

# Plan de Implementación y Arquitectura Técnica: "El Círculo Kostik" en Azure con Interledger y AI Foundry

## Sección 1: Arquitectura de Solución Unificada: "El Círculo Kostik"

### 1.1. Visión Arquitectónica y Flujo de Datos

La arquitectura de "El Círculo Kostik" se concibe como un conjunto de microservicios *serverless* y contenerizados, desplegados íntegramente en la plataforma Azure. Esta arquitectura está diseñada para ser escalable, segura y rápida de implementar, en línea con los objetivos de un Producto Mínimo Viable (MVP) de alto impacto.<sup>1</sup> La solución desacopla la lógica de la interfaz conversacional de la lógica transaccional del *ledger*, permitiendo el desarrollo en paralelo de los dos equipos.

La arquitectura general interactúa de la siguiente manera <sup>2</sup>:

1. **Capa de Interfaz (Equipo 1):** Un *sandbox* de **Twilio para WhatsApp** <sup>4</sup> actúa como el canal de comunicación. Este servicio envía webhooks a una **Azure Function App (Node.js)** <sup>6</sup>, que orquesta la conversación. Esta Function App se comunica con **Azure AI Foundry** <sup>7</sup> para los servicios de IA (Voz/Idioma).
2. **Capa de Negocio (Equipo 2):** Una segunda **Azure Function App (Node.js)** <sup>9</sup> expone una API REST segura que maneja toda la lógica de negocio y las transacciones.
3. **Capa de Datos (Equipo 2):** Una base de datos **Azure Database for MySQL (Flexible Server)** <sup>10</sup> sirve como el *ledger* virtual o "Wallet de Confianza" <sup>1</sup>, almacenando los saldos y estados de los productores.
4. **Capa de Pagos (Equipo 2):** **Interledger Rafiki** <sup>12</sup>, desplegado en **Azure Container Instances (ACI)** <sup>13</sup>, gestiona el wallet maestro para la recepción de fondos (como Web Monetization).<sup>1</sup>

A continuación, se detallan los flujos de datos críticos del sistema.

#### Flujo de Datos 1: Voz a Transacción (Ejemplo: Consulta de Saldo)

Este es el flujo síncrono principal que demuestra la integración de IA <sup>1</sup>:

1. El productor graba un mensaje de voz ("¿cuánto tengo?") y lo envía por WhatsApp.
2. **Twilio** recibe el mensaje y dispara un webhook HTTP POST a la Azure Function (Interfaz) del Equipo 1. La solicitud de Twilio incluye una URL pública al archivo de audio (MediaUrlO).<sup>15</sup>
3. La **Azure Function (Interfaz)** <sup>6</sup> recibe el webhook, detecta la presencia de MediaUrlO <sup>1</sup> e inicia el flujo de IA.
4. La función llama a **Azure AI Foundry** <sup>7</sup>, específicamente al servicio de **Azure AI Speech (Speech-to-Text)**.<sup>16</sup>
5. Se pasa la MediaUrlO directamente al servicio STT de Azure (que puede procesar audio desde URLs).<sup>17</sup> El servicio transcribe el audio y devuelve el texto: "cuánto tengo".
6. La Function (Interfaz) realiza un **mapeo de intención** (intent mapping) <sup>19</sup>, convirtiendo el texto en un comando de programa (ej. INTENT\_SALDO).
7. La Function (Interfaz) realiza una llamada HTTP GET segura (usando una clave de función <sup>21</sup>) al endpoint del Equipo 2: GET /api/balance?telefono\_wa=+52....
8. La **Azure Function (API de Wallet)** recibe la solicitud, consulta el saldo en **Azure Database for MySQL** <sup>23</sup> y devuelve una respuesta JSON: {"balance": 500.00}.
9. La Function (Interfaz) recibe el JSON y genera un texto de respuesta: "Tienes 500 pesos".
10. Este texto de respuesta se envía al servicio **Azure AI Speech (Text-to-Speech)** <sup>25</sup> dentro de AI Foundry.
11. El servicio TTS devuelve una URL a un archivo.mp3 <sup>1</sup> con la respuesta de voz.
12. La Function (Interfaz) responde al webhook de Twilio, instruyéndole a enviar un nuevo mensaje de WhatsApp al productor, esta vez adjuntando el.mp3 (mediaUrl).<sup>15</sup>

### Flujo de Datos 2: Asíncrono (Ejemplo: Ventas y Tandas)

1. **Ventas (Webhook Externo):** Un cliente completa una compra en WooCommerce. El *hook woocommerce\_order\_status\_completed* <sup>1</sup> se activa y envía un webhook <sup>28</sup> al endpoint /api/credit-sale de la Azure Function (API de Wallet) del Equipo 2. Esta función acredita el saldo\_mxn al productor correspondiente en MySQL.
2. **Web Monetization (Simulado):** Los fondos se acumulan en el wallet maestro de Rafiki <sup>1</sup> (desplegado en ACI <sup>13</sup>). Una llamada manual al endpoint simulador /api/distribute-wm <sup>1</sup> en la API del Equipo 2 distribuye estos fondos entre los productores en la base de datos MySQL.
3. **Tandas (Trigger Interno):** Una **Azure Function (Timer Trigger)** <sup>30</sup> configurada con una expresión CRON (ej. "todos los viernes a las 8 AM") <sup>32</sup> se activa automáticamente. Esta función ejecuta una transacción de base de datos <sup>33</sup> que debita el aporte de todos los miembros de la tanda y acredita la suma total al beneficiario de la semana.<sup>1</sup>

## 1.2. Límites de Microservicios y Responsabilidades de los Equipos

La división de responsabilidades solicitada es fundamental para el éxito del proyecto. Los

límites de los microservicios se definen de la siguiente manera:

