



# Prueba backend para Merqueo

**César Andrés Villegas Tabares**

## Planeación:

Para el desarrollo de la prueba inicialmente se estableció como tiempo límite el primer fin de semana de enero, pero debido a obligaciones laborales la prueba no pudo ser desarrollada, por lo que fue reprogramada para para el segundo fin de semana de enero.

Teniendo en cuenta lo anterior, se dispone entonces abordar el problema de la siguiente manera:

1. Comprender el problema y los requerimientos que se desean abordar.
2. Definición de tecnologías a emplear.
3. Planificar tareas.

De las actividades anteriores se obtiene un mapa de ruta, una priorización y una selección de tecnologías, obteniendo entonces lo siguiente:

## Selección de tecnologías:

Para el desarrollo del problema se seleccionó:

- JavaScript con node js 12.X como lenguaje / entorno de ejecución de código para soportar lógica de negocio.
- MySQL como motor de bases de datos.
- AWS como proveedor de infraestructura.
- AWS cdk como framework para definir infraestructura como código.

- Typescript como lenguaje de definición de infraestructura.
- Jest como framework de pruebas.
- Postman para pruebas funcionales y documentación del API.
- Github para manejo de repositorio y servidor de integración a través de Github Actions.

Adicionalmente, se mantendrá un flujo de trabajo como el propuesto por **“GitHub Flow”**, en donde se propone mantener la rama **“master”** como el sistema de producción, y se crean ramas de desarrollo para features pequeñas que se prueban y se pasan a producción.

## Definición de estructura de base de datos:

Para la base de datos se estableció un esquema como el mostrado en la figura 1.

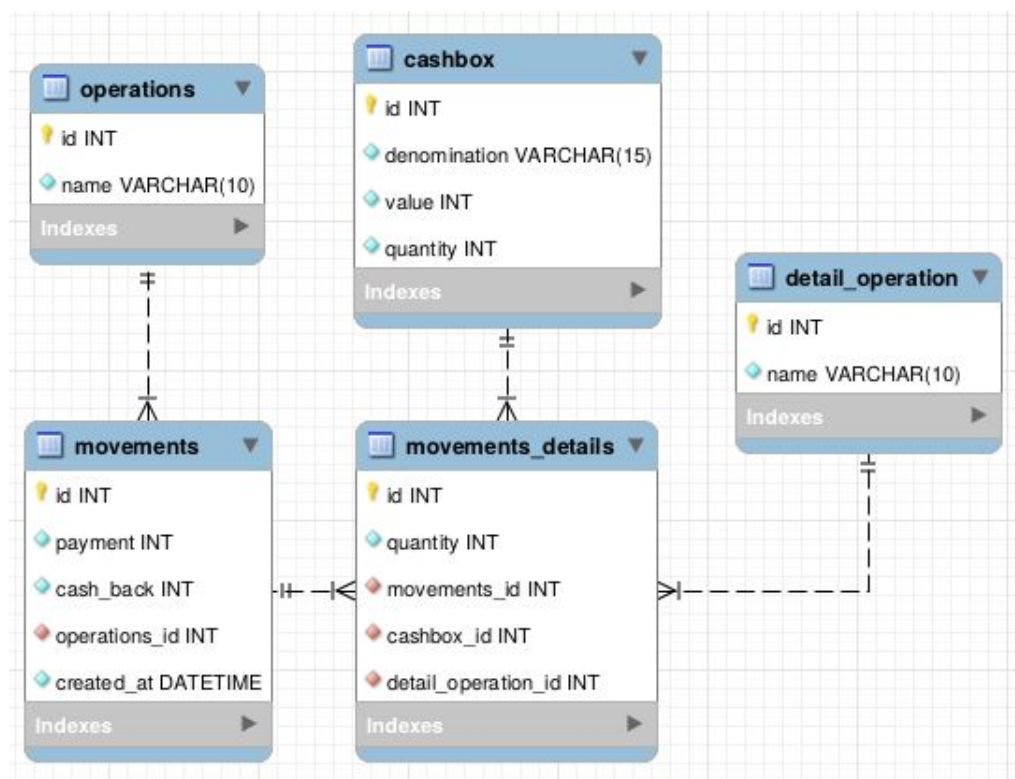


Figura 1. Diagrama e-r base de datos.

