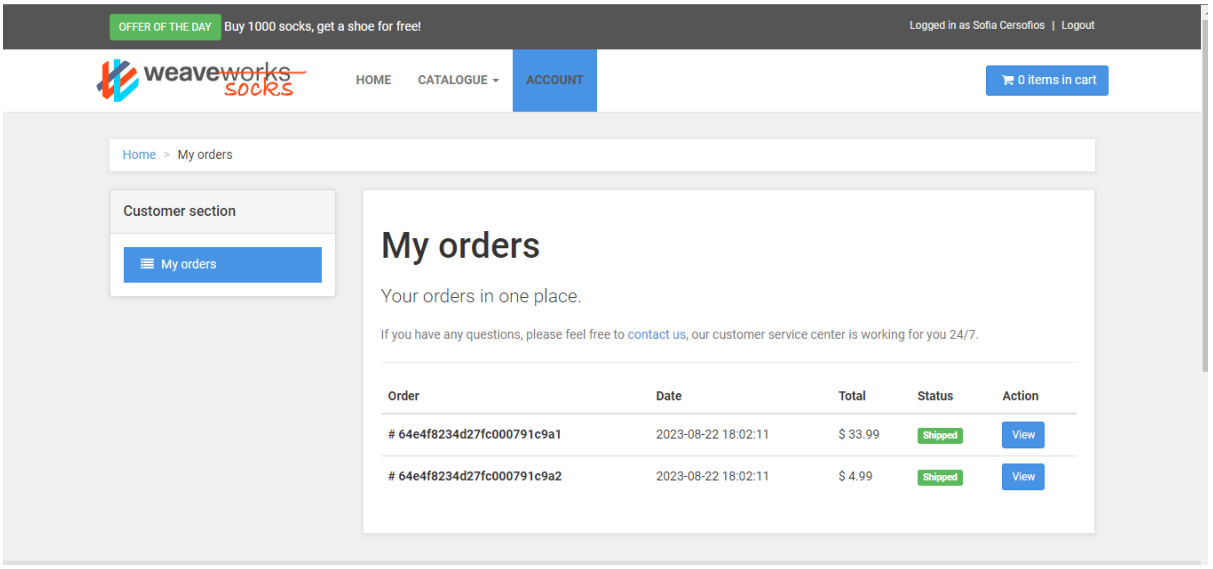
TP 4 ARQUITECTURA DE MICROSERVICIOS



2- Investigación de los componentes

- Describa los contenedores creados, indicando cuales son los puntos de ingreso del sistema

```
PS C:\Users\sofia\OneDrive\Escritorio\microservicios\microservices-demo> docker ps
CONTAINER ID   IMAGE                                  COMMAND                  CREATED          STATUS          PORTS
      NAMES
bb631d4adb72   weaveworksdemos/catalogue-db:0.3.0     "docker-entrypoint.s…"   32 minutes ago   Up 31 minutes   3306/tcp
      docker-compose_catalogue-db_1
84f8425e7c06   weaveworksdemos/front-end:0.3.12       "/usr/local/bin/npm …"   32 minutes ago   Up 31 minutes   8079/tcp
      docker-compose_front-end_1
51de40d4b7ca   mongo:3.4                              "docker-entrypoint.s…"   32 minutes ago   Up 31 minutes   27017/tcp
      docker-compose_carts-db_1
93c110de6fde   weaveworksdemos/user-db:0.4.0          "/entrypoint.sh mong…"   32 minutes ago   Up 31 minutes   27017/tcp
      docker-compose_user-db_1
f4c980a522cd   rabbitmq:3.6.8                         "docker-entrypoint.s…"   32 minutes ago   Up 31 minutes   4369/tcp, 5671-5672/tcp, 25672/tcp
      docker-compose_rabbitmq_1
4f5e45661bad   weaveworksdemos/catalogue:0.3.5        "/app -port=80"          32 minutes ago   Up 31 minutes   80/tcp
      docker-compose_catalogue_1
fc0b89ba2b6d   weaveworksdemos/carts:0.4.8            "/usr/local/bin/java…"   32 minutes ago   Up 31 minutes
      docker-compose_carts_1
5d99b254ee47   weaveworksdemos/edge-router:0.1.1      "traefik"                32 minutes ago   Up 31 minutes   0.0.0.0:80->80/tcp, 0.0.0.0:8080->8080
/tcp   docker-compose_edge-router_1
590cb6dc04e6   weaveworksdemos/payment:0.4.3          "/app -port=80"          32 minutes ago   Up 31 minutes   80/tcp
      docker-compose_payment_1
cc6f8506727f   weaveworksdemos/user:0.4.4             "/user -port=80"         32 minutes ago   Up 31 minutes   80/tcp
      docker-compose_user_1
d1195659e7d6   weaveworksdemos/shipping:0.4.8         "/usr/local/bin/java…"   32 minutes ago   Up 31 minutes
      docker-compose_shipping_1
b9f40bd6acf2   weaveworksdemos/queue-master:0.3.1     "/usr/local/bin/java…"   32 minutes ago   Up 31 minutes
      docker-compose_queue-master_1
2bc671e64622   mongo:3.4                              "docker-entrypoint.s…"   32 minutes ago   Up 31 minutes   27017/tcp
      docker-compose_orders-db_1
c0c108528849   weaveworksdemos/orders:0.4.7           "/usr/local/bin/java…"   32 minutes ago   Up 31 minutes
      docker-compose_orders_1
```

