

Trabajo Práctico 4 - Arquitectura de Microservicios

1- Instanciación del sistema

Clonar el repositorio <https://github.com/microservices-demo/microservices-demo>

`mkdir -p socks-demo`

`cd socks-demo`

`git clone https://github.com/microservices-demo/microservices-demo.git`

```
PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4> mkdir -p socks-demo

Directorio: C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4

Mode                LastWriteTime         Length Name
----                -
d-----          16/10/2023         13:59         socks-demo

PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4> cd socks-demo
PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-demo> ls

Directorio: C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-demo

Mode                LastWriteTime         Length Name
----                -
d-----          16/10/2023         13:58         microservices-demo

PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-demo> |
```

Ejecutar lo siguiente

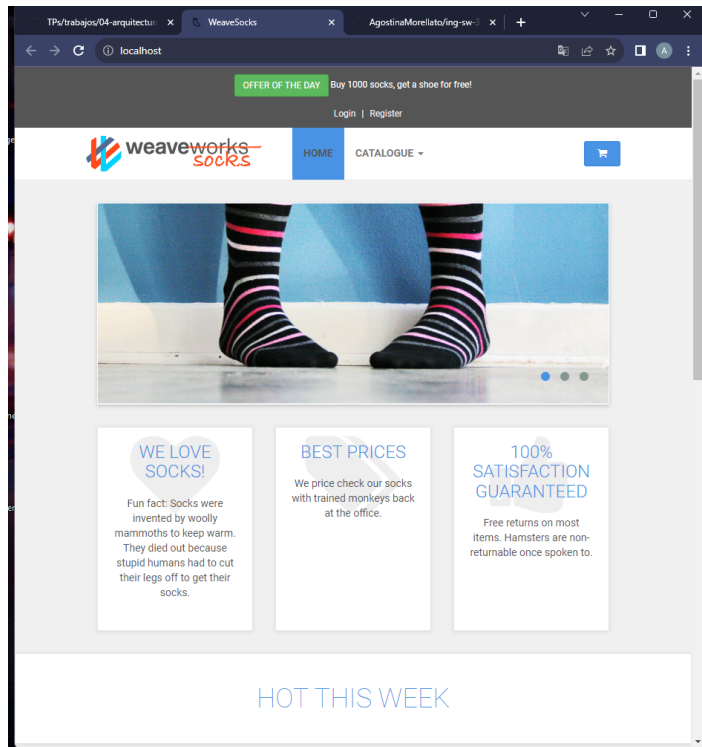
`cd microservices-demo`

`docker-compose -f deploy/docker-compose/docker-compose.yml up -d`

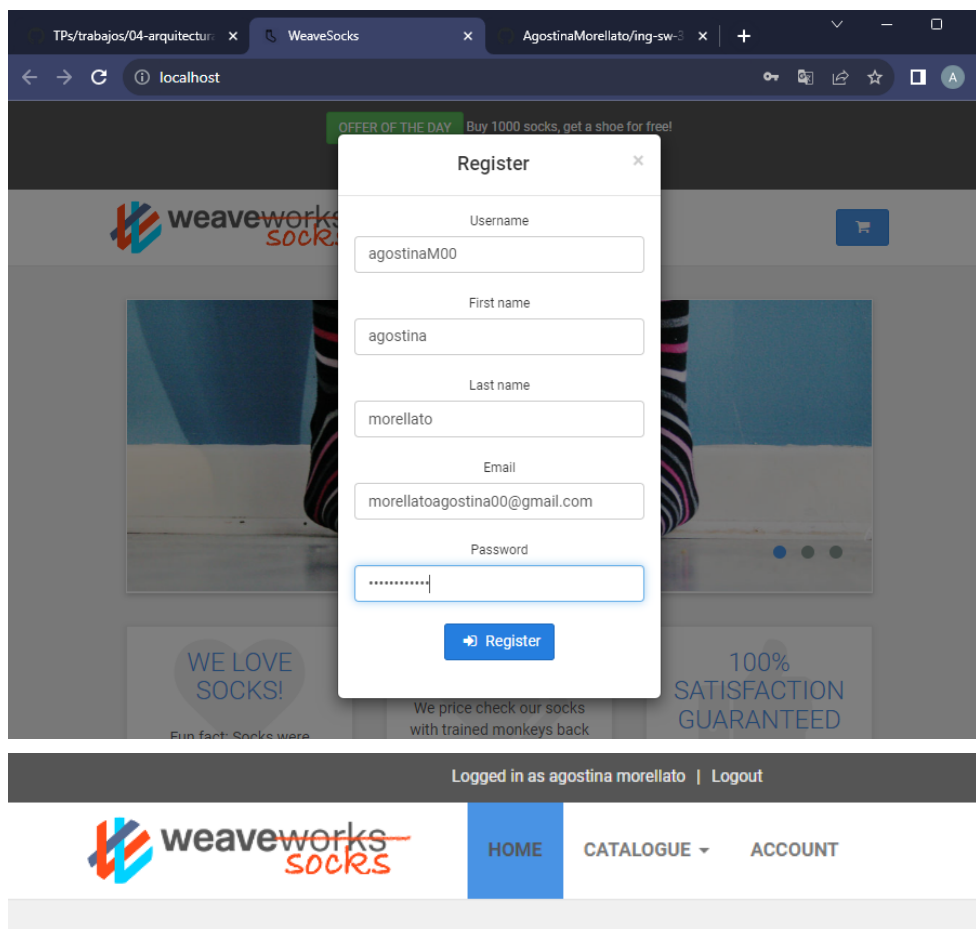
```
Windows PowerShell
PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-demo> cd microservices-demo
PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-demo\microservices-demo> docker-compose -f deploy/docker-com
pose/docker-compose.yml up -d
time="2023-10-16T14:00:31-03:00" level=warning msg="The \"MYSQL_ROOT_PASSWORD\" variable is not set. Defaulting to a bla
