DynamoDB**.
    v. Returns the shortened URL to the frontend.
  - **DynamoDB Interaction**:
    - Inserts a new item into the `ShortenedUrls` table with attributes:
      - `shortUrl` : The unique identifier.
      - `longUrl` : The original URL.
      - `clickCount` : Initialized to 0.
      - `createdAt` : Timestamp of creation.
  - **Hash Generation Process**
    i. **Unique Combination**:
      - Each generated hash combines a timestamp (nanosecond precision) with a unique counter value, ensuring the uniqueness of each hash within the same process.
    ii. **Fast and Efficient**:
      - The use of an atomic counter and timestamp makes the hash generation extremely fast, suitable for high-speed applications that require generating **over a million hashes per second**.
    iii. **URL Safety**:
      - The Base64 URL encoder ensures that the resulting hash does not contain any characters that are unsafe for URLs (`+` , `/` , or `=` are no included).
    iv. **Deterministic Uniqueness**:
      - Within the same process, the combination of the high-resolution timestamp and incrementing counter guarantees that no two hashes will be the same.

2. Redirect URL

- **Lambda Function:** `RedirectUrlFunction`
  - **Purpose**: Redirects users to the original URL when the shortened URL is accessed.
  - **Workflow**:
    i. The function receives the `shortUrl` as a path parameter from the API Gateway.
    ii. It queries **DynamoDB** for the item with the corresponding `shortUrl` .
    iii. If found, it retrieves the `longUrl` .
    iv. The `clickCount` attribute is incremented by 1.
    v. The function returns the `longUrl` , which is then used to redirect the user.
  - **DynamoDB Interaction**:
    - Reads the item associated with the `shortUrl` .

- Updates the `clickCount` to reflect the number of redirections.

3. Analytics

- **Lambda Function:** `AnalyticsFunction`
  - **Purpose**: Provides analytics, such as the number of times a shortened URL has been accessed.
  - **Workflow**:
    i. The function receives the `shortUrl` as a path parameter from the API Gateway.
    ii. It queries **DynamoDB** for the item with the corresponding `shortUrl`.
    iii. Retrieves and returns analytics data, including `clickCount`, `createdAt`, and `longUrl`.
  - **DynamoDB Interaction**:
    - Reads the item associated with the `shortUrl` to fetch analytics data.

4. Frontend

- **Developed in React** and hosted via **AWS Amplify**, providing a user interface for interacting with the backend API.

5. Backend

- **Managed with API Gateway**:
  - Routes incoming requests to the appropriate Lambda functions.
  - Handles three primary routes:
    - **POST** `url/`: Triggers `ShortenUrlFunction` for URL shortening.
    - **GET** `url/{shortUrl}`: Triggers `RedirectUrlFunction` for redirection.
    - **GET** `url/analytics/{shortUrl}`: Triggers `AnalyticsFunction` for retrieving analytics.

Security Measures

- **URL Validation**: The system checks the format of URLs to prevent malformed.
- **Rate Limiting**: Configured at the API Gateway level to prevent abuse by limiting the number of requests from a single client.

Non-Functional Requirements

- **Scalability**: The system is designed to handle up to **60 concurrent requests per second with a zero-error rate**.
- **Reliability**: Leveraging AWS's managed services ensures high uptime and reliability.
- **Performance**: The use of AWS Lambda and DynamoDB ensures low latency for URL shortening and redirection.

Unimplemented Features

- **User Authentication**: Not yet implemented.
- **CI/CD Pipelines**: Currently, manual deployment processes are used.

Deployment Summary

- **Frontend**: Hosted on AWS Amplify for automated scaling and ease of deployment.
- **Backend**: Deployed using AWS Lambda for serverless computing, managed through API Gateway for routing requests.
- **Database**: DynamoDB used for storing URL mappings and analytics data, chosen for its scalability and ease of integration with Lambda functions.

Future Enhancements

- **User Authentication.**
- **CI/CD Pipelines.**
- **Tackle ColdStart problem for Lambda Functions.**
- **Adding User Specific Analytics.**