front-end:
Imagen: weaveworksdemos/front-end:0.3.12
Punto de entrada: Interfaz de usuario web (front-end) para interactuar con el sistema.

edge-router:
Imagen: weaveworksdemos/edge-router:0.1.1
Puertos expuestos: 80 (HTTP) y 8080
Punto de entrada: Router de borde que enruta el tráfico entrante a diferentes componentes del sistema.

catalogue:
Imagen: weaveworksdemos/catalogue:0.3.5

Punto de entrada: Servicio de catálogo para listar productos disponibles.

catalogue-db:
Imagen: weaveworksdemos/catalogue-db:0.3.0
Punto de entrada: Base de datos MySQL para el catálogo de productos.

carts:
Imagen: weaveworksdemos/carts:0.4.8
Punto de entrada: Servicio de carrito de compras para gestionar los artículos en el carrito.

carts-db:
Imagen: mongo:3.4
Punto de entrada: Base de datos MongoDB para el carrito de compras.

orders:
Imagen: weaveworksdemos/orders:0.4.7
Punto de entrada: Servicio de pedidos para realizar y gestionar pedidos.

orders-db:
Imagen: mongo:3.4
Punto de entrada: Base de datos MongoDB para los pedidos.

shipping:
Imagen: weaveworksdemos/shipping:0.4.8
Punto de entrada: Servicio de envío para gestionar los envíos de los pedidos.

queue-master:
Imagen: weaveworksdemos/queue-master:0.3.1
Punto de entrada: Gestión de colas y planificación de tareas para el sistema.

rabbitmq:
Imagen: rabbitmq:3.6.8
Punto de entrada: Servicio de cola de mensajes RabbitMQ.

payment:
Imagen: weaveworksdemos/payment:0.4.3
Punto de entrada: Servicio de pagos para gestionar transacciones de pago.

user:
Imagen: weaveworksdemos/user:0.4.4
Punto de entrada: Servicio de usuarios para gestionar cuentas de usuario.

user-db:
Imagen: weaveworksdemos/user-db:0.4.0
Punto de entrada: Base de datos MongoDB para la gestión de usuarios.

user-sim:
Imagen: weaveworksdemos/load-test:0.1.1

Punto de entrada: Simulador de carga para probar el sistema con una alta carga de usuarios.

- Clonar algunos de los repositorios con el código de las aplicaciones

```
cd socks-demo
git clone https://github.com/microservices-demo/front-end.git
git clone https://github.com/microservices-demo/user.git
git clone https://github.com/microservices-demo/edge-router.git
```

- ¿Por qué cree usted que se está utilizando repositorios separados para el código y/o la configuración del sistema? Explique puntos a favor y en contra.

El uso de repositorios separados para el código y/o la configuración de un sistema es una práctica común en el desarrollo de microservicios y aplicaciones distribuidas. Esta elección generalmente se basa en varios factores y tiene ventajas y desventajas.

Ventajas de utilizar repositorios separados:

- Desarrollo y despliegue independiente: los equipos de desarrollo pueden trabajar de manera independiente en sus respectivas áreas sin afectar directamente a otros equipos. Esto permite una mayor agilidad en el desarrollo y despliegue de nuevos cambios.

- Aislamiento de cambios: si se presenta un problema o se requiere una actualización en un componente específico, es más fácil aislar y gestionar esos cambios en un repositorio independiente sin afectar a otros componentes.

- Escalabilidad y paralelismo: diferentes equipos pueden trabajar simultáneamente en distintos repositorios, lo que permite una mejor utilización de recursos y la posibilidad de escalar equipos y procesos por separado.

- Gestión de versiones: cada repositorio puede tener su propio ciclo de versionado, lo que facilita la gestión de versiones específicas para cada componente sin depender de otros componentes.

Desventajas de utilizar repositorios separados:

- Mayor complejidad inicial: mantener múltiples repositorios puede aumentar la complejidad inicial del proyecto, ya que es necesario gestionar y sincronizar las interacciones entre los componentes.

- Configuración y coordinación: pueden volverse más complicadas, especialmente cuando se trata de implementaciones y pruebas.