De esta manera, se pueden manejar los requerimientos de datos del problema, pero adicionalmente se puede escalar a otras denominaciones de billetes, y de ser necesario, se puede agregar una nueva tabla relacionada a **cashbox** en donde se agreguen nuevas cajas registradoras y se pueda escalar el sistema para manejar más cajas.

Se mantienen identificadores auto-incrementables a cada tabla y se establecen las relaciones necesarias para mantener la integridad.

Adicionalmente, se despliega una copia de la base de datos a un ambiente de pruebas que puede ser empleado para revisiones y pruebas. Las credenciales de acceso se dejaron en un archivo público.

## Inicio del proyecto y configuración inicial:

Como primera medida se inicia el repositorio correspondiente, se clona y se inicia el proyecto con AWS cdk, y aunque se mantiene la estructura propuesta por el framework, se agregan los paquetes requeridos para el desarrollo de la infraestructura, las funciones lambda empleadas y el layer de MySQL. Esto último es muy importante ya que este recurso es necesario para poder establecer comunicación con el servidor de base de datos, pero AWS no cuenta con esta librería entre sus recursos de uso común, por lo que es necesario agregarlo desde nuestro código.

## Flujo de integración:




Una vez lista la estructura básica del proyecto se procedió a configurar el flujo de integración. Para ello se configuraron los secretos con las credenciales de la cuenta de AWS en los “Secrets” de Github.

### Actions secrets

New repository secret

Secrets are environment variables that are **encrypted**. Anyone with **collaborator** access to this repository can use these secrets for Actions.

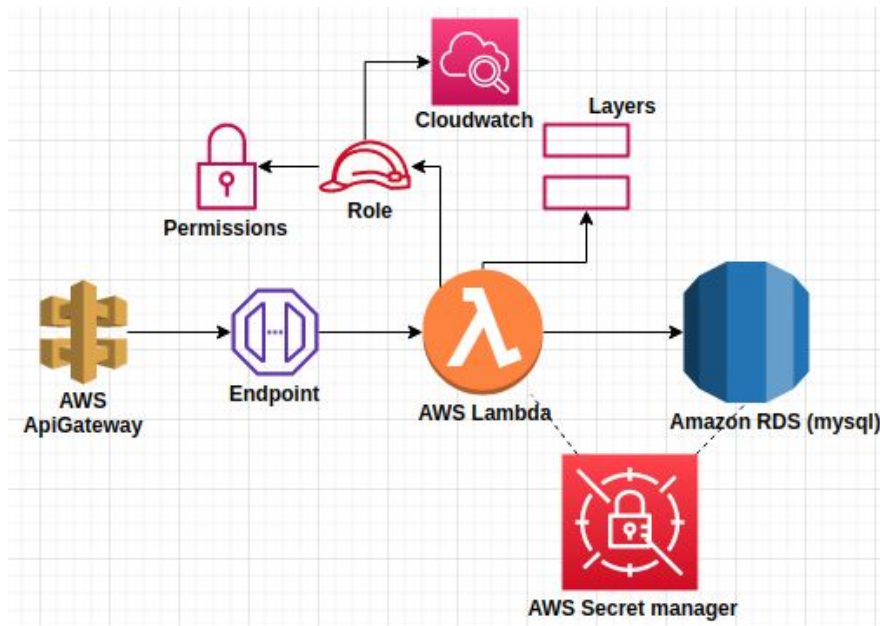
Secrets are not passed to workflows that are triggered by a pull request from a fork. [Learn more](#).

 AWS_ACCESS_KEY_ID	Updated 8 days ago	<button>Update</button>	<button>Remove</button>
 AWS_ACCOUNT_REGION	Updated 8 days ago	<button>Update</button>	<button>Remove</button>
 AWS_SECRET_ACCESS_KEY	Updated 8 days ago	<button>Update</button>	<button>Remove</button>

Se configura dentro del proyecto la carpeta .github/workflows -> y se agregan 2 archivos para manejar 2 flujos de trabajo diferentes, uno para ejecutar automáticamente el set de pruebas establecido al momento de hacer push de la rama de desarrollo, y uno para realizar pruebas y desplegar automáticamente el proyecto en AWS al momento de hacer merge a la rama master.

## Solución desarrollada:

Para la solución desarrollada se siguió un patrón serverless de cdk denominado “**API simple**” soportado por la siguiente arquitectura:



De lo anterior es muy importante tener en cuenta 2 cosas.

