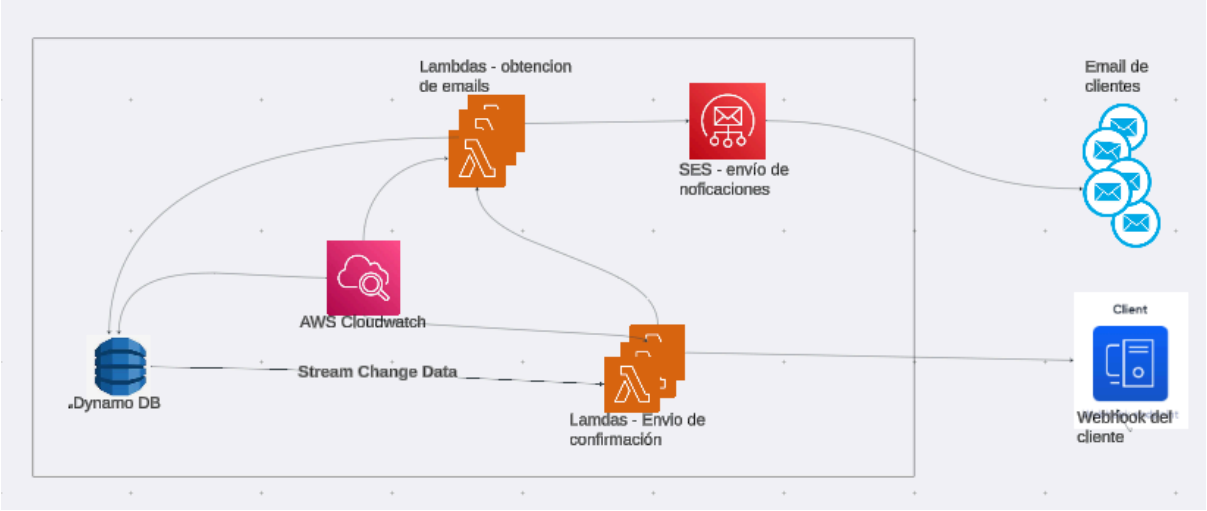
Propuesta 1

Para la primera solución se sugiere el uso de una base de datos de AWS DynamoDB para almacenar los datos de las transacciones pendientes por confirmar. Para el envío de la confirmación se utilizará el servicio de AWS Lambda. El autoescalado de las lambdas permitirá que pueda responder a aumentos en el número de eventos (transacciones por confirmar).
Las lambdas serán activadas por medio de streams de DynamoDB. Con esto cada vez que se inserten datos (transacciones por confirmar) en esta base de datos, se ejecutarán las lambdas de envío de confirmación.
Adicionalmente se propone el uso del servicio de AWS SES para enviar a los clientes notificaciones vía email con la confirmación de la transacción.



La plantilla requerida para el despliegue de estos recursos usando AWS Cloudformation se entrega junto con este documento con el nombre de IaCV1.yml.
El contenido de la misma se muestra a continuación:

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Resources:
 LambdaEmail:
   Type: AWS::Lambda::Function
   Properties:
     FunctionName: LambdaEmail
     Handler: lambda_email.lambda_handler  # Asumiendo que el archivo es lambda_email.py
     Role:
       Fn::GetAtt: [LambdaExecutionRole, Arn]
     Code:
       S3Bucket: my-lambda-code-bucket
       S3Key: lambda-email.zip
     Runtime: python3.8  # Usando Python 3.8
     MemorySize: 128
     Timeout: 60

 LambdaConfirmaciones:
   Type: AWS::Lambda::Function
   Properties:
     FunctionName: LambdaConfirmaciones
```

```
    Handler: lambda_confirmaciones.lambda_handler  # Asumiendo que el archivo es
lambda_confirmaciones.py
    Role:
      Fn::GetAtt: [LambdaExecutionRole, Arn]
    Code:
      S3Bucket: my-lambda-code-bucket
      S3Key: lambda-confirmaciones.zip
    Runtime: python3.8  # Usando Python 3.8
    MemorySize: 128
    Timeout: 60

