**EMPLOYEE DETAILS QUERY APPLICATION**

*Designed as AWS Server-less website*

URL: <http://reshma-cloud490.s3-website.us-east-2.amazonaws.com/>

*Designed and Developed by Reshma Maduri Sivakumar*

**OVERVIEW**

Employee Details Query Application is designed as a Server-less web application which is hosted as a static website on Amazon AWS S3 Bucket. There is no dedicated server for this application.

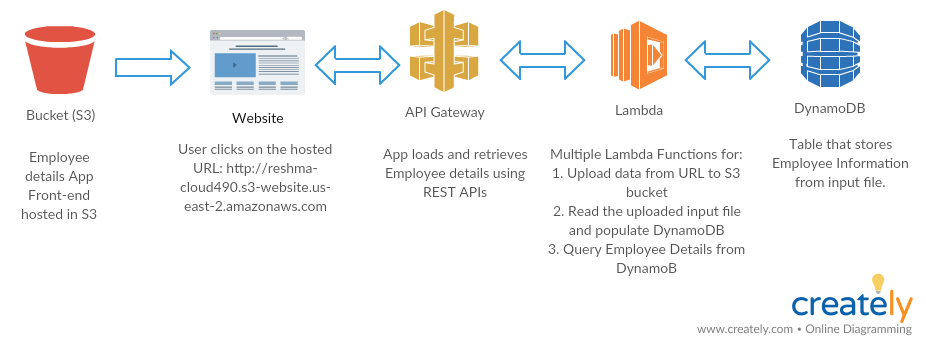
The frontend is developed as a responsive website using HTML5, CSS3, Bootstrap and JQuery/JS. The frontend is hosted on Amazon AWS S3 bucket along with the REST APIs that were designed in AWS API Gateway.

The backend components are developed using AWS services including Lambda Functions written in Python 3.6, DynamoDB table and GSI index, and API Gateway to create REST APIs. The Lambda functions are decoupled micro-services that serves as business logic tier.

The benefits of this design are:

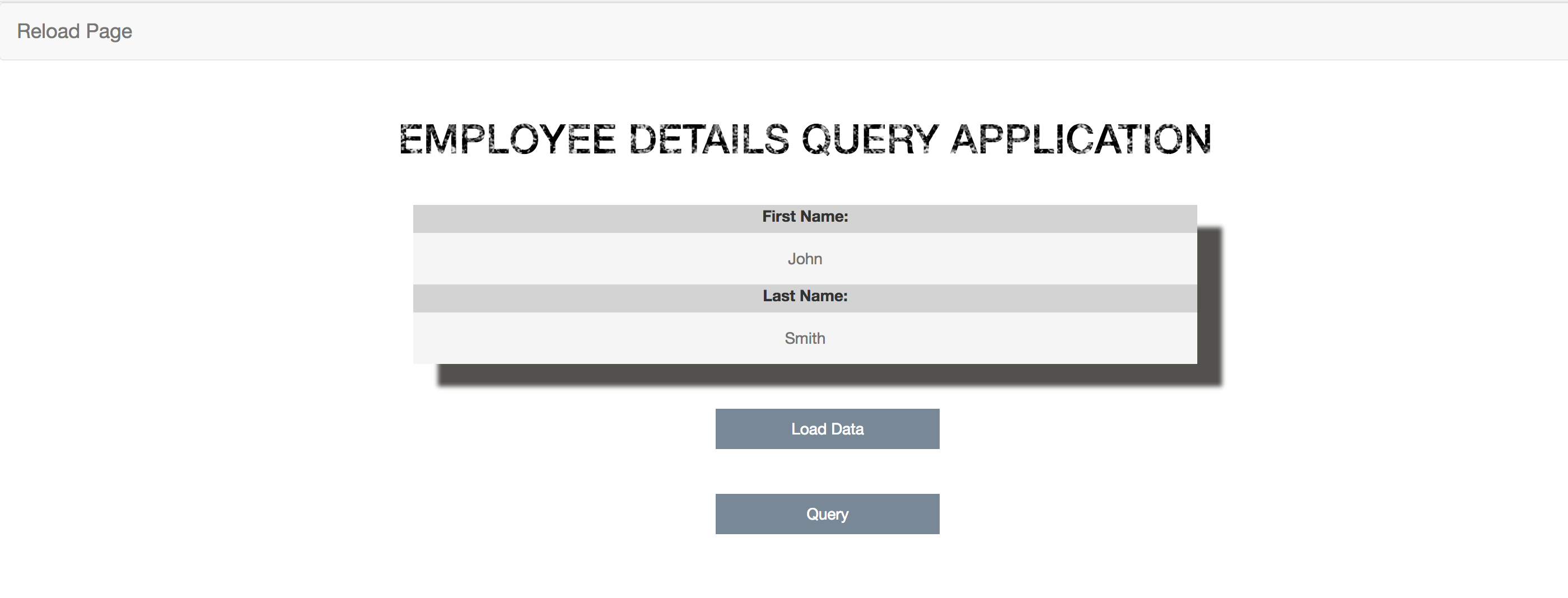
1. **High Availability:** As this application is hosted and managed in AWS, uptime will be 99%/month.
2. **Scalability:** As this application is Server-less application, AWS will serve the website with low latency and high bandwidth.
3. **Cost Efficiency:** As this is Server-less application, it follows pay per use model. Hence, it’s cheaper.

