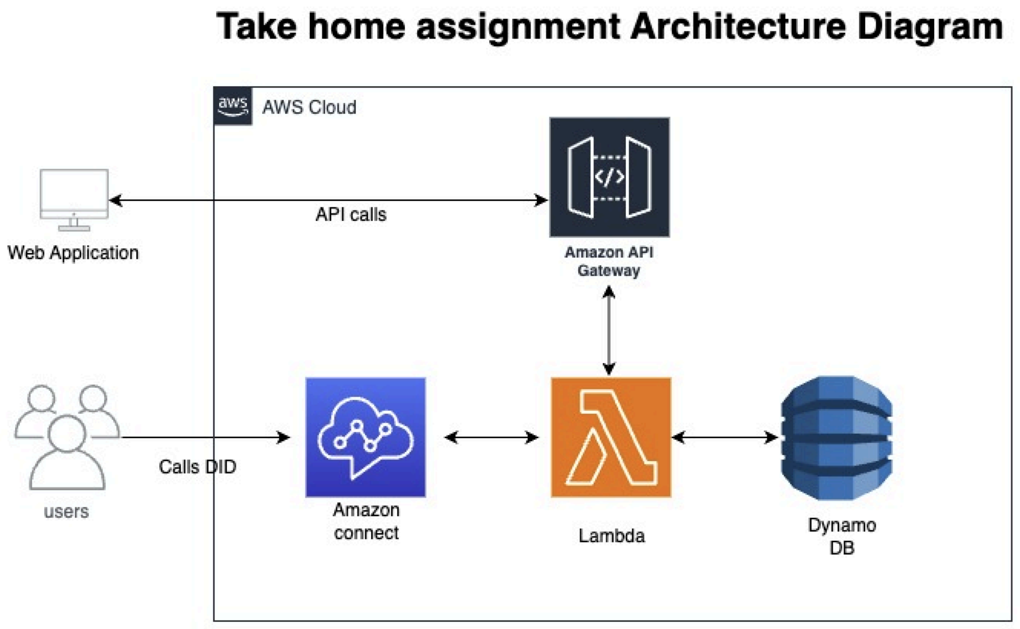
Take home Assessment Documentation



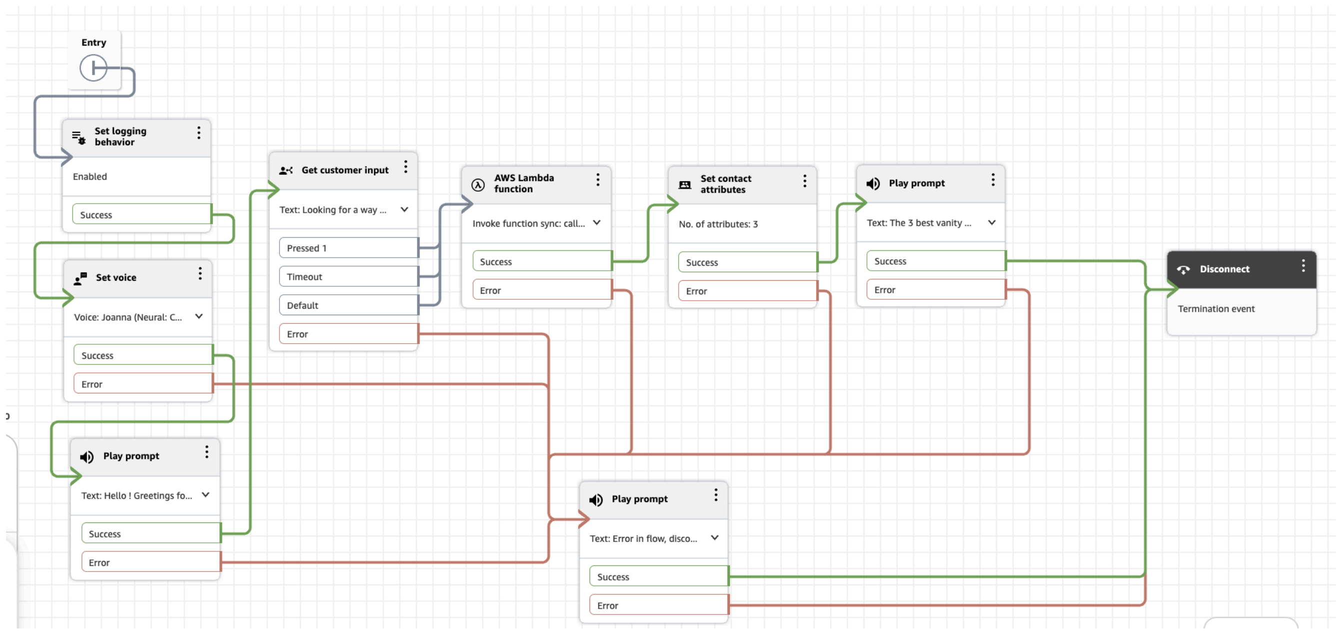
Architecture Overview

When a user places a call to a Direct Inward Dialing (DID) number or Toll-Free Number (TFN) associated with a configured Amazon Connect contact flow, the contact flow is triggered and executes the defined interaction logic. As part of this flow, a Lambda function is invoked to perform the business logic responsible for generating vanity numbers based on the caller's phone number.

The Lambda function receives the caller's number from Amazon Connect, computes the corresponding vanity number combinations, selects the top five results, and stores them in a DynamoDB table for persistence.

A separate web application, built as a single-page React application, is used to display these vanity number mappings. The application issues a GET request to an Amazon API Gateway endpoint, which in turn invokes another Lambda function. This Lambda function retrieves the stored vanity number records from DynamoDB and returns the data to the API Gateway. The API Gateway then provides the response back to the React application, where the results are rendered in the user interface.

Contact flow design



Contact Flow Design Explanation

The objective of this contact flow is to identify the caller's phone number and provide the three best vanity number alternatives generated by a Lambda function.

The flow begins with standard initialization steps, including enabling logging and setting voice parameters. A greeting message is then presented to the caller. Following the greeting, the caller is prompted to press 1 to hear the top three vanity number possibilities derived from their phone number.

When the user selects this option, the contact flow invokes a Lambda function. The function receives the caller's phone number from the event data passed by Amazon Connect. It processes the number, computes the vanity number combinations, and returns a string map containing the three best results. These values are stored in contact attributes within the flow, allowing them to be referenced in a subsequent Play Prompt block where the caller can hear their vanity number options.

