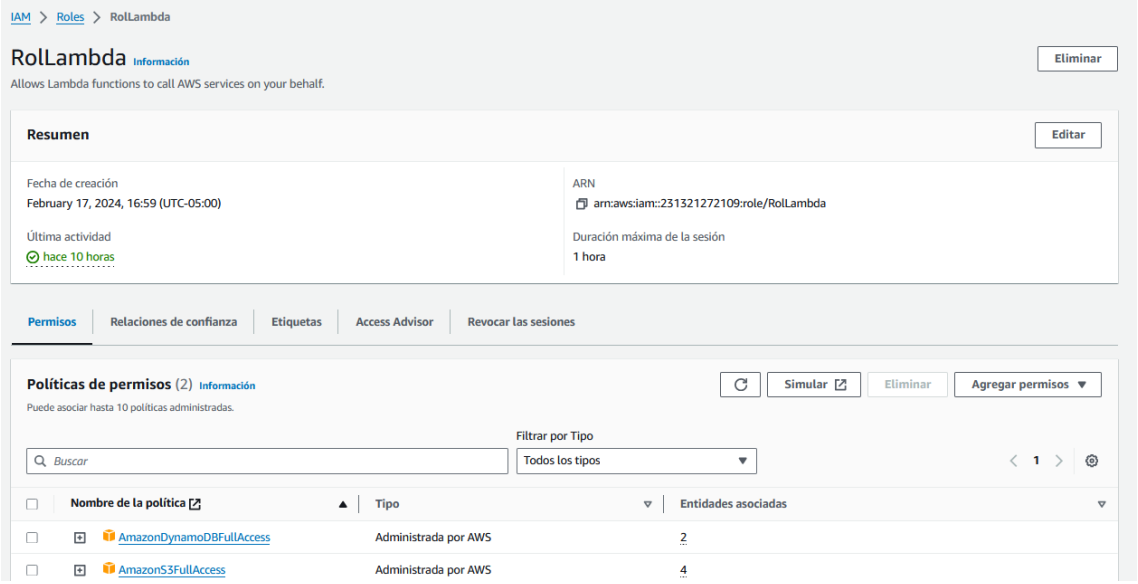
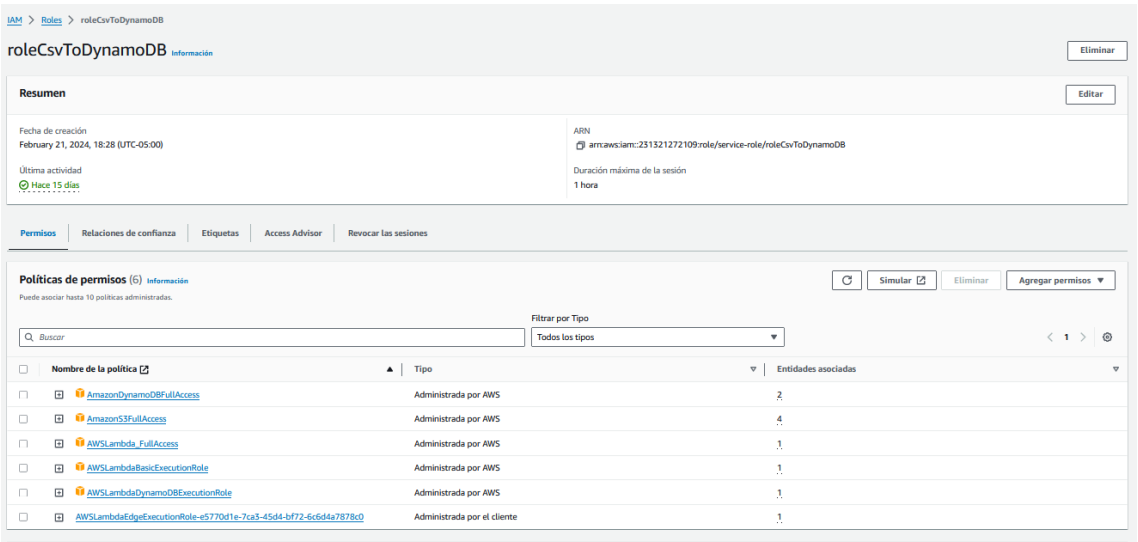
CREACION DE COMPONENTES AWS PARA LA ARQUITECTURA SPEECH ANALYZER

