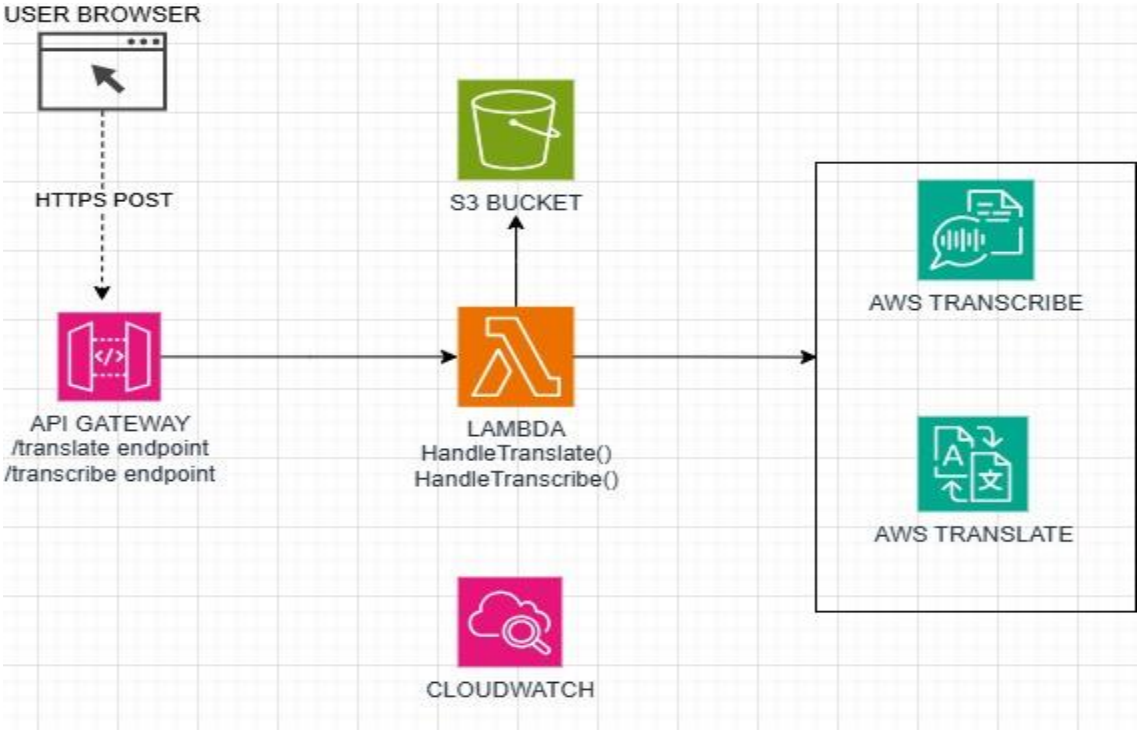
Multilingual Voice Assistant using AWS Transcribe & AWS Translate

Project Index

1. Project Overview

This project implements a multilingual voice assistant that allows users to record audio in the browser, a base64-encoded audio is sent through an API Gateway endpoint to a Lambda function, and have the audio stored in S3 and transcribed using Amazon Transcribe. The Lambda function polls the transcription job until completion, retrieves the transcription JSON from an S3 URI using Node.js HTTPS module, extracts the text, and returns it to the frontend. The user can then submit the transcribed text to another API endpoint where AWS Translate processes it and returns the translated output. This system provides a streamlined, serverless workflow for audio recording, speech-to-text processing, and multilingual translation.

2. Architecture Diagram



Transcription Flow

Microphone → MediaRecorder API → Base64 Encoding → API Gateway (/transcribe) → Lambda (handleTranscribe) → S3 Upload → AWS Transcribe Job → Polling Loop → HTTPS Fetch Transcript → Parse JSON → Lambda → API Gateway → Browser

Translation Flow

Translate button → API Gateway (/translate) → Lambda (handleTranslate) → AWS Translate → Lambda → API Gateway → Browser

3. AWS Services Used

- **API Gateway** – REST endpoints
- **Lambda** – Node.js functions for transcription & translation
- **S3** – audio storage and transcription job output
- **Amazon Transcribe** – speech-to-text

- **Amazon Translate** – language translation
- **Cloudwatch** - For logging to troubleshoot issues