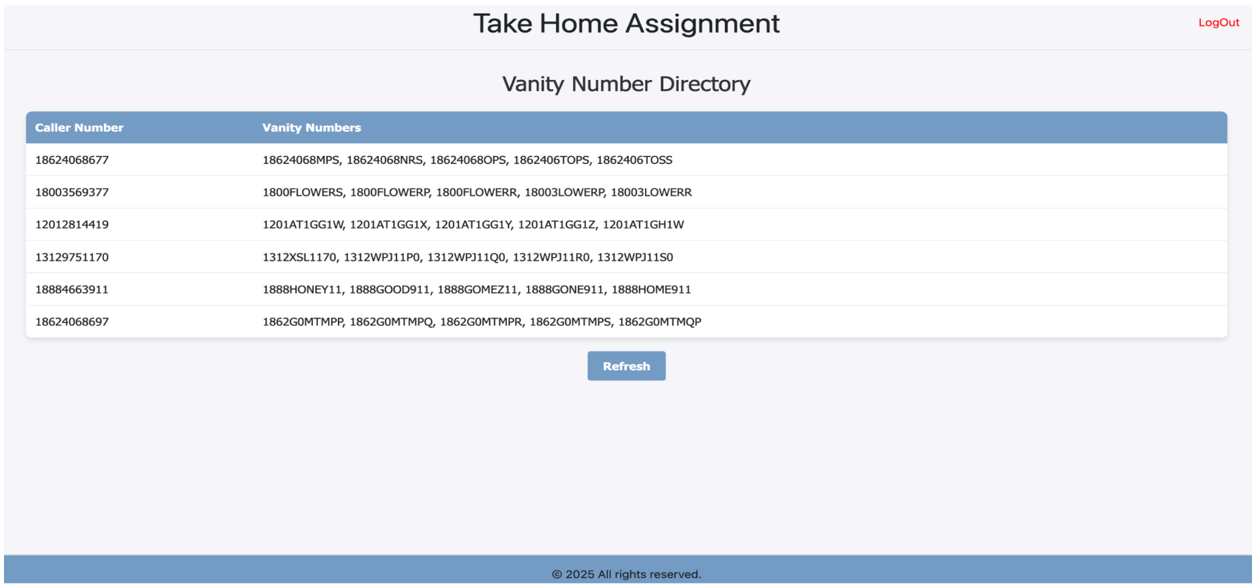
Potential Enhancements

Additional enhancements can further improve the functionality and user experience of the contact flow. One enhancement is to allow callers to enter an alternate phone number for which they would like to generate vanity number recommendations. The contact flow can capture this input and pass it to the Lambda function instead of using the automatically detected caller ID.

Another improvement would be to offer callers the option to replay or repeat the top three vanity number suggestions if they wish to hear them again. This ensures greater clarity and a more user-friendly interaction.

Furthermore, integrating an Amazon Lex bot would provide a more conversational and personalized experience. Leveraging natural language understanding would enable callers to interact with the system more intuitively, enhancing usability and overall customer satisfaction.

Web Application



Take Home Assignment

Vanity Number Directory

Caller Number	Vanity Numbers
18624068677	18624068MPS, 18624068NRS, 18624068OPS, 1862406TOPS, 1862406TOSS
18003569377	1800FLOWERS, 1800FLOWERP, 1800FLOWERR, 18003LOWERP, 18003LOWERR
12012814419	1201AT1GG1W, 1201AT1GG1X, 1201AT1GG1Y, 1201AT1GG1Z, 1201AT1GH1W
13129751170	1312XSL1170, 1312WPJ11P0, 1312WPJ11Q0, 1312WPJ11R0, 1312WPJ11S0
18884663911	1888HONEY11, 1888GOOD911, 1888GOMEZ11, 1888GONE911, 1888HOME911
18624068697	1862G0MTMPP, 1862G0MTMPQ, 1862G0MTMPR, 1862G0MTMPS, 1862G0MTMQP

© 2025 All rights reserved.

This single-page application retrieves data from DynamoDB and presents it in a tabular format. The interface also includes a refresh button that allows users to update the displayed data on demand. Potential enhancements include adding functionality that enables users to input a phone number directly within the application and receive the corresponding vanity number results in real time, thereby improving interactivity and overall user experience.

Lambda implementation :

Vanity Number Generation Logic – Technical Explanation

Python module implements a hybrid vanity number generation approach that combines a third-party library with a custom fallback algorithm. The goal is to reliably produce exactly **five** vanity number variations for any input phone number, ensuring consistent results even when external libraries return incomplete or empty outputs.

The primary mechanism for generating vanity numbers uses the open-source package `vanitynumber 0.5`, which is designed to compute meaningful word combinations based on standard telephone keypad mappings. This library works well for most inputs, producing high-quality vanity wordifications.

Initially, a local custom brute-force implementation was attempted. However, iterating through digit-to-letter permutations produced extremely large and impractical search spaces, yielding results that were mostly meaningless. After researching more effective approaches, the `vanitynumber` package was identified as a reliable foundation for generating word-based vanity numbers.

During Lambda development, the library initially failed due to packaging and import-path issues. These were resolved by updating portions of the library's internal `vanitynumber.py` file and packaging the module into an AWS Lambda Layer in the correct folder structure . Attached both the Lambda and the local Python code for reference.

Challenges

One of the primary challenges encountered during the implementation was the inability to claim a phone number within the AWS account. Despite not having any previously allocated numbers, the system consistently returned a *"quota exceeded"* error when attempting to claim a DID or toll-free number. A support case was opened with AWS to address the issue; however, a resolution has not yet been provided. As a result, full end-to-end testing of the contact flow using an actual inbound call could not be completed.