- **Equipo 1 (Interfaz Conversacional):**
  - **Componentes Propios:** Una Azure Function App (Node.js) que aloja el endpoint `/api/whatsapp-hook`.
  - **Responsabilidades Clave:**
    - Gestionar la configuración del **Twilio Sandbox**.<sup>4</sup>
    - Orquestar la conversación y ser el *cliente* de **Azure AI Foundry**.<sup>7</sup>
    - Implementar el flujo de **Speech-to-Text** <sup>16</sup> y **Text-to-Speech**.<sup>25</sup>
    - Gestionar el estado de la conversación (ver Sección 1.3).
    - Actuar como *cliente* de la API de Wallet del Equipo 2.
  - **Métrica de Éxito:** La fluidez, precisión y naturalidad de la conversación de voz y texto con el productor.<sup>1</sup>
- **Equipo 2 (API de Wallet y Ledger):**
  - **Componentes Propios:** Una Azure Function App (Node.js) <sup>9</sup> que aloja la API REST (ej. `/api/balance`, `/api/p2p-payment`); la base de datos **Azure Database for MySQL** <sup>10</sup>; y la instancia de **Rafiki en ACI**.<sup>13</sup>
  - **Responsabilidades Clave:**
    - Definir y asegurar la integridad del esquema de la base de datos (el *ledger* virtual).
    - Implementar toda la lógica de negocio (P2P, Retiros, Tandas, Ventas).
    - Garantizar la **atomicidad** de las transacciones financieras (usando transacciones SQL).<sup>33</sup>
    - Implementar la seguridad del PIN (hashing).<sup>34</sup>
    - Desplegar Rafiki <sup>12</sup> y exponer los webhooks para WooCommerce.<sup>28</sup>
  - **Métrica de Éxito:** La seguridad, fiabilidad y atomicidad de cada transacción. Debe ser el "motor de confianza".<sup>1</sup>

### 1.3. Estrategia de Seguridad y Cumplimiento (MVP)

Para un "Wallet de Confianza" <sup>1</sup>, la seguridad no es opcional, incluso en un MVP.

- **Seguridad de API (Función a Función):** La comunicación entre la Function (Interfaz) del Equipo 1 y la Function (API) del Equipo 2 *no debe ser anónima*. Se protegerá utilizando **Claves de Función (Function Keys)** de Azure.<sup>21</sup> La API del Equipo 2 requerirá una clave de host, que se almacenará de forma segura en la configuración de la aplicación del Equipo 1.
- **Seguridad de Base de Datos (Función a BD):** La Azure Function (API de Wallet) *no debe* almacenar cadenas de conexión de base de datos en su configuración. En su lugar, se configurará una **Identidad Administrada (Managed Identity)** <sup>35</sup> para la Function App del Equipo 2. A esta identidad se le otorgarán permisos de acceso (CONNECT, SELECT, UPDATE, etc.) directamente en el servidor Azure MySQL.<sup>37</sup> Esto

elimina la necesidad de gestionar credenciales de base de datos.<sup>35</sup>

- **Seguridad de Datos del Usuario (PIN):** El PIN de 4 dígitos <sup>1</sup> NUNCA debe almacenarse en texto plano. La API del Equipo 2 implementará un *hash + salt* usando la biblioteca *bcrypt* en Node.js.<sup>34</sup>
  1. **Flujo de Creación:** El usuario envía su PIN "1234". La API calcula *hash = bcrypt.hash("1234", salt)* y almacena el hash (ej. \$2b\$10\$...) en la columna *pin\_hash*.<sup>41</sup>
  2. **Flujo de Verificación:** El usuario envía "1234" para un pago. La API recupera el *pin\_hash* de la BD y ejecuta *bcrypt.compare("1234", pin\_hash)*. Esta función devuelve *true* o *false* sin revelar el PIN.<sup>41</sup>

### Gestión de Estado en un Entorno *Stateless*

Dado que Azure Functions es una plataforma *stateless* (sin estado) <sup>42</sup>, un desafío clave es gestionar conversaciones de múltiples pasos, como el flujo "PAGAR" -> "INGRESA PIN".<sup>1</sup> Si un usuario envía "1234", el bot no tiene contexto para saber que es un PIN.

En lugar de introducir servicios de estado complejos como Azure Bot Service <sup>44</sup> o Durable Functions <sup>46</sup>, lo cual excede el alcance de un hackathon de 5 horas, se aprovechará la base de datos MySQL existente.<sup>1</sup> Este enfoque es ligero y centra el estado del usuario junto con sus datos financieros.<sup>47</sup>

Se proponen dos columnas adicionales en la tabla *productores\_wallet* (ver Sección 2.1):

1. *conversation\_state* (VARCHAR): Almacena el estado actual, ej. *awaiting\_pin*, *awaiting\_tanda\_join\_amount*.
2. *state\_context* (JSON): Almacena el contexto de la solicitud, ej. *{"pending\_payment\_to": "+52...", "amount": 150}*.

### Flujo de Estado (Ejemplo de Pago P2P):

1. El productor envía (por voz): "Pagar 150 a Juan".
2. Equipo 1 (Interfaz) procesa la intención (*INTENT\_PAGAR*), extrae las entidades (monto: 150, receptor: +52...) y actualiza la BD: *UPDATE productores\_wallet SET conversation\_state = 'awaiting\_pin', state\_context = '{"to":"+52...", "amount":150}' WHERE telefono\_wa = ?*.
3. Equipo 1 responde (vía TTS) <sup>25</sup>: "Para confirmar el pago de \$150 a Juan, por favor ingresa tu PIN de 4 dígitos."<sup>1</sup>
4. El productor responde con el texto: "1234".
5. Equipo 1 (Interfaz) recibe "1234". Primero consulta el *conversation\_state*. Ve que es *awaiting\_pin*.
6. Lee el *state\_context* y el PIN ("1234"). Ahora tiene toda la información necesaria.
7. Llama al endpoint *POST /api/p2p-payment* (Team 2) con el *body* completo: *{"from\_wa": "...", "to\_wa": "...", "amount": 150, "pin": "1234"}*.
8. Equipo 2 (API) verifica el PIN <sup>41</sup> y procesa la transacción.
9. Equipo 1 limpia el estado en la BD: *UPDATE productores\_wallet SET conversation\_state = NULL, state\_context = NULL*.
10. Equipo 1 responde (vía TTS): "Pago exitoso".

## Mitigación del Riesgo de IA: Soporte de Náhuatl

Un pilar central de la propuesta de valor es el soporte para el idioma Náhuatl. Sin embargo, un análisis de la documentación de Azure AI Speech <sup>49</sup> indica que el Náhuatl no es un idioma soportado de forma nativa para Speech-to-Text. Intentar que funcione en el hackathon resultará en un fracaso.