4. Frontend Workflow

Record audio

- **Use MediaRecorder API**
- **Convert audio Blob to base64**

Transcribe UI

- Button: **Record and stop to transcribe**
- Text area: **Shows transcription**

Translate

- Button: **Translate**
- **Shows translated output**

5. Backend Workflow

For /transcribe

1. API Gateway receives base64 audio and is sent to the lambda function.
2. Lambda decodes base64 audio to a file.
3. Uploads to S3.
4. Starts Transcribe job.
5. Polls transcription job till completion.
6. When completed, job output contains an S3 URI.
7. Lambda uses **https** module to fetch JSON output from the URI.
8. Extracts results (transcribed data).
9. Returns transcription back to the user interface.

For /translate

1. Takes text input from frontend.
2. Calls Amazon Translate.
3. Returns translated text.

6. Deployment With Terraform

The entire serverless architecture, including the S3 bucket, Lambda functions, API Gateway endpoints, IAM roles, and necessary permissions can be provisioned and managed using Terraform. Full configuration files can be found in this GitHub repo.

7. Cost Considerations

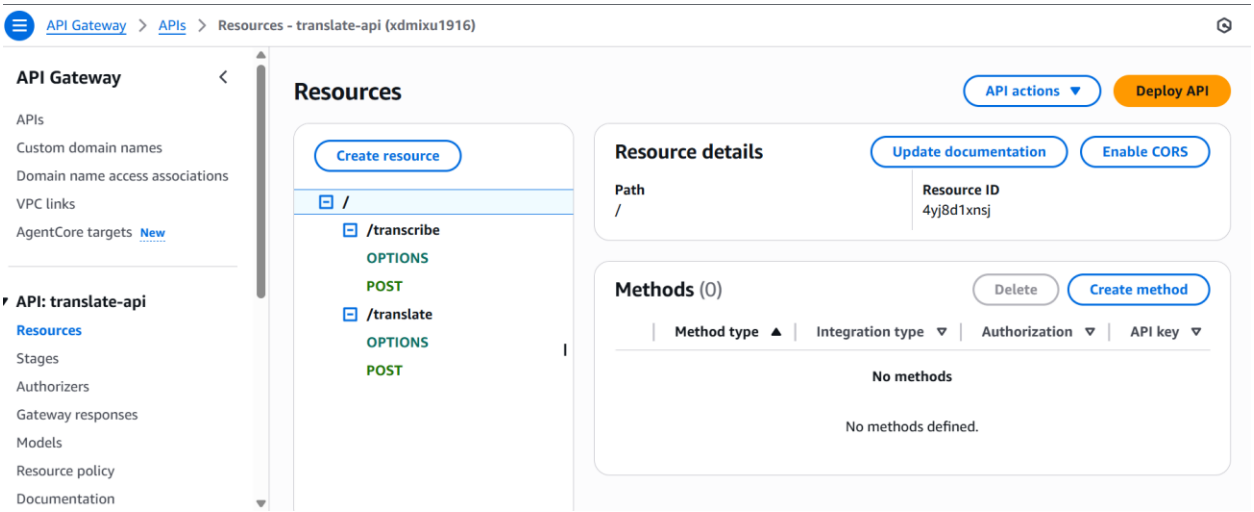
This solution is built entirely on serverless services, meaning costs scale based on usage and can remain very low for testing or low-traffic applications. AWS Lambda charges per millisecond of compute time, and typical transcription or translation workloads will cost only fractions of a dollar unless invoked at very high volume. Amazon Transcribe charges per minute of processed audio, with pricing based on the audio duration rather than transcription results. Amazon Translate charges per million characters processed, making short text translations inexpensive. S3 storage costs apply for storing audio files and transcription outputs, though typical storage usage for short audio clips is negligible. API Gateway charges per API call, making it affordable for small apps but potentially a cost factor for high-frequency workloads. Overall, the architecture is cost-efficient, but setting lifecycle policies for S3 storage, monitoring usage with CloudWatch and setting budget alerts is recommended.

