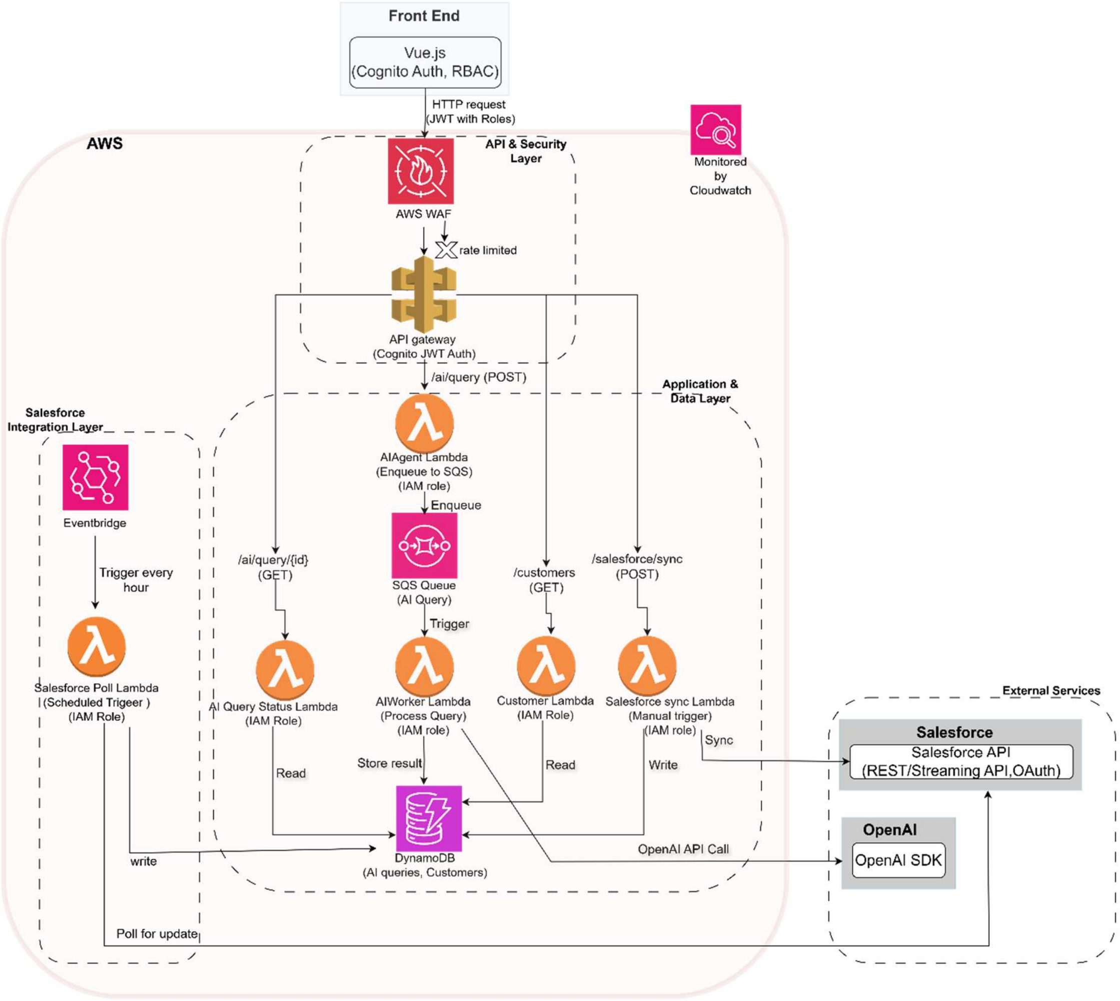
Author: Pratik Vyas

Index

System Architecture Overview



This document outlines the serverless, event-driven architecture for an AI-powered SaaS application. The system is designed on Amazon Web Services (AWS) to be scalable, fault-tolerant, and secure. It comprises a Vue.js **Front End** for user interaction, a suite of serverless **backend services**, and integrations with **external platforms** including **OpenAI** for AI processing and **Salesforce** for customer data management.