Para mitigar este riesgo <sup>1</sup> y aun así demostrar la visión, el plan se divide en dos niveles:

1. **Para el Hackathon (MVP Funcional):** Se utilizará el modelo es-MX (Español, México).<sup>49</sup> La demostración *debe* realizarse en español. El mapeo de intención se basará en frases clave en español.<sup>19</sup>
2. **Para la Visión de Impacto (El Pitch):** La solución real para la inclusión <sup>1</sup> se logrará con **Custom Speech** de Azure.<sup>16</sup> Este servicio permite entrenar modelos de reconocimiento de voz personalizados subiendo datos de audio propios. El "siguiente paso" (post-hackathon) será trabajar con la comunidad para recolectar y etiquetar muestras de audio en Náhuatl y entrenar un modelo nah-MX personalizado.<sup>16</sup>

Esta estrategia permite al equipo tener un MVP 100% funcional <sup>1</sup> mientras demuestra un entendimiento maduro de la plataforma de Azure AI y la hoja de ruta hacia la "Inclusión Real".

## Sección 2: Contratos de Datos y Esquemas Centrales (El "Single Source of Truth")

Para que ambos equipos trabajen en paralelo sin bloqueos, deben acordar esta "fuente única de verdad". El Equipo 2 es el *dueño* de estos contratos; el Equipo 1 es el *consumidor*.

### 2.1. Tabla: Esquema del Ledger Virtual (Azure Database for MySQL)

El Equipo 2 desplegará un **Azure Database for MySQL Flexible Server** <sup>10</sup> y ejecutará el siguiente script SQL para crear el *ledger* virtual.<sup>1</sup>

Tabla 1: productores\_wallet

(Almacena el estado principal de cada productor, su saldo y el estado de su conversación)

Columna	Tipo	Notas
id	INT (PK, AI)	Identificador único.
telefono_wa	VARCHAR(20) (UNIQUE)	Teléfono del productor (ej. whatsapp:+52...). Clave principal de negocio.
wp_user_id	BIGINT (NULL)	ID opcional de WordPress/Dokan. <sup>1</sup>
saldo_mxn	DECIMAL(10, 2)	Saldo actual. Crítico para la

		precisión.
clabe_registrada	VARCHAR(18) (NULL)	CLABE para retiros. <sup>1</sup>
pin_hash	VARCHAR(60) (NULL)	Hash de Bcrypt del PIN de 4 dígitos. <sup>1</sup>
conversation_state	VARCHAR(50) (NULL)	Máquina de estados (ej. 'awaiting_pin', 'awaiting_tanda_amount').
state_context	JSON (NULL)	Contexto para el estado (ej. {"to":"+52...", "amount":150}).

Tabla 2: tandas

(Define cada grupo de ahorro o "Tanda") 1

Columna	Tipo	Notas
id	INT (PK, AI)	ID de la tanda.
nombre_grupo	VARCHAR(100)	Nombre de la tanda (ej. "Tanda Semanal").
monto_aporte	DECIMAL(10, 2)	Aporte por período (ej. \$100).
periodicidad	VARCHAR(20)	(ej. 'weekly', 'monthly').
num_miembros	INT	Número de miembros (ej. 10).
fecha_inicio	DATE	Fecha de inicio.

Tabla 3: tanda\_miembros

(Vincula a los productores con las tandas a las que pertenecen) 1

Columna	Tipo	Notas
id	INT (PK, AI)	
tanda_id	INT (FK -> tandas.id)	ID de la tanda.
telefono_wa	VARCHAR(20) (FK -> productores_wallet.telefono_wa)	ID del miembro.
num_sorteo	INT	Número de turno (ej. 1 a 10).

Tabla 4: tanda\_calendario

(Define el calendario de pagos para cada tanda) 1

Columna	Tipo	Notas
id	INT (PK, AI)	
tanda_id	INT (FK -> tandas.id)	ID de la tanda.
fecha_pago	DATE	Fecha en que se recibe el dinero.
telefono_wa_recibe	VARCHAR(20) (FK -> productores_wallet.telefono_wa)	Quién recibe el dinero en esta fecha.

## 2.2. Tabla: Contrato de la API de Wallet (REST/JSON)

Estos son los endpoints que el **Equipo 2** debe implementar en su Azure Function App, siguiendo las mejores prácticas de diseño de API REST.<sup>53</sup> Todas las respuestas deben usar JSON.<sup>53</sup>

### Endpoints para el Equipo 1 (Interfaz Conversacional):

Método	Endpoint	Parámetros/Cuerpo (Body)	Respuesta Exitosa (200)	Respuestas de Error
GET	/api/balance	Query Param: telefono_wa={telefono}	{"balance": 500.00}	404: {"error": "Usuario no encontrado"}
POST	/api/set-pin	Body: {"telefono_wa": "+52...", "new_pin": "1234"}	{"success": true, "message": "PIN configurado"}	400: {"error": "PIN inválido"}
POST	/api/p2p-payment	Body: {"from_wa": "+52...", "to_wa": "+52...", "amount": 150.00, "pin": "1234"}	{"success": true, "new_balance": 350.00, "recipient_name": "Juan"}	401: {"error": "PIN_INCORRECTO"} 400: {"error": "FONDOS_INSUFICIENTES"} 404: {"error": "RECEPTOR_NO_ENCONTRADO"}
POST	/api/initiate-withdrawal	Body: {"telefono_wa": "+52...", "amount": 500.00, "pin": "1234"}	{"success": true, "message": "Simulación de SPEI a CLABE [XXXX] exitosa."} <sup>1</sup>	401: {"error": "PIN_INCORRECTO"}
GET	/api/tandas	Query Param: telefono_wa={telefono}	{"tandas":} <sup>1</sup>	404: {"error": "Usuario no encontrado"}

### Endpoints para Webhooks Externos (WooCommerce, Simuladores):

Método	Endpoint	Body Esperado	Respuesta Exitosa (200)	Notas
POST	/api/credit-sale	Body: {"wp_user_id":	{"success": true}	Endpoint para el webhook de

		123, "amount": 500.00} (O telefono_wa)		WooCommerce. <sup>28</sup>
POST	/api/distribute-wm	Body: {"amount": 1000.00}	{"success": true, "distributed_to": 25, "amount_per_user": 40.00}	Endpoint simulador <sup>1</sup> para distribuir fondos de Rafiki.