8. Future Enhancements

Several improvements can enhance the system's functionality and user experience. One potential enhancement is integrating Amazon Comprehend to analyze the sentiment or extract key entities from the transcribed text. Another improvement is implementing automatic language detection before translation to reduce user input and streamline the workflow. For richer audio processing, you could replace the batch Transcribe job with Amazon Transcribe Streaming, allowing near real-time transcription with WebSocket connections. Adding authentication with Amazon Cognito would secure API endpoints and prevent unauthorized usage. Finally, the application could be expanded by introducing a database such as DynamoDB to store transcripts, translations, and user history for analytics and improved user experience.

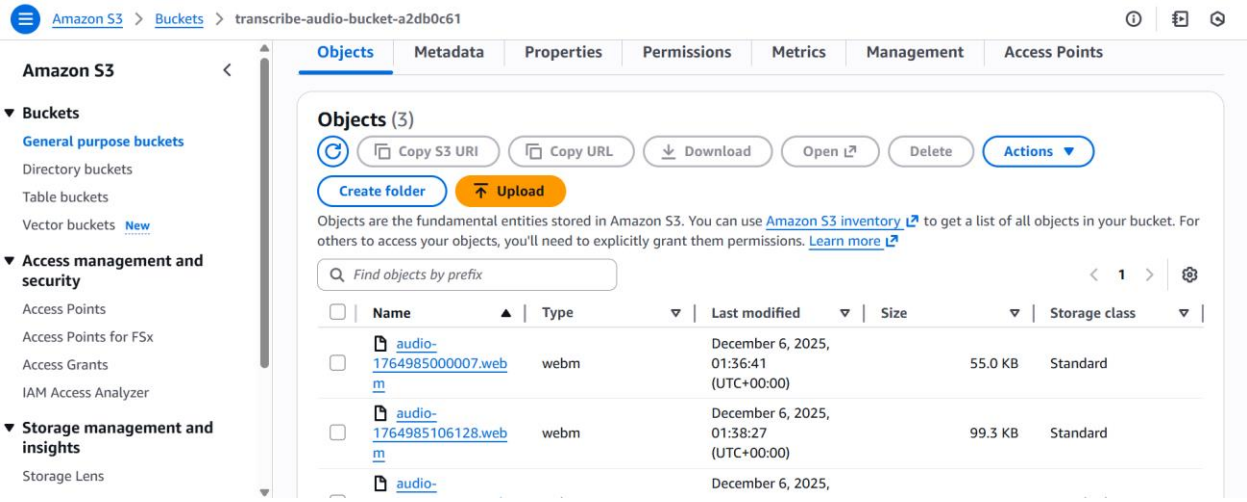
9. Proof of Concept

Provision of API Gateway resources.