nk string."
[+] Running 117/24
  ✓ orders 3 layers [####] 0B/0B Pulled 133.3s
  ✓ front-end 8 layers [#####] 0B/0B Pulled 150.4s
  ✓ orders-db Pulled 195.0s
  ✓ carts-db 14 layers [#####] 0B/0B Pulled 195.0s
  ✓ carts 9 layers [#####] 0B/0B Pulled 33.7s
  ✓ rabbitmq 14 layers [#####] 0B/0B Pulled 129.3s
  ✓ edge-router 4 layers [####] 0B/0B Pulled 147.4s
  ✓ shipping 3 layers [###] 0B/0B Pulled 139.3s
  ✓ catalogue 4 layers [####] 0B/0B Pulled 153.0s
  ✓ user-db 11 layers [#####] 0B/0B Pulled 69.3s
  ✓ user-sim 11 layers [#####] 0B/0B Pulled 120.5s
  ✓ catalogue-db 12 layers [#####] 0B/0B Pulled 84.0s
  ✓ queue-master 2 layers [##] 0B/0B Pulled 162.7s
  ✓ user 3 layers [###] 0B/0B Pulled 157.5s
  ✓ payment 4 layers [####] 0B/0B Pulled 27.8s

[+] Building 0.0s (0/0) docker:default
[+] Running 16/16
  ✓ Network docker-compose_default Created 0.1s
  ✓ Container docker-compose-queue-master-1 Started 0.6s
  ✓ Container docker-compose-edge-router-1 Started 0.6s
  ✓ Container docker-compose-catalogue-1 Started 0.6s
  ✓ Container docker-compose-orders-1 Started 0.6s
  ✓ Container docker-compose-payment-1 Started 0.5s
  ✓ Container docker-compose-carts-db-1 Started 0.6s
  ✓ Container docker-compose-catalogue-db-1 Started 0.6s
  ✓ Container docker-compose-carts-1 Started 0.6s
  ✓ Container docker-compose-user-1 Started 0.5s
  ✓ Container docker-compose-shipping-1 Started 0.5s
  ✓ Container docker-compose-user-sim-1 Started 0.6s
```