Creación de Roles:

Se requiere crear los roles en AWS de acuerdo como se muestran a continuación:

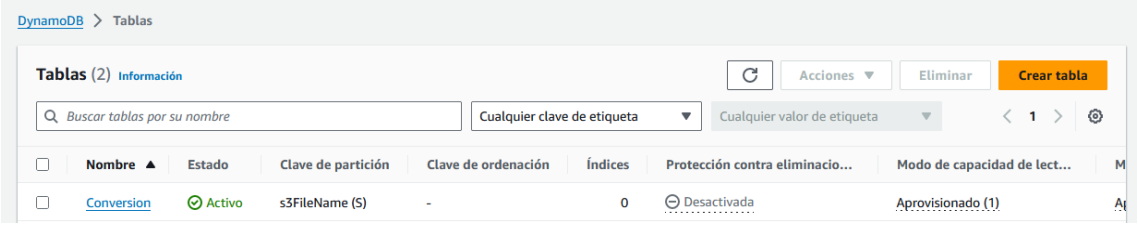


Además, se requiere establecer las siguientes políticas de permisos:



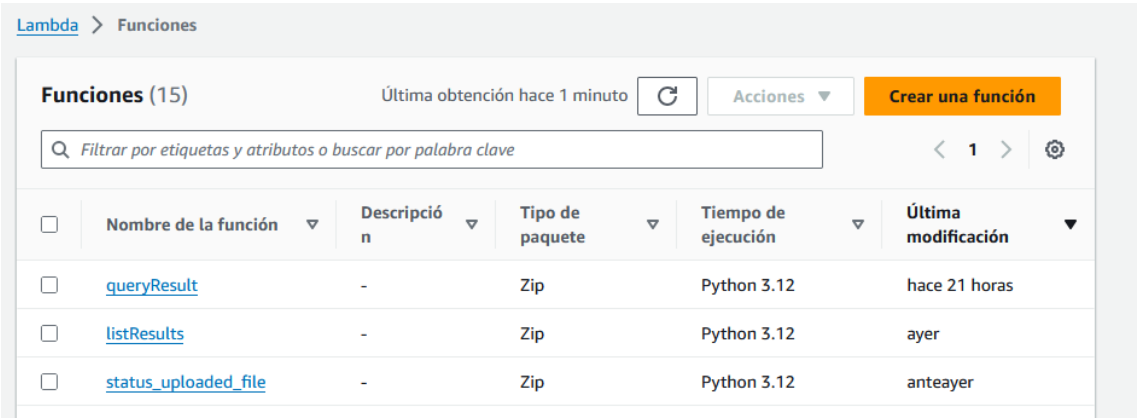
Creación de una tabla de datos en DynamoDB:

En este caso se crea una Tabla llamada Conversion, la cual almacenará los resultados de los análisis de audio.



Creación de Lambda Functions

La arquitectura desplegada requiere las siguientes funciones Lambda que se encargan de buscar un resultado, mostrar el listado de resultados almacenados y notificar la subida de un archivo nuevo.



status_uploaded_file: Esta función se encarga de guardar o modificar un registro en la Tabla "Conversion" de DynamoDB. A continuación, se presenta el código fuente de la función.

```
import json
import boto3


def lambda_handler(event, context):
  """Función Lambda para guardar/modificar un registro en la tabla Conversion de DynamoDB."""


  # Get data from the event, ensuring s3FileName is present
  try:
      fileName = event['s3FileName']
  except KeyError:
```

```
      return {
          'statusCode': 400,
          'body': json.dumps('Error: Missing required field
"s3FileName"')
      }

  # Extract optional fields with default values
  transcriptionResult = event.get('transcriptionResult', None)
  analysisResult = event.get('analysisResult', None)
  status_storage = event.get('status_storage', None)
  status_transcription = event.get('status_transcription', None)
  status_analysis = event.get('status_analysis', None)

  # Connect to DynamoDB
  dynamodb = boto3.resource('dynamodb', region_name='sa-east-1')
  table = dynamodb.Table('Conversion')

  # Update logic with checks for empty optional fields
  try:
      # Get the existing item (if any)
      response = table.get_item(Key={'s3FileName': fileName})
      item = response.get('Item')
      # Update only non-empty fields
      updates = {}
      if transcriptionResult is not None:
          updates['transcriptionResult']=transcriptionResult
      if analysisResult is not None:
          updates['analysisResult']=analysisResult
      if status_storage is not None:
          updates['status_storage']=status_storage
      if status_transcription is not None:
          updates['status_transcription']=status_transcription
      if status_analysis is not None:
          updates['status_analysis']=status_analysis
```

```
      if item:
          return {'statusCode': 200, 'body': "Name exists already "}
      else:
          new_item = {
              's3FileName': fileName,
              'transcriptionResult': transcriptionResult,
              'analysisResult': analysisResult,
              'status_storage': status_storage,
              'status_transcription': status_transcription,
              'status_analysis': status_analysis
          }
          table.put_item(Item=new_item)
          return {'statusCode': 200, 'body': json.dumps('Record
correctly saved')}


  except Exception as e:
     print(f"Error al procesar la solicitud: {e}")
     return {'statusCode': 500, 'body': json.dumps(f"Error:
{str(e)}")}
```

