Serverless Portfolio Website with Automated Resume Parsing and Insights Dashboard: A Comprehensive Guide

This document provides a step-by-step guide for Yanga Mgudwa to build a serverless portfolio website with automated resume parsing and an insights dashboard on AWS. It assumes a basic understanding of AWS, Python, and web development fundamentals.

I. Project Overview:

This project will create a portfolio website hosted on AWS S3 and served via CloudFront. Uploaded resumes will be processed by an AWS Lambda function using Amazon Textract for data extraction. This data will be stored in DynamoDB and visualized in a dashboard built with AWS Quicksight. A CI/CD pipeline will automate deployments.

II. Project Components and Technologies:

- **Frontend (Website):** HTML, CSS, JavaScript, AWS S3, AWS CloudFront
- **Resume Parsing Backend:** AWS Lambda (Python), Amazon S3, Amazon Textract
- **Data Storage:** Amazon DynamoDB
- **Dashboard:** AWS Quicksight
- **Automation:** AWS CodePipeline, AWS CodeBuild (optional: CloudFormation for infrastructure)

III. Step-by-Step Execution Plan:

Phase 1: Portfolio Website Creation

1. **Website Design and Development:**
    - Create a simple static website (HTML, CSS, JavaScript). Include sections for your bio, achievements, skills, and a resume upload form.

The upload form should use a <form> element with enctype="multipart/form-data" to handle file uploads. It should POST the resume file to an API Gateway endpoint (created later). Example:

```
<form id="resumeForm" method="POST" action="/resume" enctype="multipart/form-data">
  <input type="file" name="resume" accept=".pdf">
  <button type="submit">Upload</button>
</form>
```

    - content_copy Use code with caution.Html
2.
3. **S3 Bucket Creation:**

- Create an S3 bucket to store the website's static files. Choose a globally unique name. Set appropriate permissions (e.g., public read for website files).

4.
5. **CloudFront Distribution:**
   - Create a CloudFront distribution pointing to your S3 bucket. This will serve your website via HTTPS and improve performance with a CDN.
6.

Phase 2: Resume Parsing Backend (Lambda Function & Textract)

1. **Create IAM Roles:**
   - Create an IAM role for your Lambda function with permissions to access S3 (read/write) and Textract. Use the least privilege principle.
   - Create an IAM Role for API Gateway to invoke the Lambda function.
2.
3. **Lambda Function (Python):**
   - Create an AWS Lambda function written in Python. This function will:
     - Receive the uploaded resume from API Gateway.
     - Save the resume to an S3 bucket (temporary location).
     - Use the Boto3 Textract client to extract text from the PDF.

Process the extracted text (using libraries like pandas to clean and structure the data, extract skills, etc.). This is the most complex part and requires custom logic. Example (simplified):

```
    import boto3
import json

s3 = boto3.client('s3')
textract = boto3.client('textract')
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('ResumeData') # Create DynamoDB table beforehand

def lambda_handler(event, context):
   resume_key = event['resumeKey'] # From API Gateway
   bucket_name = 'your-s3-bucket-name'

   try:
      response = textract.detect_document_text(
         Document={'S3Object':{'Bucket': bucket_name, 'Name': resume_key}}
      )

      text = ' '.join([block['Text'] for block in response['Blocks'] if 'Text' in block])
      # ...Your custom text processing logic here to extract skills, experience, etc...
      parsed_data = {
         'resume_key': resume_key,
         'skills': ['skill1', 'skill2'], # extracted skills
```

```
        'experience': '...', # extracted experience
        }

    table.put_item(Item=parsed_data)
    return {'statusCode': 200, 'body': json.dumps('Success!')}
  except Exception as e:
    return {'statusCode': 500, 'body': json.dumps(str(e))}
```

- content_copy Use code with caution.Python

4.
5. **API Gateway:**
   - Create an API Gateway REST API with a POST method that invokes your Lambda function. Configure the API Gateway to accept multipart/form-data.
6.
7. **S3 Trigger:**
   - Configure an S3 event notification (using the AWS console or CLI) to trigger the Lambda function when a new resume is uploaded to the designated S3 bucket.
8.

Phase 3: Data Storage and Dashboard

1. **DynamoDB Table Creation:**
   - Create a DynamoDB table to store the parsed resume data. Define appropriate schema (e.g., primary key: resume_key, attributes: skills, experience, job_title, etc.).
2.
3. **AWS Quicksight Dashboard:**
   - Create a Quicksight data source connecting to your DynamoDB table.
   - Build a dashboard visualizing the aggregated data (e.g., frequency of skills, experience levels).
4.

Phase 4: CI/CD Pipeline (Optional)

1. **CodePipeline/CodeBuild Setup:** (if using these services)
   - Configure a CodePipeline pipeline to automatically build and deploy your website and Lambda function code upon code changes in your Git repository (e.g., GitHub). CodeBuild will handle building the Lambda function package.
2.

IV. Deployment and Testing:

1. Deploy your website files to S3.

2. Deploy your Lambda function.
3. Test the resume upload functionality and verify that data is correctly parsed and stored in DynamoDB.
4. Create and test your Quicksight dashboard.

V. Additional Considerations:

- **Error Handling:** Implement robust error handling in your Lambda function to catch and log exceptions. Send error notifications (e.g., via SNS).
- **Security:** Carefully configure IAM roles and permissions to minimize security risks. Use secrets manager for sensitive information.
- **Scalability:** Design your application to handle a large number of resume uploads. Consider using DynamoDB's auto-scaling features.
- **Documentation:** Write a detailed README file explaining the architecture, deployment process, and how to use the application.

This comprehensive guide provides a detailed roadmap for creating your serverless portfolio website. Remember to replace placeholder names (bucket names, table names, etc.) with your own values and adapt the code to your specific requirements. Thorough testing and iterative development are crucial for success.