**APPLICATION FLOW**

****

*Figure 1: Server-less application flow of Employee Details Query App.*

1. The user will connect to the application through the URL: <http://reshma-cloud490.s3-website.us-east-2.amazonaws.com> which is hosted in AWS S3.
2. The website is launched upon clicking on the URL (See Figure 2).



*Figure 2: Server-less application hosted website UI.*

1. Click on “Load Data” button to load the input file from the URL: <https://s3-us-west-2.amazonaws.com/css490/input.txt> to AWS S3 bucket. The “Load Data” button calls a Lambda function through REST API, which in turn calls the Lambda function to upload the input file from this URL into S3.
   1. Data Format: The format will be a text file with one line per item. Each line will have a last name, a first name and then a variable number of attributes:

***lastName firstName [attribute1=value1 attibrute2=value2 attribute3=value3 ...]***

*Example Data:*

*Dimpsey Robert id=65764 phone=4528769876 office=trulyhouse*

*Vollmann William id=87643 phone=8769870987 age=57*

*Trevor Billy id=7638 age=81 gender=M*

*Munro Alice id=9876 age=65 gender=F*

*Dimpsey Pearl id=98776 phone=4528769876 office=none*

1. When the file gets uploaded in S3 bucket, it triggers an upload event which calls a Lambda function to read the uploaded file, parses the contents into key value pairs and populates the employee details into the DynamoDB table.
2. Click on “Query” button in figure 2 to query employee details from DynamoDB table using first name and/or last name criteria. The “Query” button calls the REST API which in turn calls the Lambda function to retrieve the data.

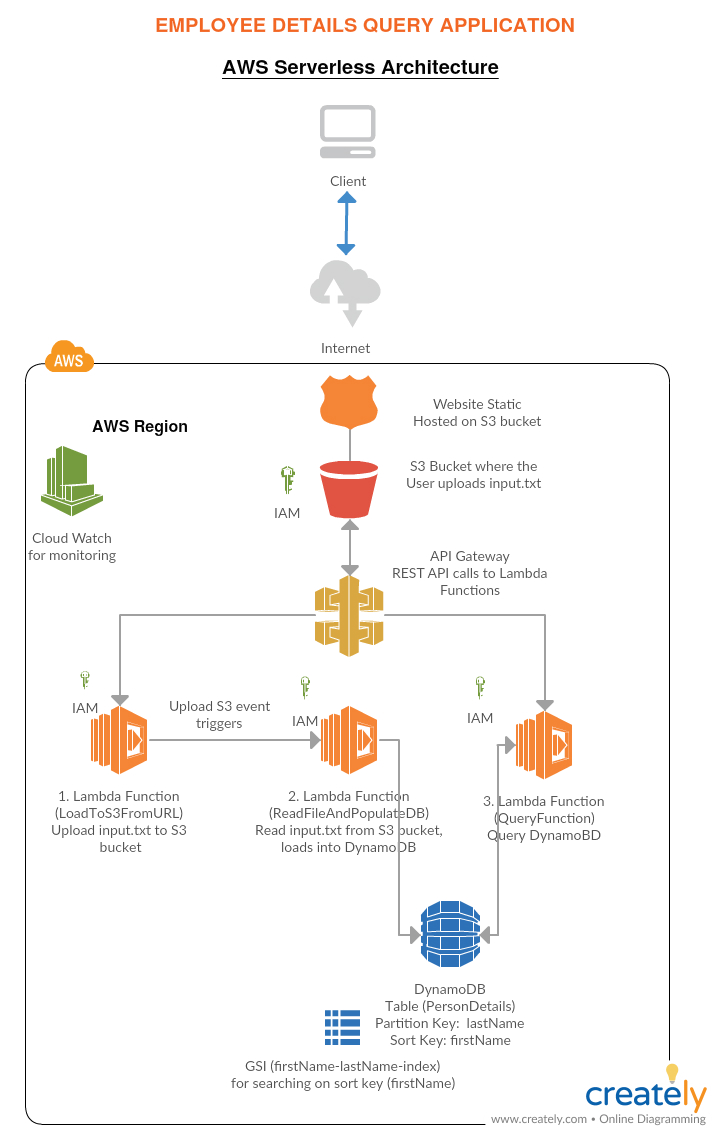
***Note:*** *Query: The last name / first name combination are guaranteed to be unique. A query of*