## Sección 3: Parte 1: Plan de Tareas – Equipo de Interfaz Conversacional (WhatsApp y IA)

**Objetivo:** Construir el "cerebro" conversacional <sup>1</sup> que gestiona la interacción del usuario, se conecta a la IA y consume la API de Wallet.

### 3.1. Fase 1: Configuración del Canal de Comunicación (Twilio)

- **Tarea 1.1:** Crear una cuenta de prueba gratuita en Twilio.<sup>5</sup>
- **Tarea 1.2:** Activar el **Twilio Sandbox para WhatsApp** desde la consola de Twilio.<sup>4</sup> Esto es crucial ya que evita el proceso de aprobación de un número de negocio, lo cual es imposible en 5 horas.<sup>58</sup>
- **Tarea 1.3:** Conectar los teléfonos de prueba del equipo <sup>1</sup> al Sandbox. Esto se hace enviando el mensaje de WhatsApp "join " al número del sandbox de Twilio.<sup>4</sup>
- **Tarea 1.4:** Localizar el Account SID y Auth Token en la consola de Twilio.<sup>15</sup> Almacenarlos de forma segura como variables de entorno en la configuración de la Azure Function App.

### 3.2. Fase 2: Despliegue del Webhook de Lógica (Azure Functions)

- **Tarea 2.1:** En el portal de Azure, crear una **Azure Function App (Node.js)**.<sup>6</sup> Se recomienda el modelo de programación v4.<sup>9</sup>
- **Tarea 2.2:** Dentro de la Function App, crear una nueva función de tipo **Trigger HTTP** <sup>6</sup> y nombrarla /api/whatsapp-hook.<sup>1</sup>
- **Tarea 2.3:** Copiar la URL de la función desplegada. En la consola de Twilio, ir a la configuración del Sandbox de WhatsApp y pegar esta URL en el campo WHEN A MESSAGE COMES IN.<sup>5</sup> Asegurarse de que el método esté configurado como HTTP POST.<sup>6</sup>

- **Tarea 2.4:** Implementar la lógica inicial de *eco de texto* para probar el flujo completo: recibir un mensaje y responder con el mismo texto, usando la biblioteca twilio de Node.js.<sup>6</sup>

### 3.3. Fase 3: Integración de Azure AI Foundry (Voz e Intención)

- **Tarea 3.1:** En el portal de Azure, crear un recurso de **Azure AI Foundry**.<sup>7</sup> Este recurso actuará como el hub para todos los servicios de IA.<sup>62</sup>
- **Tarea 3.2:** Dentro de AI Foundry, crear un nuevo proyecto<sup>7</sup> y desplegar los modelos de IA necesarios<sup>8</sup>:
  - **Azure AI Speech (Speech-to-Text)**.<sup>16</sup>
  - **Azure AI Speech (Text-to-Speech)**.<sup>25</sup>
- **Tarea 3.3 (Mitigación de Riesgo Náhuatl):** Configurar el cliente del SDK de Speech-to-Text para usar el modelo es-MX (Español, México)<sup>49</sup> como se discutió en la Sección 1.3.
- **Tarea 3.4 (Flujo de Audio Entrante):**
  - En el código del webhook /api/whatsapp-hook, analizar el cuerpo de la solicitud de Twilio.
  - Detectar si el mensaje es de tipo audio<sup>1</sup> buscando el campo MediaUrlO en el *body*.
  - Si MediaUrlO existe, pasar esta URL pública al **SDK de Azure AI Speech** o al REST API<sup>17</sup> para iniciar la transcripción.
- **Tarea 3.5 (Mapeo de Intención):**
  - Recibir el string de texto transcrito (ej. "quiero pagarle cien pesos a juan").
  - Implementar un mapeador de intención simple<sup>19</sup> (un switch o una serie de if/else es suficiente para el MVP<sup>1</sup>) para mapear frases a comandos:
 

```

            ■ `if (text.includes("cuánto tengo") |
| text.includes("saldo")) { intent = 'SALDO'; }* if (text.includes("pagar") |
| text.includes("manda")) { intent = 'PAGAR'; }* if (text.includes("retirar")) { intent = 'RETIRAR';
}* if (text.includes("tandas")) { intent = 'TANDAS'; }`
          
```

### 3.4. Fase 4: Implementación del Flujo de Conversación (Estado y TTS)

- **Tarea 4.1 (Gestión de Estado):** Implementar la lógica de máquina de estados finitos<sup>45</sup> basada en la lectura/escritura de las columnas *conversation\_state* y *state\_context* en la BD MySQL (como se detalla en la Sección 1.3).
  - **Nota:** El Equipo 1 necesitará acceso de *lectura/escritura* a estas dos columnas en la tabla *productores\_wallet*. Deberán coordinar esta conexión con el Equipo 2.<sup>48</sup>
- **Tarea 4.2 (Implementar Flujo de Pago):**

- Si intent == 'PAGAR' y conversation\_state == NULL: Extraer el monto y el receptor. Llamar a la BD para SET state='awaiting\_pin', context={...}. Llamar a la Tarea 4.3 con el texto: "Para confirmar, ingresa tu PIN."<sup>1</sup>
- Si el msg es numérico y conversation\_state == 'awaiting\_pin': Leer el state\_context de la BD. Llamar al endpoint POST /api/p2p-payment (Team 2) con todos los datos (from, to, amount, pin). Limpiar el estado en la BD. Llamar a la Tarea 4.3 con el mensaje de éxito/error de la API.
- **Tarea 4.3 (Flujo de Audio Saliente):**
  - Crear una función helper sendAudioResponse(textResponse).
  - Esta función llamará a la API de **Azure AI Text-to-Speech**<sup>25</sup> con el textResponse.
  - Recibirá un archivo de audio.mp3<sup>1</sup> (o una URL al mismo).
  - Usará el cliente de Twilio (twilio.messages.create()) para enviar un nuevo mensaje de WhatsApp que contenga este audio, usando el parámetro mediaUrl.<sup>15</sup>

## Sección 4: Parte 2: Plan de Tareas – Equipo de API de Wallet (Ledger y Pagos)

**Objetivo:** Construir el "motor de confianza"<sup>1</sup> y el *ledger*, exponiéndolo como una API segura y transaccional.