 # Tabla DynamoDB con Streams
DynamoDBTable:
  Type: "AWS::DynamoDB::Table"
  Properties:
    TableName: "MyDynamoDBTable"
    AttributeDefinitions:
      - AttributeName: "ID"
        AttributeType: "S"
      - AttributeName: "transaccion"
        AttributeType: "S"
      - AttributeName: "email"
        AttributeType: "S"
    KeySchema:
      - AttributeName: "ID"
        KeyType: "HASH"
    BillingMode: PAY_PER_REQUEST
    StreamSpecification:
      StreamViewType: NEW_AND_OLD_IMAGES


SESConfigurationSet:
  Type: AWS::SES::ConfigurationSet
  Properties:
    Name: MySESConfigurationSet

LambdaExecutionRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Action: sts:AssumeRole
          Principal:
            Service: lambda.amazonaws.com
    Policies:
      - PolicyName: LambdaBasicExecutionPolicy
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
            - Effect: Allow
              Action:
                - logs:CreateLogGroup
                - logs:CreateLogStream
                - logs:PutLogEvents
              Resource: arn:aws:logs:*:*:*
            - Effect: Allow
              Action:
                - dynamodb:PutItem
```

```
                - dynamodb:GetItem
                - dynamodb:UpdateItem
                - dynamodb:Query
                - dynamodb:Scan
              Resource: !GetAtt DynamoDBTable.Arn
            - Effect: Allow
              Action:
                - ses:SendEmail
                - ses:SendRawEmail
              Resource: "*"

Outputs:
 LambdaEmailName:
   Value: !Ref LambdaEmail
   Description: "Lambda Email Function Name"
 LambdaConfirmacionesName:
   Value: !Ref LambdaConfirmaciones
   Description: "Lambda Confirmaciones Function Name"
 DynamoDBTableName:
   Value: !Ref DynamoDBTable
   Description: "DynamoDB Table Name"
 SESConfigurationSetName:
   Value: !Ref SESConfigurationSet
   Description: "SES Configuration Set Name"
```

Para la automatización del despliegue se sugiere el uso de un pipeline de Azure DevOps. Dicho pipeline está descrito en el archivo azure-pipelineV1.yml entregado con este documento.

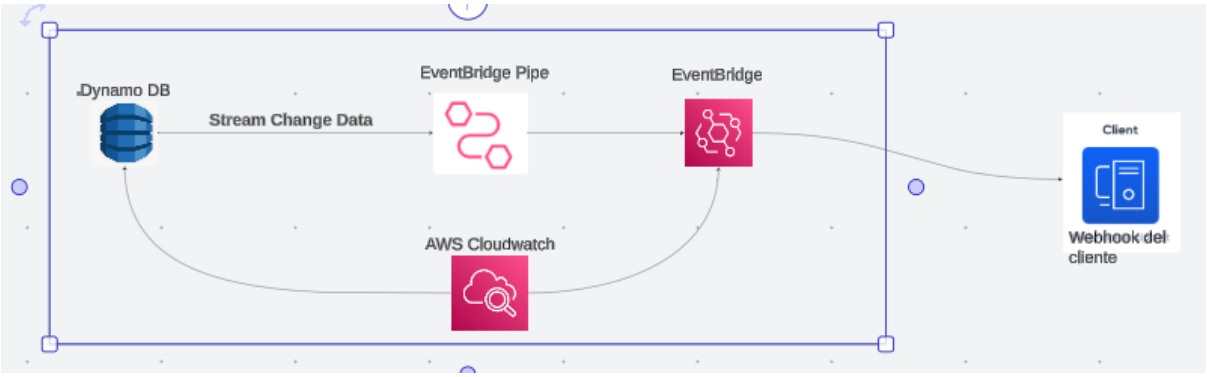
El contenido del mismo se muestra a continuación:

```
trigger:
- main  # la integracion de codigo en esta rama del repositorio dispara el pipeline

pool:
 vmImage: 'ubuntu-latest'  # Usamos una imagen de Ubuntu para el build

variables:
 awsRegion: 'us-east-1'  # Región del bucket S3 y recursos de CloudFormation
 s3BucketName: 'my-lambda-code-bucket'  # Nombre del bucket S3
 cloudFormationTemplatePath: 'path/to/IaCV1.yml'  # Ruta de la plantilla CloudFormation IaCV1.yml
 stackName: 'SendConfirmationStack'  # Nombre del stack CloudFormation
 artifactDirectory: $(Build.ArtifactStagingDirectory)  # Directorio de staging de artefactos

jobs:
- job: BuildAndDeploy
 steps:
 # 1. Este paso permite a las tareas posteriores usar los archivos del repositorio como archivos
locales
 - checkout: self

 # 2. Se comprimen los archivos que contienen el codigo de las lambdas
 - task: ArchiveFiles@2
   inputs:
     rootFolderOrFile: $(Build.SourcesDirectory)/lambda-email.py
     includeRootFolder: false
     archiveFile: $(artifactDirectory)/lambda-email.zip
```

```
    overwrite: true

- task: ArchiveFiles@2
  inputs:
    rootFolderOrFile: $(Build.SourcesDirectory)/lambda-confirmaciones.py
    includeRootFolder: false
    archiveFile: $(artifactDirectory)/lambda-confirmaciones.zip
    overwrite: true
 # 3. Instalar AWS CLI
- task: UsePythonVersion@0
  inputs:
    versionSpec: '3.x'
    addToPath: true

- script: |
    python -m pip install --upgrade pip
   pip install awscli  # Instalamos AWS CLI para interactuar con S3 y CloudFormation
  displayName: 'Instalar AWS CLI'

 # 4. Configurar credenciales de AWS, Estas claves deben ser configuradas dentro de un service
connection en Azure DevOps y estas variables definidas en el library asociado al repositorio
- script: |
    aws configure set aws_access_key_id $(awsAccessKeyId)
   aws configure set aws_secret_access_key $(awsSecretAccessKey)
   aws configure set default.region $(awsRegion)
  displayName: 'Configurar AWS CLI'

 # 5. Subir los archivos comprimidos al bucket de S3 asociado con las lambdas ya que estos contienen
el codigo de dichas lambdas.
- script: |
    aws s3 cp $(artifactDirectory) s3://$(s3BucketName)/ --recursive
  displayName: 'Subir todos los archivos a S3'