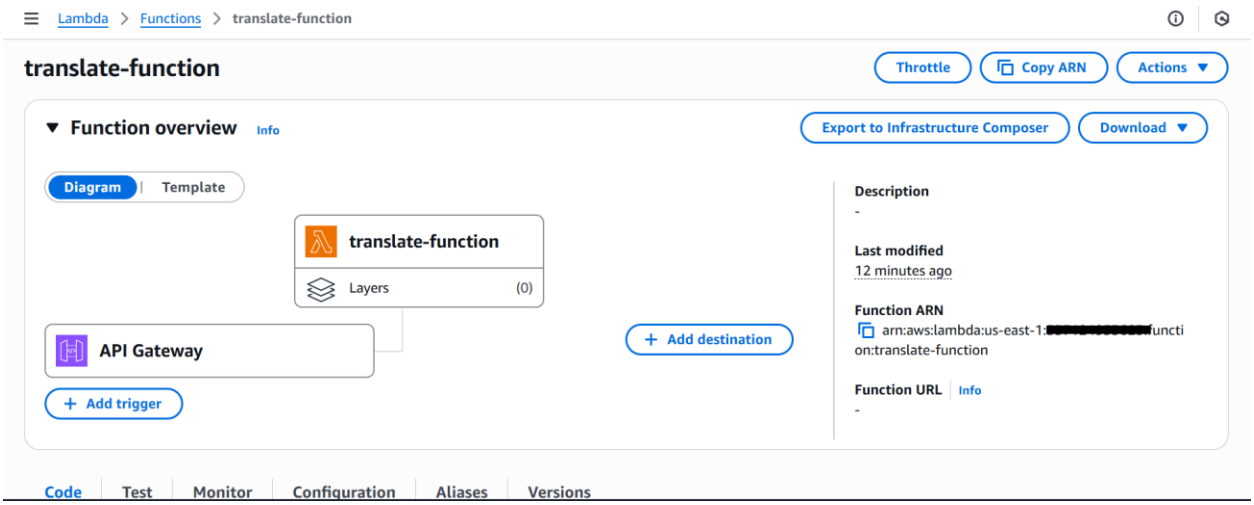


Full demo video here

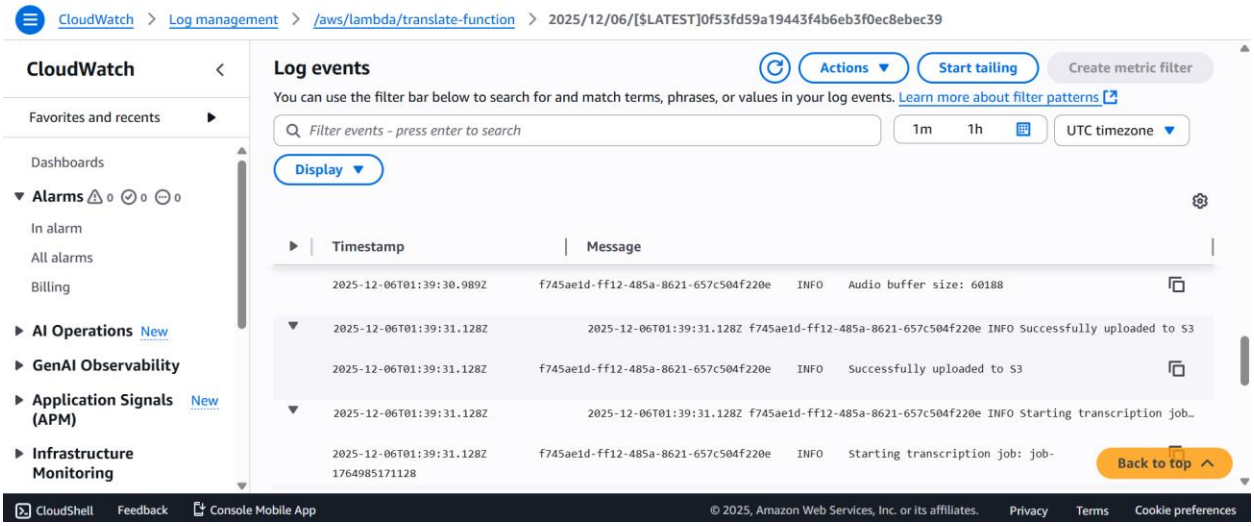
S3 bucket showing uploaded audio in webm format.