### 4.1. Fase 1: Despliegue del Backend de Pagos (Interledger)

- **Tarea 4.1:** Investigar los docker-compose files disponibles en el repositorio de **Rafiki**.<sup>12</sup>
- **Tarea 4.2 (Decisión de Diseño):** Para el hackathon, evitar la compleja configuración de producción de Rafiki.<sup>65</sup> Utilizar el docker-compose del "**Local Playground**"<sup>66</sup> proporcionado por el equipo de Rafiki. Este playground está diseñado para un inicio rápido, incluye un Mock ASE (Account Servicing Entity) y está preconfigurado para el desarrollo.<sup>66</sup>
- **Tarea 4.3:** En el portal de Azure, crear un recurso de **Azure Container Instances (ACI)**.<sup>13</sup> ACI es la forma más rápida de ejecutar un contenedor Docker en Azure.<sup>13</sup>
- **Tarea 4.4:** Desplegar la configuración de docker-compose del Local Playground de Rafiki directamente en ACI (ACI tiene soporte nativo para archivos Compose).<sup>13</sup>
- **Tarea 4.5:** Configurar un (1) wallet maestro de Kostik<sup>1</sup> dentro de la instancia de Rafiki desplegada para simular la recepción de fondos de Web Monetization.

### 4.2. Fase 2: Configuración del Ledger Virtual (Base de Datos)

- **Tarea 2.1:** En el portal de Azure, crear un recurso de **Azure Database for MySQL - Flexible Server**.<sup>10</sup>
- **Tarea 2.2:** Configurar las opciones de "Redes" del servidor MySQL <sup>68</sup> para permitir conexiones entrantes desde la Azure Function App (Team 2).<sup>38</sup>
- **Tarea 2.3:** Usar un cliente (como MySQL Workbench <sup>68</sup> o DBeaver) para conectarse a la base de datos y ejecutar el script SQL de la Sección 2.1 para crear las 4 tablas.<sup>1</sup>

### 4.3. Fase 3: Implementación de la Lógica de Negocio (Azure Functions)

- **Tarea 3.1:** En el portal de Azure, crear una *segunda Azure Function App (Node.js)* <sup>9</sup>, separada de la del Equipo 1.
- **Tarea 3.2:** Configurar la conexión segura a Azure MySQL.
  - Instalar el conector de Node.js para MySQL (ej. mysql2).<sup>23</sup>
  - Habilitar la **Identidad Administrada** en la Function App.<sup>35</sup>
  - Otorgar a esta identidad permisos de conexión y DML en el servidor MySQL.<sup>37</sup>
- **Tarea 3.3 (Endpoint de Seguridad):** Implementar POST /api/set-pin.
  - Instalar bcrypt: npm install bcrypt.<sup>34</sup>
  - Generar un salt (bcrypt.genSalt) y hashear el PIN (bcrypt.hash).<sup>39</sup>
  - Almacenar el pin\_hash resultante en la tabla productores\_wallet.<sup>1</sup>
- **Tarea 3.4 (Endpoint P2P Transaccional):** Implementar POST /api/p2p-payment.
  - Esta operación *debe* ser atómica para evitar la pérdida de fondos.<sup>1</sup> Se debe usar una **transacción de base de datos**.<sup>33</sup>
  - **Lógica:**
    1. Validar el PIN: Obtener el pin\_hash del usuario from\_wa.
    2. Ejecutar const isValid = await bcrypt.compare(pin, pin\_hash);.<sup>41</sup> Si !isValid, retornar 401.
    3. Obtener una conexión a la BD e iniciar una transacción: await connection.beginTransaction();.<sup>33</sup>
    4. **Try/Catch Block:**
      - UPDATE productores\_wallet SET saldo\_mxn = saldo\_mxn -? WHERE telefono\_wa =? AND saldo\_mxn >=?; (El AND saldo\_mxn >=? previene saldos negativos).
      - Verificar si la fila fue afectada. Si no, lanzar un error de "Fondos Insuficientes".
      - UPDATE productores\_wallet SET saldo\_mxn = saldo\_mxn +? WHERE telefono\_wa =?;
      - await connection.commit();.<sup>33</sup>
    5. **Catch (error):**

- `await connection.rollback();`<sup>33</sup>
  - Retornar una respuesta de error 400.
6. Retornar la respuesta de éxito 200.

#### 4.4. Fase 4: Implementación del Motor de Tandas (Lógica Asíncrona)

- **Tarea 4.1:** Crear una **Azure Function (Timer Trigger)**<sup>30</sup> dentro de la Function App del Equipo 2.
- **Tarea 4.2:** Configurar el *schedule*<sup>1</sup> usando una expresión CRON.<sup>31</sup> (ej. `0 0 8 * * 5` para "Viernes a las 8:00 AM UTC"<sup>32</sup>).
- **Tarea 4.3:** Implementar la lógica del motor de la Tanda.<sup>1</sup>
  - Conectarse a MySQL.<sup>23</sup>
  - Buscar en `tanda_calendario` a quién le toca recibir dinero *hoy*.
  - Si se encuentra un beneficiario:
    1. Obtener todos los miembros de esa `tanda_id` de `tanda_miembros`.
    2. Ejecutar la lógica de transacción atómica<sup>33</sup> para debitar (ej. \$100) a todos los miembros y acreditar el total (ej. \$100 \* `num_miembros`) al beneficiario.<sup>1</sup>
    3. Usar el cliente de Twilio (configurado también en esta Function App) para enviar notificaciones de confirmación por WhatsApp a todos los miembros.<sup>1</sup>

#### 4.5. Fase 5: Integración de Pagos Externos (Webhooks)

- **Tarea 5.1:** Implementar el endpoint `POST /api/credit-sale`.<sup>1</sup> Este endpoint debe ser simple: recibe una solicitud, valida una clave secreta (para seguridad básica del webhook) y ejecuta un `UPDATE` en `productores_wallet` para acreditar el `amount`.
- **Tarea 5.2:** En el panel de control de WordPress/WooCommerce<sup>1</sup>, navegar a `WooCommerce > Settings > Advanced > Webhooks`.<sup>28</sup>
- **Tarea 5.3:** Crear un nuevo Webhook.<sup>28</sup>
  - **Topic:** Seleccionar `Order Updated`<sup>27</sup> o usar la acción `woocommerce_order_status_completed`.<sup>1</sup>
  - **Delivery URL:** Pegar la URL de la Azure Function `/api/credit-sale` desplegada.
- **Tarea 5.4:** Implementar el endpoint simulador `/api/distribute-wm`<sup>1</sup> que simplemente distribuye un monto fijo entre todos los usuarios en la tabla `productores_wallet`.