listResults: Esta función recupera los registros de la Tabla "Conversion" de DynamoDB. A continuación, se presenta el código fuente de la función.

```
import boto3
def lambda_handler(event, context):
  # Create a DynamoDB client
  dynamodb = boto3.resource('dynamodb')
  table = dynamodb.Table('Conversion')


  # Scan the entire table
  response = table.scan()
  items = response['Items']


  # Check for additional pages of results
  while 'LastEvaluatedKey' in response:
      response =
table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
```

```
        items.extend(response['Items'])

    # Process or return the list of items (modify as needed)
    for item in items:
        print(item)  # Print each item

    return {
        'statusCode': 200,
        'body': str(items)
    }
```

queryResult: se encarga de obtener todos los detalles de un resultado específico. Adicionalmente, agrega una URL temporal de S3 para descargar el archivo de audio del análisis. Requiere el nombre del archivo cuyo análisis se quiere recuperar de la base de datos.

```
import json
import boto3


def generate_presigned_url(bucket_name, file_name, expiration_time):
    """
    Genera un enlace firmado temporal para acceder al objeto en el
bucket de S3.

    Args:
    - bucket_name: El nombre del bucket de S3.
    - file_name: El nombre del objeto en el bucket de S3.
    - expiration_time: El tiempo de expiración del enlace firmado en
segundos.

    Returns:
    - El enlace firmado temporal.
    """
    # Crea un cliente de S3
    s3 = boto3.client('s3')

    # Genera el enlace firmado temporal
    presigned_url = s3.generate_presigned_url(
```

```
        'get_object',
        Params={'Bucket': bucket_name, 'Key': file_name},
        ExpiresIn=expiration_time
    )

    return presigned_url


def lambda_handler(event, context):
  # Get the s3FileName from the event or function input (modify as
needed)
  try:
      fileName = event['s3FileName']
  except KeyError:
      return {
          'statusCode': 400,
          'body': json.dumps('Error: Missing required field
"s3FileName"')
      }
  s3_file_name = fileName
  bucket_name = event.get('bucket_name')
  bucket_name = "finalmfsxes"


  # Nombre del archivo en el bucket de S3
  file_name = s3_file_name
  # Tiempo de expiración del enlace firmado en segundos
  expiration_time = 300  # 5 minutos


  # Si el nombre del bucket o el nombre del archivo no se encuentran
en el evento, devuelve un error
  if not bucket_name:
      return {
          'statusCode': 400,
          'body': 'El nombre del bucket no se encontro en el evento.'
      }



  # Create a DynamoDB client
```

```
  dynamodb = boto3.resource('dynamodb')
  table = dynamodb.Table('Conversion')

  # Get item using s3FileName as the key
  response = table.get_item(Key={'s3FileName': s3_file_name})

  # Check if item exists
  if 'Item' in response:
      item = response['Item']
      try:
          # Genera el enlace firmado temporal
          presigned_url = generate_presigned_url(bucket_name,
s3_file_name, expiration_time)
          item['url']=presigned_url
      except Exception as e:
          # Manejo de errores
          item['url']=""

      print(item)  # Print the item
      return {
          'statusCode': 200,
          'body': str(item),

      }
  else:
      return {
          'statusCode': 404,
          'body': f'Item with s3FileName: {s3_file_name} not found'
      }
```