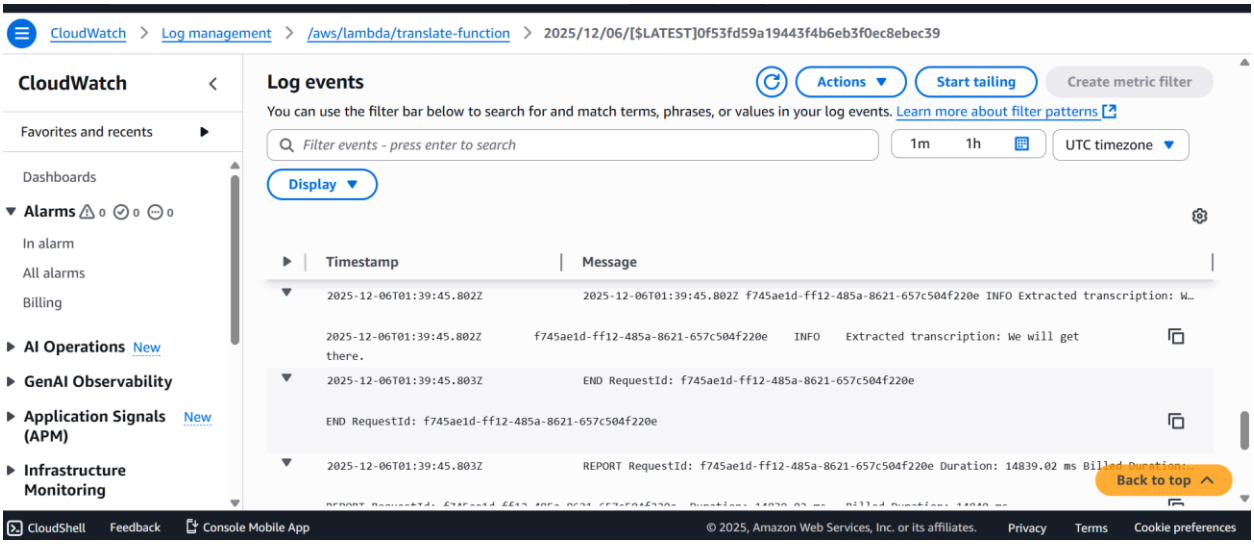


Provision of Lambda function and cloudwatch showing logs for debugging and monitoring.

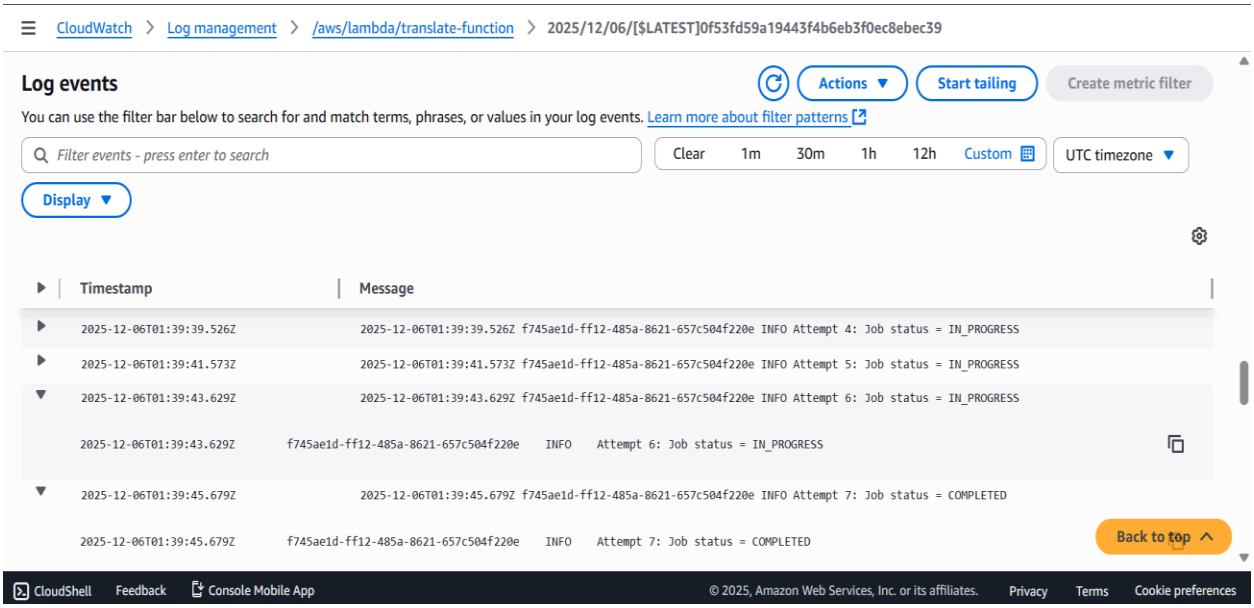


Full demo video [here]



