Yanga Mgudwa's Resume Parser: AWS Infrastructure Framework

This document details the AWS infrastructure framework for Yanga Mgudwa's resume parser application, aligning with the provided architecture diagram. The system is designed for scalability, maintainability, and security using a serverless architecture.

I. Overview:

The resume parser processes uploaded resumes (.pdf, .doc, .docx) to extract key information (name, contact details, skills, experience). The extracted data is stored in DynamoDB and visualized in a QuickSight dashboard. The system leverages serverless components (Lambda, API Gateway), AI (OpenAI), and CI/CD for automated deployments.

II. Components and Technologies:

- **Frontend:**
    - **S3:** Static website hosting (HTML, CSS, JavaScript).
    - **CloudFront:** Content Delivery Network (CDN) for global reach and improved performance.
-
- **Backend:**
    - **API Gateway:** RESTful API for handling resume uploads.
    - **Lambda Functions:** Serverless functions written in Python:
        - **upload_preprocess:** Receives resumes, validates file types, performs basic preprocessing (text extraction), and stores the preprocessed text in S3 (preprocessed-resumes bucket).
        - **resume_parsing:** Retrieves preprocessed text from S3, uses Amazon Textract and OpenAI API for detailed parsing, structures the data into JSON, and stores the structured data in S3 (parsed-resume-data bucket).
        - **data_storage:** Retrieves structured JSON data from S3 and stores it in DynamoDB.
    -
    - **Amazon Textract:** Used for initial text extraction from resume files.
    - **OpenAI API:** Used for advanced NLP to enhance data extraction accuracy.
    - **Secrets Manager:** Securely stores the OpenAI API key.
    - **S3 Buckets:**
        - preprocessed-resumes: Stores preprocessed resume text.
        - parsed-resume-data: Stores structured JSON data.
    -
-
- **Data Storage & Visualization:**
    - **DynamoDB:** NoSQL database for storing parsed resume data.
    - **AWS QuickSight:** Creates a dashboard to visualize the aggregated data from DynamoDB.

-
- **Deployment & Automation (CI/CD):**
    - **CodePipeline:** Orchestrates the CI/CD pipeline.
    - **CodeBuild:** Builds the Lambda function packages.
-

III. Data Flow:

1. **Resume Upload:** User uploads a resume via the frontend (S3 & CloudFront).
2. **API Gateway:** The upload triggers a POST request to the API Gateway.
3. **upload_preprocess Lambda:** Receives the resume, validates the file type, extracts text, and stores the preprocessed text in the preprocessed-resumes S3 bucket. It then triggers the resume_parsing Lambda function.
4. **resume_parsing Lambda:** Retrieves the preprocessed text from S3, uses Textract and OpenAI API for parsing, structures the data into JSON, and stores the structured JSON data in the parsed-resume-data S3 bucket. It then triggers the data_storage Lambda function.
5. **data_storage Lambda:** Retrieves the structured JSON data from S3 and stores it in DynamoDB.
6. **AWS QuickSight:** Connects to DynamoDB to generate and display the insights dashboard.
7. **(Optional) CI/CD:** Code changes are pushed to a Git repository, triggering CodePipeline, which uses CodeBuild to build and deploy updates to the Lambda functions and other resources.

IV. IAM Roles and Permissions:

IAM roles with least privilege are crucial for security. Roles are needed for:

- **Lambda Functions:** Permissions to access S3 (read/write), DynamoDB (write), Textract, OpenAI API, and Secrets Manager (read).
- **API Gateway:** Permissions to invoke the Lambda functions.
- **QuickSight:** Permissions to access the DynamoDB table.

V. Deployment:

1. Deploy the frontend (HTML, CSS, JavaScript) to the S3 bucket.
2. Configure the CloudFront distribution.
3. Create the IAM roles and policies.
4. Deploy the Lambda functions.
5. Configure the API Gateway endpoints.
6. Create the DynamoDB table.
7. Configure the S3 trigger.
8. Create the QuickSight dashboard.
9. (Optional) Configure the CI/CD pipeline.

VI. Testing:

Thorough testing is essential:

- Unit testing of individual Lambda functions.
- Integration testing of the entire system (e.g., using Postman).
- Functional testing of the frontend and dashboard.

This detailed framework ensures that Yanga Mgudwa's resume parser is well-architected, secure, and scalable. Remember to replace placeholder names (bucket names, Lambda function names, etc.) with the actual names used in your implementation. Regularly monitor your AWS resources for performance and cost optimization.