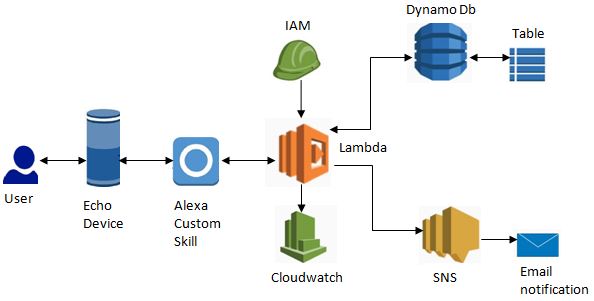
**Problem Statement:**

Ability to query the AWS Support Cases through voice and get notified in case of a critical case being opened.

**Services Used:**

1. Alexa custom skill
2. Lambda
3. DynamoDb
4. IAM
5. Cloudwatch
6. SNS

**Architecture Diagram:**

****

**Implementation:**

To illustrate the main idea behind an interaction with Alexa, let’s look at the launch of a skill called “Case Organiser”. When the user says: “Open case organiser ,” the “Case Organiser” bit is the skill’s Invocation name, which Alexa uses to communicate. The user’s device passes what the user said to the “Case Organiser” custom skill, this is called skill invocation. At this point, Alexa uses the skill’s Interaction model to understand what the user requested. The interaction model is a JSON file that maps what the user says to a request type. In this case, it will map it to the built-in AMAZON.LaunchRequest.

Next, it calls its backend written in Nodejs, an AWS Lambda function, which receives the identified request. The Lambda searches for a function that can handle the LaunchRequest and executes it. This function then returns a response that gets sent back all the way up to the user’s device. Similarly multiple handlers are created in lambda for handling different user requests. At this point, the “Case Organiser” skill will answer to their queries by searching them in DynamoDB table which is having all the AWS support cases data like case id, language, status, communications, time, subject and submitted by etc. The user can query along this data for say “tell me about cases opened yesterday”, “tell me about high severity cases” and many other utterances.

IAM is used in two places, firstly as a role for our Lambda to be able to interact with DynamoDB client and also grant it Cloudwatch logs permissions to write important details about the requests being received. Secondly, Lambda permission is needed to allow this custom Alexa skill to invoke its backend Lambda. As a best practice it is suggested to check whether requests coming to Lambda are from case organiser skill only, and no other skill can use this Lambda function.

Additionally in case of a critical getting opened, the lambda function uses SNS service to fan out push notification to user’s email hence alerting the stakeholders of the critical case. Along with email notification, Alexa device will also be able to notify the stakeholder. As a parallel functionality, a dashboard is also created using React frontend for ease of querying the AWS support case data which is getting stored in DynamoDb.

**Challenges:**

1. Proactive API implementation-

We wanted to program the Alexa device color to change its color to red on critical case arrival but the ability to play around with colors is not available to developers yet. But then we came across proactive events feature which allows the developers to send notifications to the users of their custom skill. When the skill successfully sends a notification to a user’s Alexa device, the user hears a chime sound which indicates that a new notification has arrived.

For this we had to re-deploy the skill using the ASK CLI with some changes which will enable the users to receive notifications. The developer needs to manage the permissions for the same by adding the notification.write permission to the manifest section in skill.json package file.

1. Providing cross account DynamoDb access keys-

When fetching support cases from one account, and keeping the DynamoDb database in other account, we had to create two separate java clients using two different key pairs. But we were unable to do so, reason being that Java supports only one key pair globally. So it was resolved by using the global key pair for the account we were fetching data from and the key pair for account having the shared database was then provided as strings locally.

Using Lambda function was also an option but that would have required interaction with two different APIs, so instead we used a single API to fetch new cases and inserted them in cross account DynamoDb, the second Java client.

1. AMAZON.SearchQuery slot-

We required the ability to trace all the cases around a particular keyword in the subject. The challenge was that for this if we define a custom slot, there was not any list of specific words that we could import as utterance for a slot value, the list was endless.

The solution was AMAZON.SearchQuery, it is a slot type which helps you to retrieve whole utterance phrase to your slot by getting the value in JSON response. This slot type provides the whole phrase as a slot value into lambda code which is then used as per skill's logic of matching the input with subject column data. It basically works as a search query string that can be entered into a search engine or for searching in a specific database like in this case study.