Documento Técnico: Prototipo de Voicebot con Capacidades de Agente

Versión: 1.0

Fecha: 23 de Mayo, 2025

Autor: Sebastian Diaz Gaviria

1. Introducción y Objetivos

Este documento describe la arquitectura, diseño y decisiones técnicas tomadas durante el desarrollo del "Prototipo de Voicebot con Capacidades de Agente". El objetivo principal del proyecto fue construir un sistema capaz de recibir instrucciones por voz, interpretarlas mediante procesamiento de lenguaje natural, ejecutar una lógica de agente interna (incluyendo consulta a fuentes de datos y simulación de APIs), y responder al usuario tanto en formato textual como en voz.

El caso de uso implementado es un **Asistente de Compras Inteligente**, que permite a los usuarios realizar consultas sobre productos, simulando una interacción de compra básica.

Este prototipo fue desarrollado siguiendo un enfoque progresivo a través de tres niveles de madurez, culminando en un sistema que integra captura de voz, procesamiento en un backend en la nube, y generación de respuesta en voz.

2. Descripción General de la Arquitectura

El sistema final (correspondiente al Nivel 3) sigue una arquitectura cliente-servidor basada en servicios de AWS y una interfaz web para la interacción del usuario.

- **Cliente (Frontend):** Una página HTML estática (voice_ui.html) que utiliza JavaScript para capturar audio del micrófono del usuario a través de la API MediaRecorder del navegador. Esta UI envía la grabación de voz al backend.
- **API Gateway (HTTP API):** Actúa como el punto de entrada seguro y escalable para las solicitudes del frontend. Expone un endpoint POST que recibe el audio.
- **Backend (AWS Lambda):** Una función Python (lambda_function.py) que orquesta el flujo principal de procesamiento:
    1. **Recepción y Decodificación de Audio:** Recibe el audio (codificado en base64) de API Gateway.
    2. **Voz a Texto (STT):** Utiliza **Amazon Transcribe** para convertir el audio hablado en texto. El audio se sube temporalmente a S3 para que Transcribe lo procese.

3. **Procesamiento por Agente Inteligente:** El texto transcrito se pasa a un agente construido con **LangGraph**.
    - **NLU (Natural Language Understanding):** Se utiliza un modelo fundacional de **Amazon Bedrock** (específicamente, amazon.titan-text-express-v1) para interpretar la intención del usuario y extraer entidades relevantes (ej. tipo de producto, marca).
    - **Lógica de Consulta:** El agente consulta un catálogo de productos simulado (un archivo products.json empaquetado con la Lambda) basándose en las entidades extraídas.
    - **Generación de Respuesta:** Se utiliza nuevamente Amazon Bedrock para generar una respuesta textual coherente basada en los resultados de la consulta o el estado de la interacción.
4. **Texto a Voz (TTS):** La respuesta textual del agente se convierte en audio hablado utilizando **Amazon Polly** (voz Lupe).
5. **Respuesta a la API:** La función Lambda devuelve la respuesta textual y el audio de Polly (codificado en base64) a API Gateway.

- **Interfaz de Usuario (Respuesta):** La 1voice_ui.html recibe la respuesta del backend, muestra el texto y reproduce el audio de la respuesta del agente.
- **Almacenamiento (S3):** Utilizado para el almacenamiento temporal de archivos de audio para Transcribe y para alojar el paquete ZIP de la capa de Lambda.
- **Capa de Lambda:** Las dependencias de Python (LangChain, LangGraph, etc.) se gestionan a través de una capa de Lambda para mantener el paquete de despliegue de la función principal más ligero y organizado.

3. Diagrama Funcional

(Aquí deberías insertar una imagen de tu diagrama de flujo conceptual, similar al que discutimos para el Nivel 1, pero actualizado para reflejar el flujo completo de voz a voz con los servicios de AWS).

Flujo Principal:

1. **Usuario:** Habla una consulta en la 1voice_ui.html.
2. **UI (1voice_ui.html):**
    - Captura audio con MediaRecorder.
    - Envía el audio (base64) al endpoint de API Gateway.