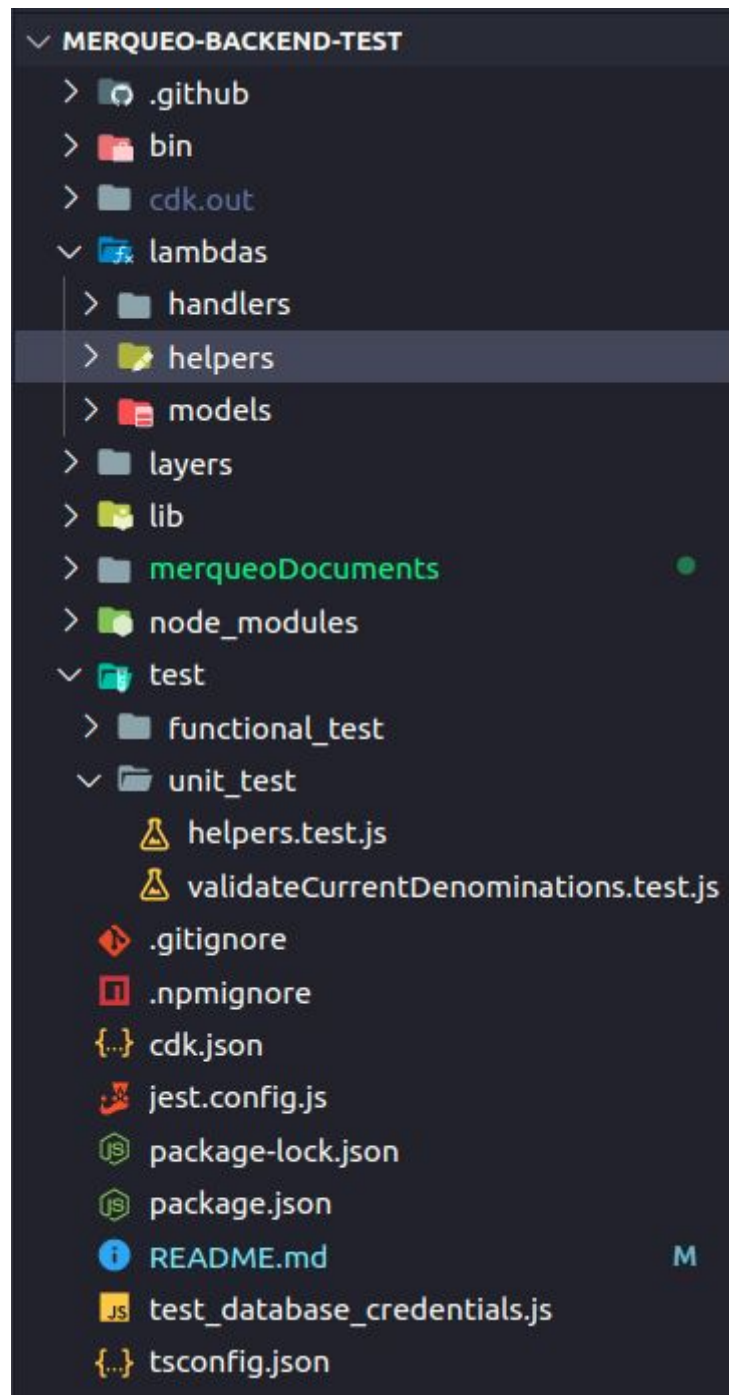
1. La base de datos no se encuentra dentro de la plantilla de AWS cdk, ya que esta se encuentra alojada en una instancia de base de datos gratuita que ya se encuentra desarrollada en la cuenta de AWS.
2. El servicio de **Secrets Manager** de AWS tiene configurados los accesos a la base de datos por razones de seguridad, y al momento de desplegar la infraestructura desde el framework, este accede al servicio de secrets manager, obtiene los parámetros de configuración de la base de datos y los agrega como variables de entorno de las funciones lambda dentro del ecosistema de AWS, por lo que estas variables nunca quedan expuestas en el código.

Dentro de la infraestructura se despliega una sola API, ya esta se le agregan los diferentes endpoints requeridos con las rutas personalizadas correspondientes, y se integran dichos endpoints a una función lambda específica.

La estructura de carpetas propuesta contiene los siguientes elementos:

- .Github/workflows -> contiene los flujos propuestos para el proceso CI con github.
- bin -> Punto de entrada de la aplicación para cdk, utiliza la pila de recursos definidos.
- lambdas -> contiene las funciones lambda que soportan la lógica de negocio.