The architecture is logically divided into four primary areas:

1. **Front End:** The client-side application that serves as the user interface.
2. **API & Security Layer:** The public-facing entry point responsible for security, authentication, and request routing.
3. **Application & Data Layer:** The core business logic, including asynchronous processing and data persistence.
4. **Salesforce Integration Layer:** A dedicated set of components for synchronizing data with Salesforce.

All services within the AWS cloud are monitored for performance and health by **Amazon CloudWatch**, which collects logs, metrics, and events.

Components

- **Front End (Vue.js):** A Single-Page Application (SPA) providing the user interface. It uses **Cognito Auth**, meaning it integrates with the AWS Cognito SDK to manage user sign-in and securely handle JSON Web Tokens (JWTs).
- **AWS WAF (Web Application Firewall):** The first line of defense at the network edge. It inspects all incoming HTTP/S traffic, filtering out malicious requests like SQL injections, cross-site scripting (XSS), and traffic from known bad IPs based on configurable security rules.
- **Rate Limiting:** A crucial function performed at the edge (by WAF and/or API Gateway) to prevent abuse and ensure service availability. It throttles requests based on rate (e.g., requests per second) or quotas (e.g., requests per day).
- **Role-Based Access Control (RBAC)** is a security method that restricts system access based on a user's assigned role, such as 'admin' or 'viewer'. Instead of assigning permissions to each individual user, you assign users to roles, and each role has its own predefined set of permissions. This simplifies managing permissions and ensures users only have the access they need for their job.
- **AWS API Gateway:** A fully managed service that acts as the application's front door. Its roles here are:
  - **Routing:** Directing incoming requests to the correct backend Lambda function based on the path and HTTP method.
  - **Authorization:** Using a **Cognito JWT Authorizer** to validate the JWT sent with each request, ensuring only authenticated and authorized users can access the API.
  - **Throttling:** Enforcing usage plans and rate limits.
- **AWS Lambda:** Serverless, event-driven compute functions. Each Lambda is a specialized microservice with a single responsibility.
- **AWS SQS (Simple Queue Service):** A managed message queue used to **decouple** the initial API request from the long-running AI processing. This is fundamental to the asynchronous pattern.
- **AWS EventBridge:** A serverless scheduler to trigger periodic tasks.
- **AWS DynamoDB:** A serverless NoSQL database providing low-latency storage for application data, including AI query history and customer information.
- **AWS CloudWatch:** The central monitoring and observability service. It collects logs, metrics, and events from all AWS services, enabling debugging, performance monitoring, and alerting.
- **External Services:**
  - **OpenAI:** An external AI platform, accessed via its **SDK** for generative AI tasks.
  - **Salesforce:** A CRM platform, integrated via its **API** for data synchronization.

Layers

1.0 Front End Layer

The Front End is built using the **Vue.js** framework. It is responsible for rendering the user interface and managing all client-side interactions with the backend API.

Key Responsibilities & Technical Implementation:

- The application integrates with **AWS Cognito** using the AWS Amplify SDK for user identity management. This handles user sign-up, sign-in, and session management.
- Upon a successful login, Cognito issues a set **of JSON Web Tokens (JWTs)** to the client.
- **Role-Based Access Control (RBAC)** is implemented via Cognito User Groups. Users are assigned to groups (e.g., 'administrators', 'standard-users'), and this group membership is included as a 'claim' within the JWT (e.g., cognito:groups": ["administrators"]).
- The Front End can **inspect these roles** client-side to conditionally render UI components, such as showing an "Admin Dashboard" button only to users with the appropriate role.
- All outgoing API calls from the SPA are sent as **HTTP requests**.
- For every request to a protected endpoint, the client attaches the JWT to the Authorization header. This token carries the user's identity and their roles, enabling fine-grained access control on the backend.
- Client-side routing is managed by Vue Router.
- **Application state** (e.g., user authentication status, customer data, active query statuses) is centrally managed using a state management library such as Pinia or Vuex for predictable and efficient data flow.

2.0 API & Security Layer

This layer serves as the secure perimeter for the application, processing all incoming requests from the Front End.

- **AWS WAF** (Web Application Firewall) is the first point of contact for all inbound traffic. It provides protection against common web exploits like SQL injection and cross-site scripting. It is also configured with rate-based rules to mitigate DDoS attacks. Malicious requests are blocked at this stage with an HTTP 403 Forbidden response.