3. **API Gateway:**
    - Recibe la solicitud POST.
    - Autoriza (si está configurado, aunque en este prototipo es abierto) y enruta a la función Lambda.
4. **AWS Lambda (lambda_function.py):**
    - **Transcribe:** Audio -> S3 -> Amazon Transcribe -> Texto.
    - **Agente LangGraph:**
        - Input: Texto transcrito.
        - Nodo NLU (Bedrock): Texto -> Intención + Entidades.
        - Nodo Catálogo: Entidades -> Consulta products.json -> Productos Encontrados.
        - Nodo Generación Respuesta (Bedrock): Productos Encontrados -> Respuesta Textual.
        - Output: Respuesta Textual.
    - **Polly:** Respuesta Textual -> Amazon Polly -> Audio de Respuesta.
    - Devuelve JSON { TextoRespuesta, AudioRespuestaBase64, LogLlamadas } a API Gateway.
5. **API Gateway:** Devuelve la respuesta JSON a la UI.
6. **UI (voice_ui.html):**
    - Muestra el TextoRespuesta.
    - Reproduce el AudioRespuesta.

4. Decisiones Técnicas y Tecnologías Utilizadas

- **Frontend (HTML + JS):** Se eligió por simplicidad para un prototipo, permitiendo una rápida implementación de la captura de audio básica. Tailwind CSS se usó para un estilo rápido.
- **Backend Serverless (Lambda + API Gateway):** Proporciona escalabilidad, gestión reducida y pago por uso, ideal para un prototipo y aplicaciones de este tipo. HTTP API se eligió sobre REST API por su simplicidad y menor costo para este caso de uso.
- **Amazon Transcribe:** Servicio gestionado y robusto para la conversión de voz a texto, con buena precisión para español.

- **Amazon Bedrock (Titan Text Express):** Se seleccionó por su disponibilidad, facilidad de integración y capacidades tanto para NLU (extracción de intención/entidades mediante prompting) como para la generación de respuestas en lenguaje natural. La elección de Titan sobre otros modelos se basó en la necesidad de resolver problemas de acceso a modelos de terceros durante el desarrollo.
- **LangGraph:** Se utilizó para orquestar los pasos del agente inteligente (NLU, consulta de herramientas/datos, generación de respuesta). Proporciona una forma estructurada y modular de construir la lógica del agente.
- **Catálogo Simulado (products.json):** Para cumplir con el requisito de interactuar con una fuente de datos estructurada sin la complejidad de configurar una base de datos externa para el prototipo. Se empaqueta con la Lambda.
- **Amazon Polly:** Servicio gestionado para la síntesis de texto a voz, ofreciendo voces naturales y facilidad de uso. Se eligió la voz "Lupe" (es-US, Neural) por su calidad.
- **Python:** Lenguaje principal para el backend (Lambda) y los notebooks de desarrollo, debido a su robusto ecosistema para IA/ML y las SDKs de AWS (Boto3) y LangChain.
- **Capa de Lambda:** Para gestionar las dependencias de Python de manera eficiente, separándolas del código de la función principal.

5. Niveles de Madurez Alcanzados

- **Nivel 1 (Lógica de Agentes Text-to-Text):** Se completó en notebooks/nivel1_text_agent.ipynb, demostrando la orquestación de un agente con LangGraph y Bedrock para procesar texto, consultar un catálogo simulado y generar respuestas textuales, incluyendo logs básicos de rendimiento.
- **Nivel 2 (Refinamiento Semántico y Capa Cognitiva con Voz a Texto):** Se desarrolló en notebooks/nivel2_voice_agent.ipynb, incorporando Amazon Transcribe para procesar archivos de audio. Se refinó el NLU para mejorar la extracción de entidades (ej. marca "SuperVision", categoría "botas de montaña") y la lógica de consulta al catálogo. Se añadió Amazon Polly para la generación de respuesta en audio. Se creó una UI básica (voice_ui.html) para la captura de voz.
- **Nivel 3 (Conversación en Tiempo Real - Prototipo):** Se integraron los componentes del Nivel 2 en una arquitectura cliente-servidor con API Gateway y AWS Lambda. La voice_ui.html ahora envía el audio capturado al backend y recibe una respuesta en texto y voz, simulando un ciclo de interacción por voz.

6. Posibles Extensiones Futuras del Prototipo

- **Conversación Multi-Turno:** Implementar memoria y manejo de estado en el agente LangGraph para permitir conversaciones más largas y contextuales.