## Sección 5: Plan Consolidado y Hoja de Ruta del Hackathon

## 5.1. Dependencias Críticas y Estrategia de Mocks (Simuladores)

El documento del hackathon advierte: "El error en un hackathon es querer construir todo".<sup>1</sup> El mayor riesgo en este proyecto de dos equipos es el bloqueo de dependencias.

- **Dependencia Crítica:** El Equipo 1 (Interfaz) está 100% bloqueado por el Equipo 2 (API). No pueden probar *nada* hasta que los endpoints estén vivos.
- **Estrategia de Desbloqueo (Mocks):** Para habilitar el desarrollo en paralelo, el principio de "simular las partes lentas"<sup>1</sup> se aplica a la propia API.
  - **Hora 0:00 - 0:30:** Ambos equipos se reúnen y acuerdan *exactamente* el **Contrato de API** (Sección 2.2). Este es el paso más importante.
  - **Hora 0:30 - 1:00:** La *primera* tarea del Equipo 2 es desplegar una Azure Function App<sup>6</sup> con todos los endpoints de la Sección 2.2 implementados como *simuladores (mocks)*.
    - Por ejemplo, el mock de POST /api/p2p-payment<sup>1</sup> simplemente recibirá el JSON, lo imprimirá en el log (context.log("Simulando pago...")) y retornará una respuesta exitosa codificada: {"success": true, "new\_balance": 123.45}.
  - **Resultado (Hora 1:00):** El Equipo 1 está completamente desbloqueado y puede construir toda la interfaz de IA y conversación<sup>1</sup> contra un backend funcional (aunque simulado).
  - **Desarrollo en Paralelo:** El Equipo 2 pasa el resto del hackathon reemplazando los *mocks* uno por uno con la lógica de BD real<sup>23</sup>, la lógica de Rafiki<sup>66</sup> y el hashing de PIN.<sup>34</sup>

## 5.2. Hoja de Ruta Priorizada (Timeline del Hackathon de 5 Horas)

- **Hora 0:00 - 1:00 (Setup y Contratos)**
  - **Ambos Equipos:** Desplegar recursos base en Azure (2 Function Apps<sup>6</sup>, 1 MySQL<sup>10</sup>, 1 ACI<sup>13</sup>).
  - **Equipo 2:** Ejecutar script SQL (Sección 2.1) en MySQL.<sup>68</sup> Desplegar **Mocks de API** (Sección 5.1).
  - **Equipo 1:** Configurar Twilio Sandbox.<sup>4</sup>
- **Hora 1:00 - 2:00 (Flujo de Texto y Conexión Crítica)**
  - **Equipo 1:** Conectar Twilio a la Function (/api/whatsapp-hook).<sup>6</sup> Implementar el flujo de *texto* "SALDO"<sup>1</sup> llamando al endpoint *mock* de GET /api/balance.
  - **Equipo 2:** Reemplazar el mock de GET /api/balance con la conexión real a MySQL.<sup>23</sup> Comenzar la implementación del P2P transaccional.<sup>33</sup>
- **Hora 2:00 - 4:00 (Desarrollo en Paralelo de Funcionalidad Central)**
  - **Equipo 1:** Implementar "La Magia".<sup>1</sup> Integrar Azure AI Foundry (STT/TTS).<sup>7</sup>

Implementar la lógica de estado de conversación (Sección 1.3) para el flujo de PIN.

- *Equipo 2*: Implementar la lógica de seguridad del PIN (bcrypt).<sup>34</sup> Finalizar la lógica de transacción P2P.<sup>33</sup> Desplegar Rafiki en ACI.<sup>13</sup>
- **Hora 4:00 - 5:00 (Funciones "Wow" y Pulido de Demo)**
  - *Equipo 1*: Refinar el mapeo de intención <sup>19</sup> y las respuestas de audio.<sup>26</sup> Probar el flujo de PIN E2E con la API real del Equipo 2.
  - *Equipo 2*: Implementar el **Timer Trigger de Tandas** <sup>30</sup> y el webhook de **WooCommerce**.<sup>27</sup>
  - *Ambos Equipos*: Prueba de integración final. Preparar la demostración.

### 5.3. Criterios de Éxito del MVP y Guion del Pitch

El éxito se define como "demostrar lo máximo" con "lo mínimo".<sup>1</sup> El MVP es exitoso si el equipo puede demostrar el siguiente guion:

1. **Contexto (Problema)**: Mostrar una diapositiva: Productores rurales, barreras de idioma, desconfianza financiera.<sup>1</sup>
2. **Flujo 1 (Ventas)**: Mostrar la pantalla de una orden de WooCommerce.<sup>28</sup> Marcarla como "completada".
3. **Flujo 2 (Notificación y Saldo)**: El productor (demo) recibe un WhatsApp.<sup>1</sup> Envía un *mensaje de voz* en español: "cuánto tengo".
4. **La Magia (IA)**: El bot <sup>1</sup> responde con un *mensaje de voz*: "Tienes 500 pesos". (Esto demuestra Twilio -> Azure Function -> AI Foundry STT -> API de Wallet -> AI Foundry TTS -> Twilio).
5. **Flujo 3 (P2P y Seguridad)**: El productor envía voz: "Pagar 150 pesos a [nombre del compañero]". El bot responde con voz: "Para confirmar, por favor ingresa tu PIN de 4 dígitos".<sup>1</sup> El productor escribe "1234". El bot responde con voz: "Pago exitoso". (Demuestra la gestión de estado, la seguridad del PIN y la transacción P2P).
6. **Flujo 4 (Economía Circular)**: Mostrar el código de la Azure Function (Timer Trigger) <sup>30</sup> y explicar: "Este motor de Tandas se ejecuta automáticamente cada viernes para mover el dinero *dentro* de la comunidad, creando una economía circular".<sup>1</sup>