 # 6. Desplegar recursos con CloudFormation
- script: |
    aws cloudformation deploy \
     --template-file $(cloudFormationTemplatePath) \
     --stack-name $(stackName) \
     --capabilities CAPABILITY_NAMED_IAM \
     --region $(awsRegion)
  displayName: 'Desplegar stack de CloudFormation'
```

Propuesta 2

La segunda propuesta utiliza el servicio de AWS EventBridge para enviar las confirmaciones al webhook del cliente disparado por Dynamo streams.



La plantilla requerida para el despliegue de estos recursos usando AWS Cloudformation se entrega junto con este documento con el nombre de IaCV2.yml. El contenido de la misma se muestra a continuación.

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: Template para envío de confirmaciones de transacciones.

Resources:
 # Tabla DynamoDB con Streams
 DynamoDBTable:
   Type: "AWS::DynamoDB::Table"
   Properties:
     TableName: "MyDynamoDBTable"
     AttributeDefinitions:
       - AttributeName: "ID"
         AttributeType: "S"
       - AttributeName: "transaccion"
         AttributeType: "S"
       - AttributeName: "email"
         AttributeType: "S"
     KeySchema:
       - AttributeName: "ID"
         KeyType: "HASH"
     BillingMode: PAY_PER_REQUEST
     StreamSpecification:
       StreamViewType: NEW_AND_OLD_IMAGES

 # EventBridge Bus
 EventBus:
   Type: "AWS::Events::EventBus"
   Properties:
     Name: "MyEventBus"

 # EventBridge API Destination
 APIDestination:
   Type: "AWS::Events::ApiDestination"
   Properties:
     Name: "MyAPIDestination"
```

```
    ConnectionArn: !GetAtt APIDestinationConnection.Arn
    InvocationEndpoint: "https://plataforma.com/webhook"  # Reemplaza con la URL del webhook del
cliente
    HttpMethod: POST  # Método HTTP
    InvocationRateLimitPerSecond: 10  # Límite de invocaciones por segundo

 # EventBridge Connection
 APIDestinationConnection:
   Type: "AWS::Events::Connection"
   Properties:
     Name: "MyAPIDestinationConnection"
     AuthorizationType: "BASIC"  # Cambia según las necesidades del webhook
     AuthParameters:
       BasicAuthParameters:
         Username: "myUsername"  # Reemplaza con el nombre de usuario
         Password: "myPassword"  # Reemplaza con la contraseña (se debe configurar Secrets Manager
para mayor seguridad)

 # Regla de EventBridge
 EventRule:
   Type: "AWS::Events::Rule"
   Properties:
     Name: "MyEventRule"
     EventBusName: !Ref EventBus
     EventSource: !GetAtt DynamoDBTable.StreamArn  # Asocia los Streams de DynamoDB como fuente de
eventos
     EventPattern:
       source:
         - "aws.dynamodb"
       detail:
         eventName:
           - "INSERT"  # Procesa eventos de inserción; es decir que se activa siempre que llega
nueva informacion a la base de datos
     Targets:
       - Id: "MyWebhookTarget"
         Arn: !GetAtt APIDestination.Arn

Outputs:
 DynamoDBTableName:
   Value: !Ref DynamoDBTable
   Description: Nombre de la tabla DynamoDB

 DynamoDBStreamArn:
   Value: !GetAtt DynamoDBTable.StreamArn
   Description: ARN del Stream de la tabla DynamoDB

 EventBusName:
   Value: !Ref EventBus
   Description: Nombre del EventBus

 WebhookDestination:
   Value: "https://plataforma.com/webhook"
   Description: Webhook de cliente configurado como destino
```

También se propone en este caso un pipeline de despliegue de Azure DevOps. Dicho pipeline está descrito en el archivo azure-pipelineV2.yml entregado con este documento. El contenido del mismo se muestra a continuación.

```yaml
trigger:
- main

pool:
 vmImage: 'ubuntu-latest'

variables:
 awsRegion: 'us-east-1'  # Región de los recursos de CloudFormation
 cloudFormationTemplatePath: 'template/IaCV2.yml'  # Ruta de la plantilla CloudFormation IaCV2.yml
 stackName: 'SendconfimationStack'  # Nombre del stack CloudFormation

jobs:
- job: DeployResources
 steps:
 # 1. Clonar el repositorio
 - checkout: self  # Clona el código del repositorio

 # 2. Instalar AWS CLI
 - task: UsePythonVersion@0
   inputs:
     versionSpec: '3.x'
     addToPath: true

 - script: |
     python -m pip install --upgrade pip
    pip install awscli
   displayName: 'Instalar AWS CLI'

 # 3. Configurar credenciales de AWS
 - script: |
     aws configure set aws_access_key_id $(awsAccessKeyId)
    aws configure set aws_secret_access_key $(awsSecretAccessKey)
    aws configure set default.region $(awsRegion)
   displayName: 'Configurar AWS CLI'

 # 4. Desplegar recursos con CloudFormation
 - script: |
     aws cloudformation deploy \
      --template-file $(cloudFormationTemplatePath) \
      --stack-name $(stackName) \
      --capabilities CAPABILITY_NAMED_IAM \
      --region $(awsRegion)
   displayName: 'Desplegar stack de CloudFormation'
```

Monitoreo

Para ambas soluciones se propone el monitoreo de los recursos mediante el uso de AWS CloudWatch. Se propone crear grupos de logs que brindan información sobre métricas clave como invocaciones y errores en las lambdas, consumos de lectura y escritura de la base de datos y latencia de los streams de DynamoDB. Además se sugiere configurar alarmas para supervisar el comportamiento reglas, buses, y destinos de EventBridge.