- **Streaming de Audio en Tiempo Real:**
    - Utilizar **LiveKit** para transmitir audio bidireccionalmente entre la UI y el backend, reduciendo la latencia y permitiendo interacciones más naturales
    - Integrar Amazon Transcribe Streaming y Amazon Polly Streaming.
- **Arquitectura Asíncrona para Procesos Largos:** Para tareas como la transcripción de audios más largos, implementar un patrón asíncrono (ej. SQS + Lambda, Step Functions) para evitar timeouts en API Gateway y mejorar la experiencia del usuario.
- **Conexión a Bases de Datos Reales:** Reemplazar products.json con una base de datos escalable como Amazon DynamoDB o Amazon RDS para el catálogo de productos.
- **Integración con APIs Externas Reales:** Conectar el agente a APIs de proveedores de comercio electrónico, sistemas de inventario, etc.
- **Interfaz de Usuario Avanzada:** Desarrollar una UI más rica con un framework como React, Vue o Angular, mejorando el feedback visual, el historial de conversación y la presentación de productos.
- **Seguridad Robusta:** Implementar autenticación y autorización en API Gateway (ej. con Amazon Cognito, Autorizadores Lambda o IAM).
- **Personalización y Recomendaciones Avanzadas:** Utilizar servicios como Amazon Personalize para ofrecer recomendaciones de productos más sofisticadas.
- **Manejo de Errores y Resiliencia Mejorados:** Implementar patrones de reintento, colas de mensajes fallidos (dead-letter queues), y un feedback de error más granular al usuario.
- **Pruebas y Monitoreo Continuo:** Establecer un framework de pruebas unitarias y de integración, y un monitoreo más exhaustivo con CloudWatch.

7. Consideraciones de Seguridad, Networking y Limitaciones

Seguridad

- **API Gateway:** Actualmente, el endpoint es público. En producción, se requeriría autenticación (ej. Amazon Cognito, claves de API, autorizadores Lambda) para proteger el acceso. Las políticas de CORS también deberían restringirse a dominios específicos.
- **Permisos IAM:** Se siguieron los permisos necesarios para cada servicio de AWS. En producción, se aplicarían estrictamente los permisos de menor privilegio para el rol de la función Lambda y otros roles.
- **Datos Sensibles:** No se manejan datos de usuario sensibles en este prototipo. Si se hiciera (ej. información de pago), se requerirían medidas de encriptación y cumplimiento normativo adicionales.

- **Validación de Entradas:** La Lambda debería validar y sanitizar las entradas recibidas de API Gateway para prevenir vulnerabilidades.

Networking

- **VPC:** Para una mayor seguridad, la función Lambda y otros recursos como bases de datos podrían desplegarse dentro de una Amazon Virtual Private Cloud (VPC), con endpoints de VPC para acceder a servicios de AWS sin pasar por la internet pública.
- **Latencia:** La latencia actual depende de la suma de los tiempos de Transcribe, Bedrock, Polly y la red. Para una conversación en tiempo real, el streaming y la optimización de cada paso serían cruciales.

Limitaciones y Riesgos

- **Timeout de API Gateway:** La arquitectura síncrona actual tiene un límite de ~29 segundos impuesto por HTTP API Gateway. Audios largos o procesamientos complejos pueden exceder este límite. (Mitigación: Arquitectura asíncrona).
- **Precisión del NLU y STT:** La precisión de Transcribe y del NLU de Bedrock puede variar según la calidad del audio, acentos, ruido de fondo y la complejidad de la consulta. (Mitigación: Refinamiento de prompts, entrenamiento de modelos personalizados si es necesario, guías al usuario sobre cómo hablar).
- **Gestión de Estado:** El agente actual es en gran medida sin estado para cada interacción. Una conversación fluida requiere una gestión de estado más sofisticada.
- **Costos:** El uso de múltiples servicios de AWS (Bedrock, Transcribe, Polly, Lambda, API Gateway, S3) incurre en costos. Para producción, se requeriría un análisis y optimización de costos.
- **Escalabilidad:** Aunque Lambda y API Gateway son escalables, los servicios dependientes (como Bedrock) tienen cuotas y límites que deben considerarse para cargas altas.
- **Manejo de Errores:** El manejo de errores actual es básico. Un sistema de producción necesitaría una estrategia de reintentos más robusta y un mejor feedback al usuario en caso de fallos.

Este prototipo demuestra con éxito la viabilidad de un asistente de voz inteligente utilizando servicios de AWS y sienta las bases para futuras mejoras y desarrollos hacia una solución de producción completa.