Development Strategy and Resource Plan

Amazon Connect Supervisor Insights

Version	1.0.0
Revision Status	Submitted
Last Update	14.03.2024

—

Safe Corp Jiva

1. Development Strategy

1.1. Architecture Design

We present a pilot design for the Amazon Connect Supervisor Insights application (see **Figure 1.1.1**). It is worth mentioning that the Amazon Connect business logic; for instance, enabled services, functions or external services used by the Contact Center like CRM systems, is not considered on the architecture as it should be independent from the Web App. Moreover, as we are using a Full Stack framework (see **Next JS** in **Section 2.1**) and the serverless-first **AWS Amplify** service, the server side functionality will be translated into lambdas by the build process used in **Amplify Hosting**. Therefore we have generalized the functions to show only the relevant relationships between services.

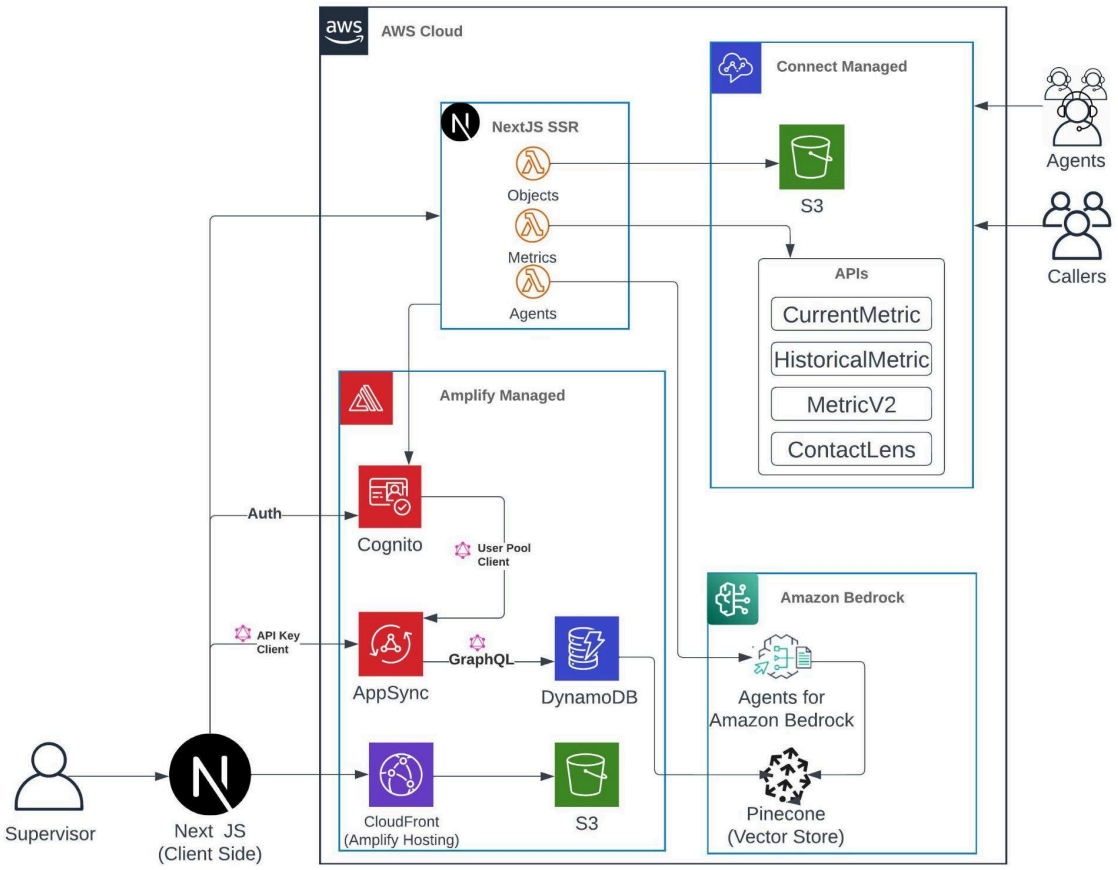

Figure 1.1.1. Statement of Architecture Work

1.2. Service Description

AWS Amplify

A unified development platform for mobile and web applications that allows developers to focus on their core business logic while AWS Amplify handles common tasks like authentication, storage, and data sources. [7]