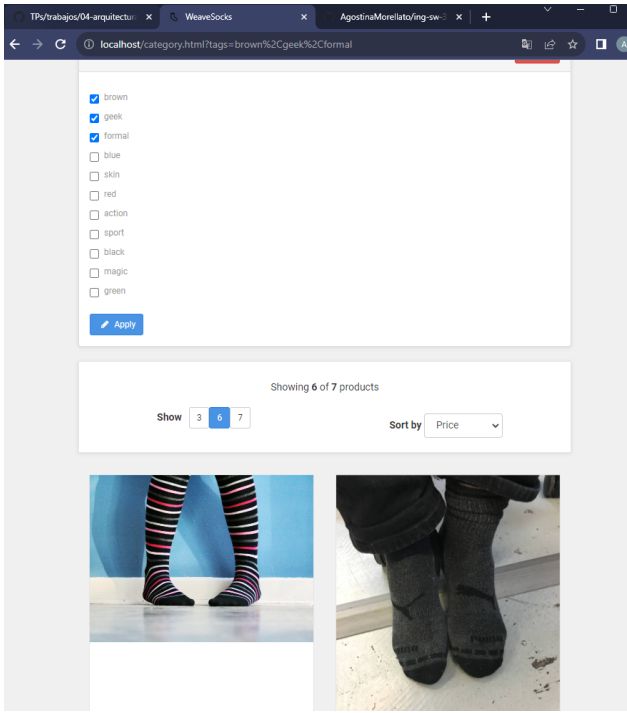
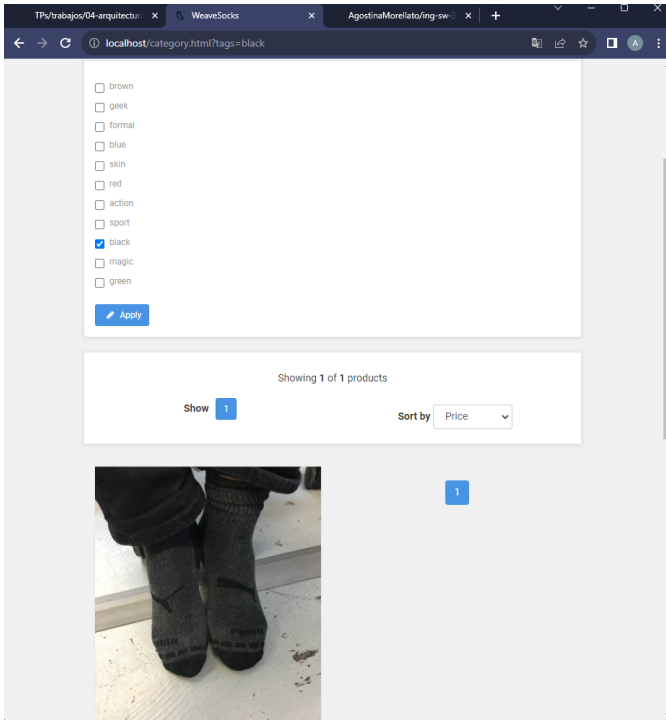
Una vez terminado el comando docker-compose acceder a <http://localhost>