*“Vollmann” and “William” should return the attributes: id=87643 phone=8769870987 age=57.*

*A query of “Dimpsey” “” should return both:*

*Dimpsey Robert id=65764 phone=4528769876 office=trulyhouse*

**DETAILED DESIGN**

****

**Authentication**

The authentication used in this application through AWS Identity and Access Management (IAM).

The IAM roles created for this application are:

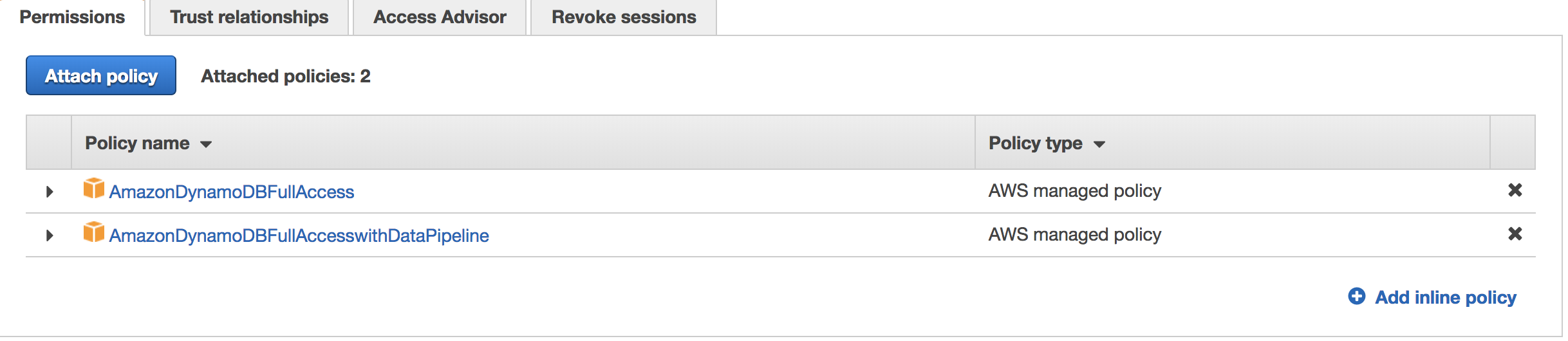
1. lambda\_basic\_execution

Policy:



1. lambda\_dynamo\_execution

Policy:



**AWS S3 Bucket**

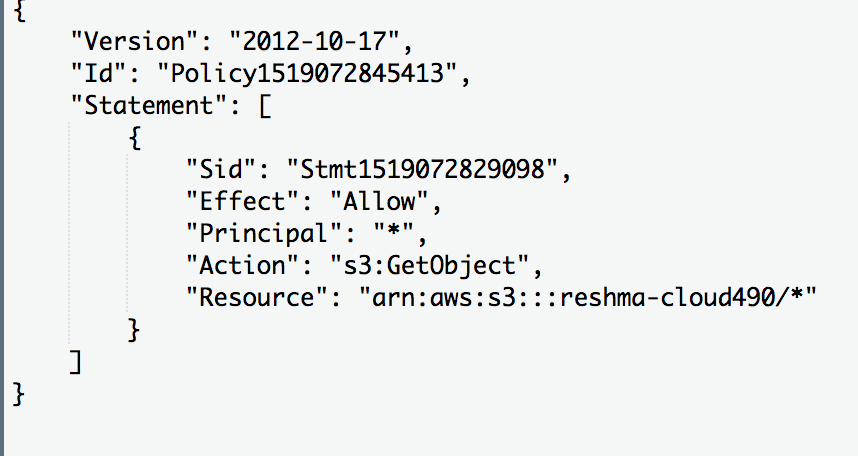
The web app is hosted as a static web application in a publicly accessible S3 bucket.