El pitch final debe encapsular la visión del proyecto <sup>1</sup>: "No construimos un wallet. Construimos una **plataforma de confianza** que: **Recibe Dinero** (Ventas y WM), **Mueve Dinero** (P2P), **Protege Dinero** (PIN) y **Hace Crecer Dinero** (Tandas). Todo esto, en Azure, y accesible en el idioma del productor gracias a Azure AI".<sup>1</sup>

#### Fuentes citadas

1. Hackathon Interledger
2. AI Architecture Design - Azure Architecture Center | Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/architecture/ai-ml/>

3. Browse Azure Architectures - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/architecture/browse/>
4. Trying Out WhatsApp with Conversations - Twilio, acceso: noviembre 8, 2025, <https://www.twilio.com/docs/conversations/use-twilio-sandbox-for-whatsapp>
5. Test WhatsApp messaging with the Sandbox | Twilio, acceso: noviembre 8, 2025, <https://www.twilio.com/docs/whatsapp/sandbox>
6. Serverless Webhooks with Azure Functions and Node.js - Twilio, acceso: noviembre 8, 2025, <https://www.twilio.com/docs/usage/tutorials/serverless-webhooks-azure-functions-and-node-js>
7. Quickstart - Create a new Azure AI Foundry Agent Service project - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/ai-foundry/agents/quickstart>
8. Azure AI Foundry SDK Quickstart - Build Your First AI App - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/ai-foundry/quickstarts/get-started-code>
9. Node.js developer reference for Azure Functions - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/azure-functions/functions-reference-node>
10. Quickstart: Create a Flexible Server Using the Azure portal - Azure Database for MySQL, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/mysql/flexible-server/quickstart-create-server-portal>
11. Introduction to Azure Database for MySQL - Training - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/training/modules/intro-to-azure-database-for-mysql/>
12. Interledger/rafiqi: An open-source, comprehensive Interledger service for wallet providers, enabling them to provide Interledger functionality to their users. - GitHub, acceso: noviembre 8, 2025, <https://github.com/interledger/rafiqi>
13. Quickstart: Deploy a container instance in Azure using the Azure portal - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/container-instances/container-instances-quickstart-portal>
14. Tutorial: Deploy a container application to Azure Container Instances - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/container-instances/container-instances-tutorial-deploy-app>
15. Send and Receive Media Messages with WhatsApp in Node.js | Twilio, acceso: noviembre 8, 2025, <https://www.twilio.com/docs/whatsapp/tutorial/send-and-receive-media-messages-whatsapp-nodejs>
16. Speech to text overview - Speech service - Azure AI services - Microsoft Learn, acceso: noviembre 8, 2025,

- <https://learn.microsoft.com/en-us/azure/ai-services/speech-service/speech-to-text>
17. Speech to text REST API - Speech service - Azure AI services - Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/ai-services/speech-service/rest-speech-to-text>
  18. azure speech to text with audio url - javascript - Stack Overflow, acceso: noviembre 8, 2025,  
<https://stackoverflow.com/questions/73969528/azure-speech-to-text-with-audio-url>
  19. Guide - Building an intent classification pipeline - Langfuse, acceso: noviembre 8, 2025,  
[https://langfuse.com/guides/cookbook/example\\_intent\\_classification\\_pipeline](https://langfuse.com/guides/cookbook/example_intent_classification_pipeline)
  20. Advanced Node.js NLP Techniques - Medium, acceso: noviembre 8, 2025,  
<https://medium.com/@vloban/nlp-advanced-node-js-techniques-70a2e85098c2>
  21. Work with access keys in Azure Functions | Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/azure-functions/function-keys-how-to>
  22. Securing Azure Functions | Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/azure-functions/security-concepts>
  23. Quickstart: Connect Using Node.js - Azure Database for MySQL | Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/mysql/flexible-server/connect-nodejs>
  24. Azure Database for MySQL Input Binding for Functions | Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-azure-mysql-input>
  25. Text to speech overview - Speech service - Azure AI services | Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/ai-services/speech-service/text-to-speech>
  26. Sending Audio Messages in Twilio - Stack Overflow, acceso: noviembre 8, 2025,  
<https://stackoverflow.com/questions/78374759/sending-audio-messages-in-twilio>
  27. In Woocommerce API, which hook for a specific order status for a webhook?, acceso: noviembre 8, 2025,  
<https://stackoverflow.com/questions/61955721/in-woocommerce-api-which-hook-for-a-specific-order-status-for-a-webhook>
  28. Webhooks Documentation - WooCommerce, acceso: noviembre 8, 2025,  
<https://woocommerce.com/document/webhooks/>
  29. How to export Woocommerce upcoming orders to Azure database with API - Reddit, acceso: noviembre 8, 2025,  
[https://www.reddit.com/r/AZURE/comments/1527ebu/how\\_to\\_export\\_woocommerce\\_upcoming\\_orders\\_to/](https://www.reddit.com/r/AZURE/comments/1527ebu/how_to_export_woocommerce_upcoming_orders_to/)
  30. Timer trigger for Azure Functions | Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-time>