AWS Amplify Hosting

A service for hosting web applications. It can be configured with CI/CD pipelines for both static and server-side rendered apps and built in support for common frontend frameworks. [8]

Amazon Cognito

A user identity and access management service that controls access to AWS resources, mobile apps and APIs. It allows creation of user pools for authentication and authorization. It also allows the creation of groups and user attributes. [4]

Amazon Connect

A cloud-based contact center platform that makes it easy to provide reliable customer engagement and allows administrators to interact with other AWS services using a low code platform or APIs. [3]

AWS AppSync

A managed service that enables real-time data synchronization and offline programming for mobile, web, and enterprise applications. It is more commonly used with GraphQL, although other data sources can be configured. [1]

Amazon DynamoDB

A fully managed NoSQL database service. It is a key-value and document database with a great focus on performance and scalability. [5]

Amazon S3

An object storage service offering scalability, data availability, security, and performance. It is commonly used for storing static websites, mobile application assets (such as images or documents) and data for analytics. [6]

AWS Lambda

Serverless functions. AWS lambda functions execute code in response to events and automatically manages the computing resources required by that code. [9]

Amazon Bedrock

Amazon's generative AI solution. It provides APIs and tools to build, deploy and manage conversational agents, chatbots and *foundational models*. [2]

Pinecone

Pinecone is a vector embeddings database (vector store) that is used as a knowledge base for RAG applications, as it allows machine learning models (most commonly LLMs) to query information by *closeness*, which represents semantic similarities according to the embeddings. [11]

2. Resource Plan

2.1 Technological Resources

Node JS v.20 (LTS) - JavaScript Runtime

We will use the popular open-source environment that lets you run JavaScript code outside of a web browser for both front-end and back-end development using a Full Stack framework (see **Next JS** in the next subsection). It uses the V8 JavaScript engine and is distinguished by its use of an event-driven architecture, which allows developers to manage multiple requests using *Promises*, the *async* - *await* syntax or callbacks. [13]

Next JS 14 (App Router) - Full Stack Framework

We will use Next.js, an open-source framework built on top of React that simplifies building web applications. It extends React by offering features like server-side rendering and static site generation, which can improve performance and SEO. It includes built-in routing and data fetching mechanisms, streamlining development. [14]

DynamoDB - NoSQL Database

As mentioned above we will use DynamoDB, a serverless NoSQL database service offered by AWS. Unlike relational databases, DynamoDB stores data as key-value pairs or documents, making it flexible for various data models and due to the serverless nature of the service DynamoDB takes care of managing the infrastructure, backups, and updates, while being cost effective (you only pay for what you use). [5]

GraphQL (AppSync) - Query Language and API Architecture

GraphQL is a query language for APIs that gives clients control over exactly what data they fetch. Unlike traditional REST APIs where you hit specific endpoints, GraphQL lets you request specific fields from different parts of your data in a single request. This reduces over-fetching and under-fetching, improving efficiency. It also offers strong typing and introspection, making it easier for developers to understand and use the API. We will use AppSync to manage the GraphQL API, which also facilitates the generation of documentation and standard queries using Amazon's code generation (*codegen*) CLI tool. [12]

Amazon Connect - Contact Center Platform

Amazon Connect is a cloud-based contact center solution from Amazon Web Services (AWS). It allows businesses to set up a virtual call center with features like voice calls, chat, and integrations with AI-powered chatbots. Although the Contact Center is not part of our development, it will be a core service for our application, due to the integrations the system must have. [3]