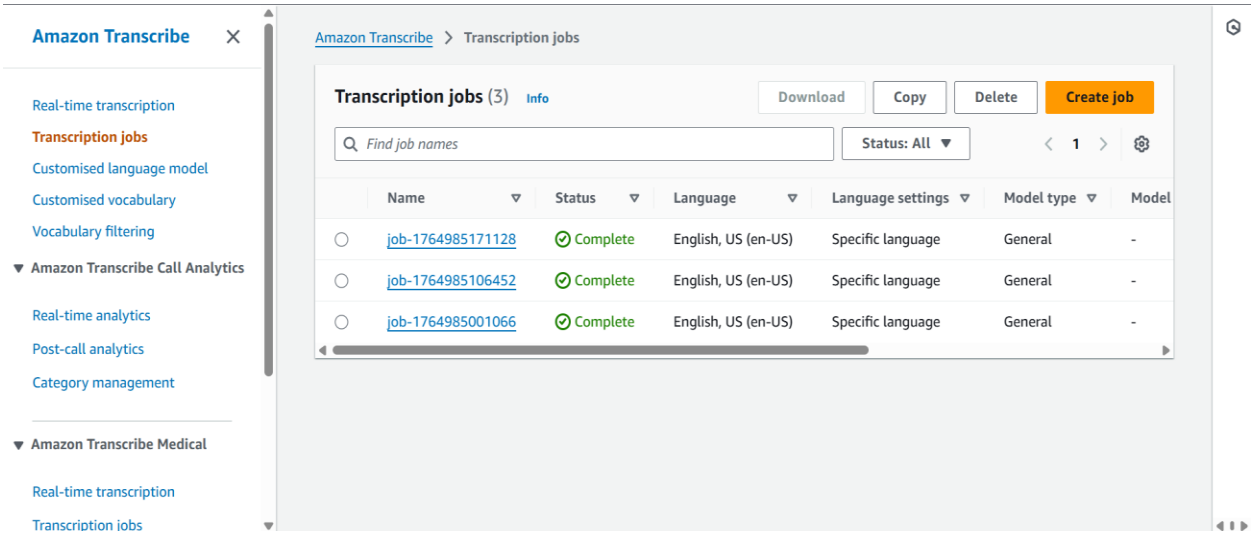


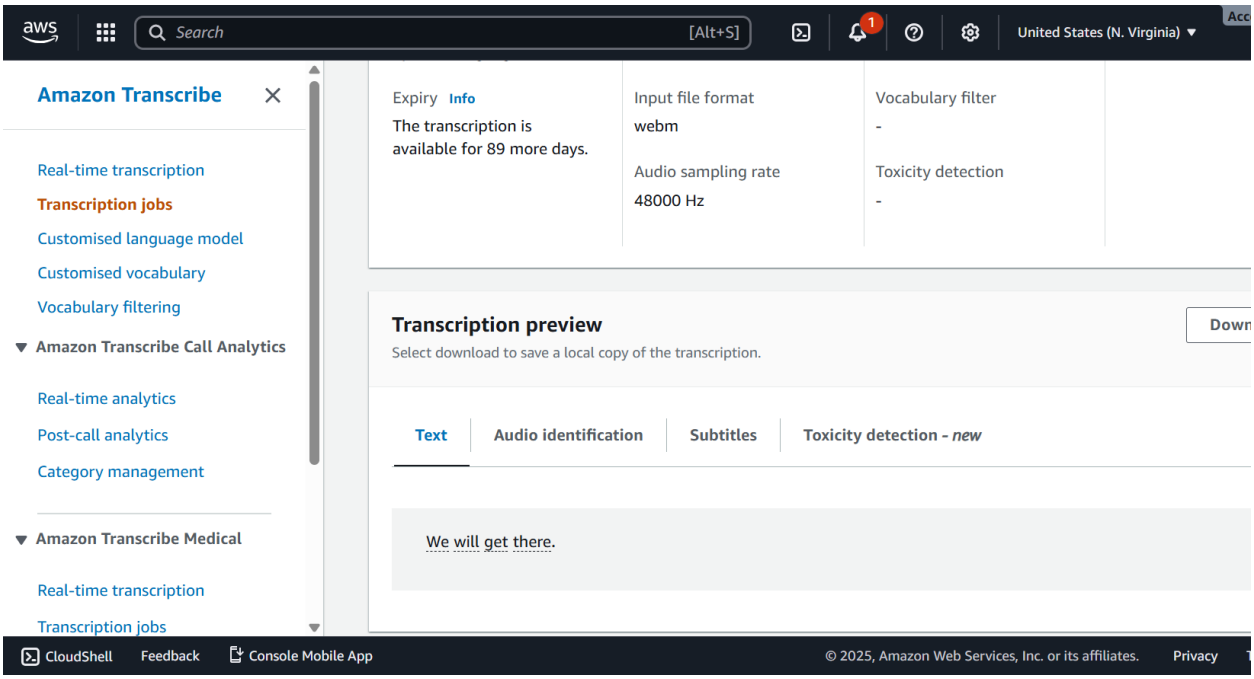
Full demo video here



Full demo video [here]

Provision of Transcribe resource showing various job IDs and transcribed data.





Full demo video here