r

31. TimerTrigger Interface | Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/java/api/com.microsoft.azure.functions.annotation.timertrigger?view=azure-java-stable>
32. Azure Functions - Time Trigger (CRON) Cheat Sheet - arminreiter.com, acceso: noviembre 8, 2025, <https://arminreiter.com/2017/02/azure-functions-time-trigger-cron-cheat-sheet/>
33. Node.js MySQL Transaction: a step-by-step tutorial with a real-life example, acceso: noviembre 8, 2025, <https://knowledgeacademy.io/node-js-mysql-transaction/>
34. Password hashing in Node.js with bcrypt - LogRocket Blog, acceso: noviembre 8, 2025, <https://blog.logrocket.com/password-hashing-node-js-bcrypt/>
35. Managed identities for Azure resources - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/entra/identity/managed-identities-azure-resources/overview>
36. Azure Functions with Managed Identity Storage Access - Transparency, acceso: noviembre 8, 2025, <https://www.transparency.com/app-innovation/azure-functions-with-managed-identity-storage-access/>
37. Tutorial: Managed Identity to Invoke Azure Functions - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/spring-apps/basic-standard/tutorial-managed-identities-functions>
38. How to connect azure mysql server with nodeJS - Stack Overflow, acceso: noviembre 8, 2025, <https://stackoverflow.com/questions/73960040/how-to-connect-azure-mysql-server-with-nodejs>
39. Hashing Passwords in Node and Express using bcrypt - YouTube, acceso: noviembre 8, 2025, [https://www.youtube.com/watch?v=AzA\\_LTDoFqY](https://www.youtube.com/watch?v=AzA_LTDoFqY)
40. How to Use Mysql Password Encryption with NodeJS, Express and Bcrypt | by crashdaddy, acceso: noviembre 8, 2025, <https://kennethscoggins.medium.com/how-to-use-mysql-password-encryption-with-nodejs-express-and-bcrypt-ad9ede661109>
41. How to fetch hashed password from DB in Node.js so that bcrypt can compare it, acceso: noviembre 8, 2025, <https://stackoverflow.com/questions/62452122/how-to-fetch-hashed-password-from-db-in-node-js-so-that-bcrypt-can-compare-it>
42. Sending and receiving session state info (user data) while exchanging WhatsApp messages, acceso: noviembre 8, 2025, <https://stackoverflow.com/questions/60374870/sending-and-receiving-session-state-info-user-data-while-exchanging-whatsapp-m>
43. Building a persistent conversational AI chatbot with Temporal, acceso: noviembre 8, 2025, <https://temporal.io/blog/building-a-persistent-conversational-ai-chatbot-with-te>

[mporal](#)

44. Save user and conversation data - Bot Service | Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/bot-service/bot-builder-howto-v4-state?view=azure-bot-service-4.0>
45. Managing state in Bot Framework SDK - Bot Service | Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/bot-service/bot-builder-concept-state?view=azure-bot-service-4.0>
46. Durable Functions Overview - Azure - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview>
47. Conversation state in Microsoft bot framework - Stack Overflow, acceso: noviembre 8, 2025, <https://stackoverflow.com/questions/74992241/conversation-state-in-microsoft-bot-framework>
48. Azure Database for MySQL Trigger Binding for Functions | Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-azure-mysql-trigger>
49. Language support - Speech service - Azure AI services - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/ai-services/speech-service/language-support>
50. Voice live API language support - Azure AI services - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/ai-services/speech-service/voice-live-language-support>
51. Language support - Azure AI services - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/ai-services/language-support>
52. Azure AI Speech | Microsoft Azure, acceso: noviembre 8, 2025, <https://azure.microsoft.com/en-us/products/ai-services/ai-speech>
53. Best practices for REST API design - The Stack Overflow Blog, acceso: noviembre 8, 2025, <https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/>
54. Best Practices in API Design - Swagger, acceso: noviembre 8, 2025, <https://swagger.io/resources/articles/best-practices-in-api-design/>
55. Web API Design Best Practices - Azure Architecture Center - Microsoft Learn, acceso: noviembre 8, 2025, <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>
56. JSON:API — A specification for building APIs in JSON, acceso: noviembre 8, 2025, <https://jsonapi.org/>
57. Quickstart: Send and receive WhatsApp messages - Twilio, acceso: noviembre 8, 2025, <https://www.twilio.com/docs/whatsapp/quickstart>

58. WhatsApp Business Platform with Twilio, acceso: noviembre 8, 2025,  
<https://www.twilio.com/docs/whatsapp>
59. Configure a WhatsApp channel through Twilio | Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/dynamics365/customer-service/administer/configure-whatsapp-channel>
60. Respond to Twilio Webhooks using Azure Functions, acceso: noviembre 8, 2025,  
<https://www.twilio.com/en-us/blog/respond-to-twilio-webhooks-using-azure-functions>
61. Create an AI Foundry resource - Azure AI services - Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/ai-services/multi-service-resource>
62. Azure AI Foundry architecture - Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/ai-foundry/concepts/architecture>
63. Using Azure Queues as a State Machine - Stack Overflow, acceso: noviembre 8, 2025,  
<https://stackoverflow.com/questions/16777559/using-azure-queues-as-a-state-machine>
64. Text to speech REST API - Azure - Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/ai-services/speech-service/rest-text-to-speech>
65. Docker Compose - Rafiki, acceso: noviembre 8, 2025,  
<https://rafiki.dev/integration/deployment/docker-compose/>
66. Overview | Rafiki, acceso: noviembre 8, 2025,  
<https://rafiki.dev/integration/playground/overview/>
67. Quickstart - Deploy Docker container to container instance - Azure CLI - Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/container-instances/container-instances-quickstart>
68. Quickstart: Connect MySQL Workbench - Azure Database for MySQL | Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/mysql/flexible-server/connect-workbench>
69. Demo: Getting Started [4 of 16] | Azure Database for MySQL - Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/shows/azure-database-for-mysql-beginners-series/demo-getting-started>
70. Node.js mysql transaction - Stack Overflow, acceso: noviembre 8, 2025,  
<https://stackoverflow.com/questions/15760067/node-js-mysql-transaction>
71. Create a function in the Azure portal that runs on a schedule - Microsoft Learn, acceso: noviembre 8, 2025,  
<https://learn.microsoft.com/en-us/azure/azure-functions/functions-create-scheduled-function>
72. Azure Function doesn't follow CRON timer schedule - Stack Overflow, acceso: noviembre 8, 2025,

<https://stackoverflow.com/questions/76730344/azure-function-doesnt-follow-cron-timer-schedule>

API KEy AI Foundry

URL

<https://interledger-r.cognitiveservices.azure.com/openai/deployments/gpt-4.1/chat/completions?api-version=2025-01-01-preview>

API KEY

49fr8ONzRPDzNf9cI7qH1uqBPjdgbFBSkyowefjnkP8wQfCqjFHXJQQJ99BKACHYHv6XJ3w3AAAAACOGa1Dn