- handlers -> funciones que se integran directamente con cada endpoint desarrollado.
- helpers -> funciones auxiliares para apoyar la lógica de negocio.
- models -> modelos de gestión de accesos a la información.
- layers -> librerías externas que se deben cargar a AWS.
- lib -> definición del stack de recursos AWS que serán desplegados.
- merqueoDocuments -> Documentación relacionada con la prueba.



- node\_modules -> dependencias del proyecto (npm).

- test -> contiene las pruebas especificadas para el proyectos.
  - functional\_test: test funcionales especificados.
  - unit\_test: pruebas unitarias especificadas.
- .gitignore -> archivos que serán ignorados por git.
- cdk.json -> “guía” de ejecución del framework.
- jest.config -> configuración del framework de pruebas.
- package.json -> manifiesto de npm.
- Readme.md -> Guía básica de despliegue de cdk y pruebas.
- test\_database\_credentials.js -> archivo de configurado agregado intencionalmente con credenciales de acceso a una base de datos de prueba que es una copia exacta de la base de producción, y puede ser empleado por desarrolladores y personas que revisen la prueba para correr pruebas unitarias y demás.
- tsconfig.json -> configuración de typescript en el proyecto.

Para el desarrollo del proyecto, además de utilizar el patrón de API simple ya mencionado, se estructuró un patrón **singleton** para instanciar el conector de base de datos, de manera que solo se mantenga una instancia y se mantengan tanto datos como tiempo de ejecución en el mínimo posible. Adicionalmente, se agregó un archivo que hace las veces de DAO y de esta manera mantener un poco mejor la separación entre capas.

Los endpoints expuestos son.

- **Estado de la caja: (GET) ->**  
[https://z4ryevcvz9.execute-api.us-east-1.amazonaws.com/prod/merqueo/cash\\_box/get\\_status](https://z4ryevcvz9.execute-api.us-east-1.amazonaws.com/prod/merqueo/cash_box/get_status)
- **Vaciar la caja: (POST) ->**  
[https://z4ryevcvz9.execute-api.us-east-1.amazonaws.com/prod/merqueo/cash\\_box/empty](https://z4ryevcvz9.execute-api.us-east-1.amazonaws.com/prod/merqueo/cash_box/empty)
- **Cargar base a la caja: (POST) ->**  
[https://z4ryevcvz9.execute-api.us-east-1.amazonaws.com/prod/merqueo/cash\\_box/set\\_base](https://z4ryevcvz9.execute-api.us-east-1.amazonaws.com/prod/merqueo/cash_box/set_base)
- **Realizar un pago: (POST) ->**  
[https://z4ryevcvz9.execute-api.us-east-1.amazonaws.com/prod/merqueo/payment\\_register](https://z4ryevcvz9.execute-api.us-east-1.amazonaws.com/prod/merqueo/payment_register)
- **Ver registro de logs: (GET) ->**  
[https://z4ryevcvz9.execute-api.us-east-1.amazonaws.com/prod/merqueo/get\\_movements](https://z4ryevcvz9.execute-api.us-east-1.amazonaws.com/prod/merqueo/get_movements)
- **Saber el estado de la caja en una fecha específica: (GET?date\_required):**  
[https://z4ryevcvz9.execute-api.us-east-1.amazonaws.com/prod/merqueo/cash\\_box/get\\_previous?date\\_required=2021-01-11 10:37:38](https://z4ryevcvz9.execute-api.us-east-1.amazonaws.com/prod/merqueo/cash_box/get_previous?date_required=2021-01-11 10:37:38)

La documentación del API puede verse en el link expuesto empleando Postman. ([link](#))

## Mejoras probables y faltantes:.

Corresponde a elementos que no alcance a hacer por razones de tiempo.

- **Refactorización y desacople:** es posible refactorizar un poco más y lograr un mayor desacople.
- **Mejorar coverage:** Con el desacople es posible mejorar mucho el coverage de pruebas.
- **Pruebas de código estático:** Es posible agregar dentro de los flujos planteados en github actions un lintern o una imagen de sonar que permita realizar pruebas de código estático.
- **Modificación de base de datos para usar dynamodb:** implementar un diseño de tipo single design table y agregar la estructura de dynamo al template de CDK, de manera que se tenga una estructura totalmente serverless y controlada por código.
- **Autenticación:** agregar algunas medidas de autenticación sencillas al API.