Bucket name: [reshma-cloud490](https://s3.console.aws.amazon.com/s3/)

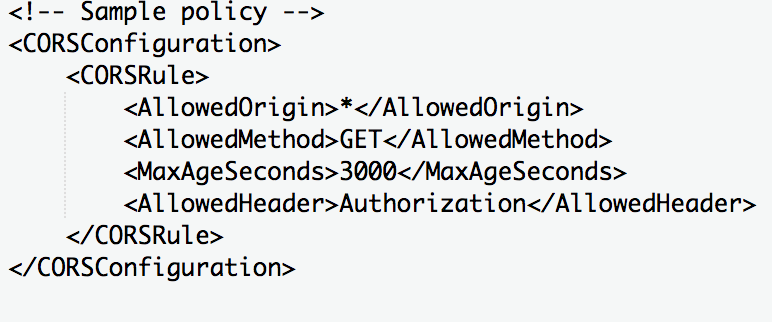
ARN: arn:aws:s3:::reshma-cloud490

Region: us-east-2

Bucket Policy:



CORS Configuration:



The CORS configuration is used to allow communication between S3 from the frontend.

**AWS Dynamo DB**

The employee details from the input file is loaded into Dynamo DB, which is a robust, cheap and highly scalable server-less database.

The following objects are created in the Dynamo DB:

1. **Table:** PersonDetails

**Partition key:** lastName

**Sort key:** firstName

1. **Global Secondary Index:** firstName-lastName-index

**Partition key:** firstName

**Sort key:** lastName

***Note****: GSI is created for any columns other than the main primary key. GSI can be on columns that don’t exist on every row*

DynamoDB pricing model requires fixed cost to provision per hour capacity i.e., it is not pay per use model but was used here for high scalability and robustness.

**AWS Lambda**

The server-side business logic for this application is written in Lambda using Python 3.6. The following are the Lambda functions for this application:

1. **LoadToS3FromURL**

**Invoked by:** API REST call from “Load Data” button from UI.

**API:** GetDetails

**IAM Role:** lambda\_basic\_execution

**Input Params:** None

**Description:** The function reads the URL: <https://s3-us-west-2.amazonaws.com/css490/input.txt> and uploads to AWS S3 bucket “reshma-cloud490”.

1. **ReadFileAndPopulateDB**

**Invoked by:** Trigger attached for S3 bucket “reshma-cloud490”

**IAM Role:** lambda\_dynamo\_execution

**Input Params:** None

**Description:** This Lambda function is triggered as soon as the input file is uploaded into S3 bucket “reshma-cloud490”. The function reads the content of the file and parses the contents into key value pairs and populates into DynamoDB table PersonDetails.

1. **QueryDetails**

**Invoked by:** API REST call from “Query” button from UI.

**API:** QueryApi

**IAM Role:** lambda\_dynamo\_execution

**Input Params:** firstName, lastName

**Description:** The function queries the DynamoDB table PersonDetails with the firstName and lastName input parameters. If firstName is not available, the function queries on the last name (partition key). If lastName is not available, the function uses GSI index to query based on firstName (sort key). If no data is passed or if lastName/firstName does not exist, it returns “No records found for search!”. The results fetched is displayed on the UI.

**AWS API Gateway**

The frontend of this application connects to the backend business logic (Lambda functions) using API Gateway service. API Gateway is used to build RESTful APIs that receives all incoming requests from the frontend, check authorization and routes them to the Lambda function. As these are managed by AWS, it guarantees high availability and scalability.

The two RESTful APIs created for this application are:

1. **GetDetails:** invoked by the “Load Data” button

**Query strings:** None.

**Stage URL:** <https://ly0c67sh49.execute-api.us-east-2.amazonaws.com/Test/GetDetails>

1. **QueryApi:** invoked by the “Query” button

**Query strings:** firstName, lastName

**Stage URL:** <https://3pcwq2m7i8.execute-api.us-east-2.amazonaws.com/Test/persondetails>

**SDK Generation**

The SDKs are developed for Javascript and downloaded as apiGateway-js-sdk package which contains the apigClient.js file and the lib directory. The apigClient.js contains both the above mentioned REST calls.

**Static Web Hosting in AWS S3**

The UI for the web app along with the apigClient.js and the lib directory are uploaded into the S3 bucket and hosted using the S3’s static web hosting service.

**Monitoring using AWS CloudWatch**

The DynamoDB tables, lambda functions and applications are monitored using CloudWatch.