- Mayor sobrecarga de administración: mantener múltiples repositorios implica una mayor sobrecarga de administración, ya que se deben gestionar los permisos de acceso, las ramas, las integraciones y otras tareas relacionadas en cada repositorio.

- Mayor riesgo de fragmentación: si no se gestionan adecuadamente, los repositorios separados pueden llevar a la fragmentación del código y los recursos, lo que dificulta la comprensión del sistema en su conjunto.

- ¿Cuál contenedor hace las veces de API Gateway?

Un API Gateway es un componente que actúa como punto de entrada único para todas las solicitudes de la aplicación, proporcionando funciones como enrutamiento, autenticación, autorización, equilibrio de carga y otras tareas relacionadas con la gestión del tráfico.
El contenedor "**edge-router**" realiza las funciones de un "API Gateway". En la descripción de los servicios, se puede ver que el contenedor está configurado para exponer los puertos 80 (HTTP) y 8080, lo que sugiere que está actuando como un punto de entrada principal para el tráfico web entrante.

- Cuando ejecuto este comando:

curl http://localhost/customers

```
PS C:\Users\sofia\OneDrive\Escritorio\microservicios\microservices-demo> curl http://localhost/customers


StatusCode        : 200
StatusDescription : OK
Content           : {"_embedded":{"customer":[{"firstName":"Eve","lastName":"Berger","username":"Eve_Berger","id":"57a9
                    8d98e4b00679b4a830af","_links":{"addresses":{"href":"http://user/customers/57a98d98e4b00679b4a830af
                    /a...
RawContent        : HTTP/1.1 200 OK
                    Date: Tue, 22 Aug 2023 18:21:01 GMT
                    Set-Cookie: _TRAEFIK_BACKEND=http://front-end:8079,md.sid=s%3A05VenPs5D06XRKQGg8r_gVp3JXBl2WuW.pNTe
                    x23LwQKUvNyaq1c5MBJiwaglmYpdcf31OUnvuS4; Path=/...
Forms             : {}
Headers           : {[Date, Tue, 22 Aug 2023 18:21:01 GMT], [Set-Cookie, _TRAEFIK_BACKEND=http://front-end:8079,md.sid=
                    s%3A05VenPs5D06XRKQGg8r_gVp3JXBl2WuW.pNTex23LwQKUvNyaq1c5MBJiwaglmYpdcf31OUnvuS4; Path=/;
                    HttpOnly], [X-Powered-By, Express], [Content-Length, 1599]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 1599
```

- ¿Cuál de todos los servicios está procesando la operación?

El servicio que está procesando la operación es el de **user.**

```
func MakeEndpoints(s Service, tracer stdopentracing.Tracer) Endpoints {
        return Endpoints{
                LoginEndpoint:       opentracing.TraceServer(tracer, "GET /login")(MakeLoginEndpoint(s)),
                RegisterEndpoint:    opentracing.TraceServer(tracer, "POST /register")(MakeRegisterEndpoint(s)),
                HealthEndpoint:      opentracing.TraceServer(tracer, "GET /health")(MakeHealthEndpoint(s)),
                UserGetEndpoint:     opentracing.TraceServer(tracer, "GET /customers")(MakeUserGetEndpoint(s)),
                UserPostEndpoint:    opentracing.TraceServer(tracer, "POST /customers")(MakeUserPostEndpoint(s)),
                AddressGetEndpoint:  opentracing.TraceServer(tracer, "GET /addresses")(MakeAddressGetEndpoint(s)),
                AddressPostEndpoint: opentracing.TraceServer(tracer, "POST /addresses")(MakeAddressPostEndpoint(s)),
                CardGetEndpoint:     opentracing.TraceServer(tracer, "GET /cards")(MakeCardGetEndpoint(s)),
                DeleteEndpoint:      opentracing.TraceServer(tracer, "DELETE /")(MakeDeleteEndpoint(s)),
                CardPostEndpoint:    opentracing.TraceServer(tracer, "POST /cards")(MakeCardPostEndpoint(s)),
        }
}
```

- ¿Y para los siguientes casos?