- After passing WAF inspection, the request is subject to **rate-based rules**. This is implemented either within WAF or using API Gateway Usage Plans. It mitigates DDoS attacks and prevents system abuse by throttling traffic from a single source that exceeds a configured threshold (e.g., 1000 requests per minute).

- **Amazon API Gateway**, a fully managed service that acts as the application's API front door. Its responsibilities include:

  - Legitimate, rate-compliant requests are forwarded to API Gateway.
  - The gateway is configured with a Cognito JWT Authorizer. This authorizer automatically performs the following validation steps on the incoming JWT:
    - ✓ Verifies the token's signature against the Cognito User Pool's public key.
    - ✓ Checks the token's expiration (exp claim).
    - ✓ Validates the audience (aud) and issuer (iss) claims.
  - If the token is valid, API Gateway passes the request and a context object (containing the token's claims, including user roles) to the integrated backend service. If invalid, it returns a 401 Unauthorized error.

    - Finally, based on the HTTP method and resource path (e.g., POST /ai/query), it routes the request to the correct Lambda function.

- **Inbound Request Flow:**

  Client Request -> AWS WAF -> Rate Limiting Check -> API Gateway (JWT Authorization & Routing) -> Backend Lambda

3.0 Application & Data Layer

This is the core of the backend, containing the application's business logic and data persistence mechanisms. Each Lambda function operates under a specific **IAM Role**, granting it only the permissions necessary to perform its function (Principle of Least Privilege).

3.1 Asynchronous AI Query Submission & Processing

- The authorized request is routed to the **AIAgent Lambda**. This function acts as a lightweight facade.
- It generates a unique correlation ID, packages the request payload into a message, and Enqueues it to the **SQS Queue (AI Query)**.
- It immediately returns an HTTP 202 Accepted response with the correlation ID, decoupling the long-running task from the user request.
- The SQS queue Triggers the **AIWorker Lambda** asynchronously. SQS provides durability, ensuring the message will be processed even if the worker fails initially.
- The **AIWorker Lambda** (with IAM Role Process Query) retrieves customer data from DynamoDB if needed, makes an **OpenAI API** Call, and upon completion, Stores the result in the **DynamoDB** table

3.2 AI Query Status Retrieval (GET /ai/query/{id})

- The client polls this endpoint with correlation ID to check for results.
- The request is routed to the **AI Query Status Lambda**, which performs a Read operation on **DynamoDB** using the correlation ID and returns the current status and result.

3.3 Customer Data Retrieval (GET /customers)

- The client sends a **GET reque**st.
- The **Customer Lambda** performs a Read on the Customers table in **DynamoDB** and returns the data in an HTTP 200 OK response.

4.0 Salesforce Integration Layer

This dedicated layer handles all data synchronization with the Salesforce platform, ensuring a clean separation from the main application logic.

4.1 Scheduled Batch Synchronization (Polling)

- Amazon **EventBridge** is configured with a rule to Trigger every hour.
- This event invokes the **Salesforce Poll Lambda** (with IAM Role Scheduled Trigger).

- The Lambda initiates an OAuth authentication flow with the **Salesforce** API, then performs a Poll for update by making a REST API call to query for records modified since the last run.
- It then writes the retrieved updates into the **DynamoDB** Customers table.

4.2 Manual Synchronization (POST /salesforce/sync)

- An authorized user (likely an administrator, whose role is checked by the authorizer) can trigger this endpoint.
- The request is routed to the **Salesforce sync Lambda** (with IAM Role Manual trigger).
- This Lambda performs a **Sync operation** (e.g., pushing data from DynamoDB to Salesforce) and may Write a log of its activity to **DynamoDB**.

5.0 External Services

These are third-party platforms that the AWS backend integrates with.

- **Salesforce:** The system of record for customer data. The architecture is designed to interact with its **REST/Streaming APIs**. All communication is secured via **OAuth**-based authentication.
- **OpenAI:** The provider of generative AI models. The **AIWorker Lambda** interacts with this service via the official **OpenAI SDK**, which abstracts the underlying HTTP API calls.

Source Code

https://github.com/Git-PratikVyas/AWS/tree/main/ExpediteCommerce