Git and GitHub - Version Control System and Hosting

Git is a version control system that allows you to track changes made to code over time. GitHub, on the other hand, is a web-based platform that stores these Git repositories. It offers features like collaboration tools, code review, and social coding, making it ideal for programmers working together on projects. We will use both these services to manage code versions, facilitate collaboration, and track progress. [15]

2.2. Information Resources

API Documentation

The rest of our documentation is hosted on the Github Pages from our repository, however, this is an example of an endpoint that the GraphQL system would have:

Service Name	listCalls(filter, limit, nextToken)
Description	A function that queries “call” entities from the DynamoDB instance using GraphQL with resolvers managed by AppSync and Amplify.
HTTP Method	POST
URL	https://api-id.appsync-api.region.amazonaws.com/graphql
Headers	**Authentication** “x-api-token”: “API Key”
Input (String)	JavaScript `query SampleQuery { listCalls(filter, limit, nextToken) { items { id agentId caller { id sentiments } metrics { id length } } } }`
Input Description	A GraphQL query that can contain an optional filter object allowing API users to use boolean operations that the result should match, a limit (similar to the SQL statement) which reduces the amount of results returned to a maximum amount and a nextToken that can be used an as offset when using a limit.

<table>
<tr><td>Response (JSON)</td><td>

```
JavaScript
{
  "data": {
    "listCalls": {
      "items": [
        {
          "id": "uuid"
          "agentId: "uuid"
          "caller": {
            "id": "uuid"
            "sentiments": [
              "neutral",
              ...
            ]
          }
          "metrics": {
            "id": "uuid"
            "length": 0
          }
        },
        ...
      ]
    }
  }
}
```

</td></tr>
<tr><td>Response Description</td><td>GraphQL responses are structured in JSON and these match both the schema and the format of the request string. For this particular query we will receive an object with a “data” key, which contains the name of the query and follows the query format: a (possibly empty) list of results which are objects containing an id, an agentId, a caller object and a metrics object. The caller object has an id and a list of sentiments, while the metrics object has an id and a call length (for this particular example).</td></tr>
<tr><td>Status Codes</td><td>We will follow the HTTP response status code standard [10]
- Informational Responses (100 - 199)
- Successful Responses (200 - 299)
- Redirection (300 - 399)
- Client Error (400 - 499)
- Server Error (500 - 599)

The most relevant ones for this endpoint are:
- 200 (Ok)
- 401 (Unauthorized)
- 403 (Forbidden)
- 404 (Not Found)
- 500 (Internal Server Error)

Status codes are located in the response metadata and these do not form part of the response body itself.</td></tr>
</table>

References

[1] AWS. (n.d). *Amazon AppSync*. https://aws.amazon.com/appsync/

[2] AWS. (n.d). *Amazon Bedrock*. https://aws.amazon.com/bedrock/

[3] AWS. (n.d). *Amazon Connect*. https://aws.amazon.com/connect/

[4] AWS. (n.d). *Amazon Cognito*. https://aws.amazon.com/cognito/

[5] AWS. (n.d.). *Amazon DynamoDB*. https://aws.amazon.com/dynamodb/

[6] AWS. (n.d). *Amazon S3*. https://aws.amazon.com/s3/

[7] AWS. (n.d.) *AWS Amplify*. https://aws.amazon.com/amplify/

[8] AWS. (n.d.). *AWS Amplify Hosting*. https://aws.amazon.com/amplify/hosting/

[9] AWS. (n.d). *AWS Lambda.* https://aws.amazon.com/lambda/

[10] MDN et al. [MDN Contributors] (November 2023).*HTTP response status codes*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

[11] Pinecone. (n.d.). *Build knowledgeable AI*. https://www.pinecone.io/

[12] The GraphQL Foundation. (n.d). *A query language for your API*. https://graphql.org/

[13] The OpenJS Foundation & the *Node.js* contributors. (n.d). *About Node.js®*. https://nodejs.org/en/about

[14] Vercel. (n.d.) *App Router*. https://nextjs.org/docs/app

[15] GitHub Inc. (2014). *Git Guide*. https://github.com/git-guides