curl http://localhost/catalogue

```
PS C:\Users\sofia\OneDrive\Escritorio\microservicios\microservices-demo> curl http://localhost/catalogue


StatusCode        : 200
StatusDescription : OK
Content           : [{"id":"03fef6ac-1896-4ce8-bd69-b798f85c6e0b","name":"Holy","description":"Socks fit for a Messiah. You too can experience
                    walking in water with these special edition beauties. Each hole is lovingly p...
RawContent        : HTTP/1.1 200 OK
                    Date: Tue, 22 Aug 2023 18:38:40 GMT
                    Set-Cookie: _TRAEFIK_BACKEND=http://front-end:8079,md.sid=s%3A9rCd51oOri6ZG_2x4dNAy-kEfkexwIdv.abjt2hhLqsyThXz5wSJJ3wgFfg34AxgY
                    96u2wWrtK8A; Path=/...
Forms             : {}
Headers           : {[Date, Tue, 22 Aug 2023 18:38:40 GMT], [Set-Cookie, _TRAEFIK_BACKEND=http://front-end:8079,md.sid=s%3A9rCd51oOri6ZG_2x4dNAy-kE
                    fkexwIdv.abjt2hhLqsyThXz5wSJJ3wgFfg34AxgY96u2wWrtK8A; Path=/; HttpOnly], [X-Powered-By, Express], [Content-Type, text/plain;
                    charset=utf-8]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 2795
```

El servicio que está procesando la operación es el de **catalogue**.

```
module.exports = {
  catalogueUrl:  util.format("http://catalogue%s", domain),
  tagsUrl:       util.format("http://catalogue%s/tags", domain),
  cartsUrl:      util.format("http://carts%s/carts", domain),
  ordersUrl:     util.format("http://orders%s", domain),
  customersUrl:  util.format("http://user%s/customers", domain),
  addressUrl:    util.format("http://user%s/addresses", domain),
  cardsUrl:      util.format("http://user%s/cards", domain),
  loginUrl:      util.format("http://user%s/login", domain),
  registerUrl:   util.format("http://user%s/register", domain),
};
}());
```

curl http://localhost/tags

```
PS C:\Users\sofia\OneDrive\Escritorio\microservicios\microservices-demo> curl http://localhost/tags


StatusCode        : 200
StatusDescription : OK
Content           : {"tags":["brown","geek","formal","blue","skin","red","action","sport","black","magic","green"],"err":null}

RawContent        : HTTP/1.1 200 OK
                    Date: Tue, 22 Aug 2023 18:42:29 GMT
                    Set-Cookie: _TRAEFIK_BACKEND=http://front-end:8079,md.sid=s%3AA9PtoC2YTd5mSRAeZcY-PiWUkTHkxn1m.UkPRIQis10eS0HjQWhtyb9SWbFVxrL57
                    7XdzVvoy0A0; Path=/...
Forms             : {}
Headers           : {[Date, Tue, 22 Aug 2023 18:42:29 GMT], [Set-Cookie, _TRAEFIK_BACKEND=http://front-end:8079,md.sid=s%3AA9PtoC2YTd5mSRAeZcY-PiWU
                    kTHkxn1m.UkPRIQis10eS0HjQWhtyb9SWbFVxrL577XdzVvoy0A0; Path=/; HttpOnly], [X-Powered-By, Express], [Content-Length, 107]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 107
```

El servicio que está procesando la operación es el de **front-end**.

```
module.exports = {
  catalogueUrl:  util.format("http://catalogue%s", domain),
  tagsUrl:       util.format("http://catalogue%s/tags", domain),
  cartsUrl:      util.format("http://carts%s/carts", domain),
  ordersUrl:     util.format("http://orders%s", domain),
  customersUrl:  util.format("http://user%s/customers", domain),
  addressUrl:    util.format("http://user%s/addresses", domain),
  cardsUrl:      util.format("http://user%s/cards", domain),
  loginUrl:      util.format("http://user%s/login", domain),
  registerUrl:   util.format("http://user%s/register", domain),
};
}());
```

- ¿Cómo persisten los datos de los servicios?

En la configuración proporcionada, algunos de los servicios utilizan bases de datos para persistir los datos. Por ejemplo, los servicios "catalogue-db", "carts-db", "orders-db", "user-db" y "user-sim" utilizan bases de datos MongoDB o MySQL para almacenar y gestionar los datos relacionados con el catálogo de productos, carritos de compras, pedidos y usuarios. Estas bases de datos proporcionan la capacidad de almacenar datos de manera duradera entre las operaciones del sistema.

- ¿Cuál es el componente encargado del procesamiento de la cola de mensajes?

En la configuración, el contenedor "rabbitmq" está destinado a actuar como un sistema de cola de mensajes. RabbitMQ es un software de mensajería que permite la comunicación asíncrona. Los microservicios pueden enviar y recibir mensajes a través de la cola para manejar tareas en segundo plano, comunicación entre servicios o procesamiento de eventos.

- ¿Qué tipo de interfaz utilizan estos microservicios para comunicarse?

Los microservicios suelen comunicarse entre sí utilizando interfaces de red, como HTTP/HTTPS, para realizar solicitudes y respuestas. En esta configuración, los microservicios están utilizando el protocolo HTTP para comunicarse entre sí. Por ejemplo, los contenedores "front-end" y "user" están configurados para escuchar en puertos específicos y procesar solicitudes HTTP.