En este documento, se analizará la arquitectura de la aplicación presentada en el enunciado de este TP, y se hará una crítica a algunos de los puntos de la arquitectura.

Descripción de componentes

La aplicación no cuenta con algún **componente frontend** o página web con la que se pueda interactuar, solo se realizan solicitudes a una **API** con un selecto grupo de **servicios** (Endpoints), que no guardan u obtienen información de una base de datos, sino de unos **archivos JSON**. Podrian guardarse los datos en un mejor lugar y no en archivos, como por ejemplo, en Redis.
Pero antes de ingresar la solicitud a la API, pasa por un **load balancer** (Nginx) para que controle el tráfico de las solicitudes y las distribuya así se evita una sobrecarga.

Como se ve, esta aplicación es muy simple y con una pequeñísima cantidad de componentes, solo un **load balancer** para el control de las solicitudes, la **API** para el procesamiento de solicitudes, y los **archivos JSON** como forma de almacenar y persistir los datos.

Otras herramientas utilizadas dentro del proyecto como Docker, Artillery y demás, no forman parte de los componentes ya que no son parte de la funcionalidad principal de la aplicación. Sin embargo, podrían incluirse dentro de las capas de la aplicación

Capas

Dentro del proyecto, las partes están bien separadas, y cada parte tiene su propia responsabilidad y no está cargada de alguna responsabilidad que no deba tener, realizando una tarea en específico. Estas partes pueden separarse en 5:

1. Capa de presentación

Recibe las solicitudes HTTP. Su componente es el **Nginx** que es el load balancer.

2. Capa de aplicación
Procesa las solicitudes y aplica la lógica del sistema. Sus componentes son **Node.js** y **Express**

3. Capa de datos
Guardar los datos, permite recuperarlos y también los persiste. Su componente son los archivos **JSON** de los cuales se obtiene información y hacia los cuales se guarda nueva información.

4. Capa de monitoreo
Mide el rendimiento, permite ver y graficar métricas. Sus componentes son **Artillery** (Pruebas de carga), **Grafana** (Visualizar métricas) y **StatsD**, **cAdvisor** y **Graphite** (Obtener métricas)

5. Capa de infraestructura
Orquesta todos los servicios y conecta todas las partes de la aplicación. Sus componentes son **Docker** (Contenerizar servicios) y **Docker compose** (Orquestar servicios)

Servicios

1. Tasas de cambio
Tiene la responsabilidad de obtener y modificar las tasas de cambio. Cuando se altera una tasa de cambio también calcula la recíproca.

Por como se desarrolló este endpoint, no hay una ganancia para arVault por la conversión (O sea no se cobra comisión por la conversión)

También se sugiere que se pueden crear niveles de usuarios, y darle una mejor tasa a un usuario de mayor nivel. Esto abre a la idea de **escalabilidad** para el sistema.

2. Cuentas
Obtiene las cuentas y modifica sus saldos. Por cómo está diseñado el método PUT, el endpoint de cuentas tiene más responsabilidad de la que debería, no

debería poder modificar su saldo. Para eso, se puede crear otro servicio que lo haga

3. Cambio

Registra las operaciones de cambios. Las validaciones de saldo suficiente están hechas para la cuenta propia pero no para la cuenta del cliente.

4. Logs

Devuelve un log de operaciones.