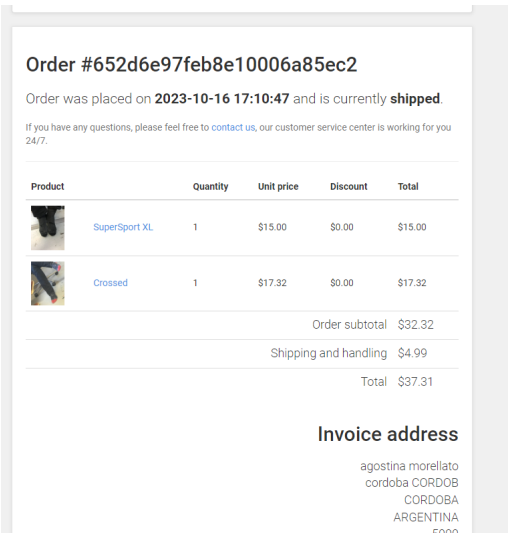
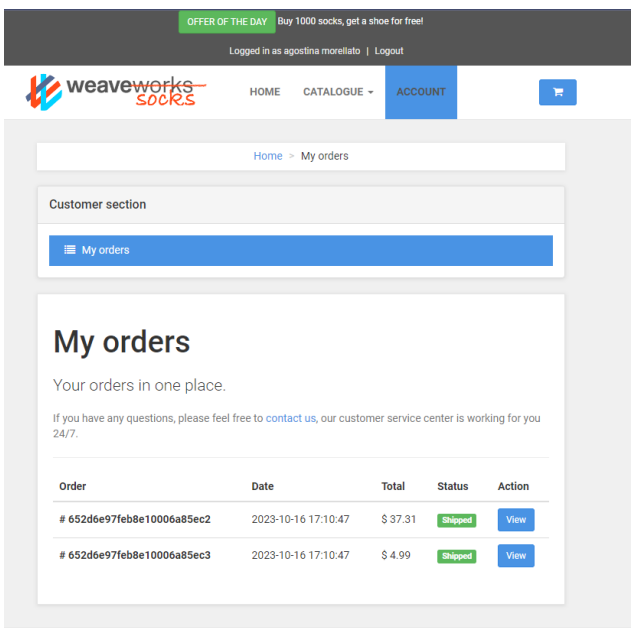
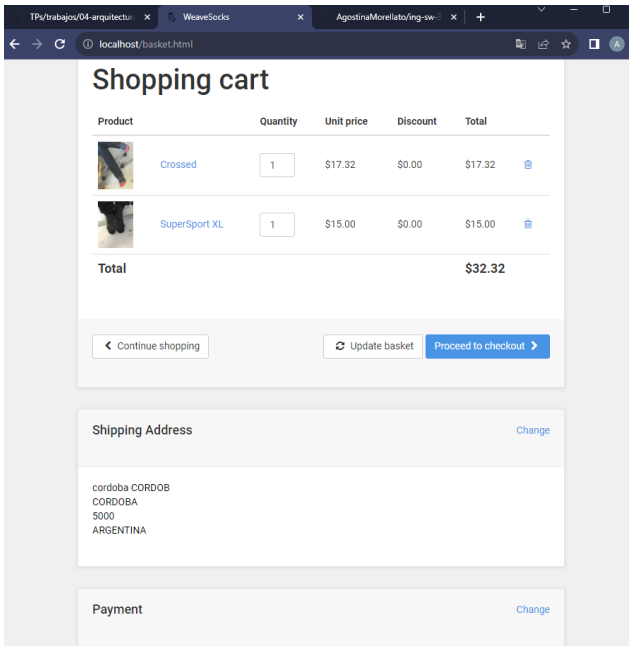
Generar un usuario



Realizar búsquedas por tipo de media, color, etc.



Hacer una compra - poner datos falsos de tarjeta de crédito ;)



2- Investigación de los componentes

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

```
PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-demo\microservices-demo> docker ps
```

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
9a3e737357fd	weaveworksdemos/front-end:0.3.12	docker-compose-front-end-1	"/usr/local/bin/npm ..."	24 minutes ago	Up 24 minutes	8079/tcp
20188e783212	weaveworksdemos/shipping:0.4.8	docker-compose-shipping-1	"/usr/local/bin/java..."	24 minutes ago	Up 24 minutes	
9eebcbaee59d	weaveworksdemos/catalogue:0.3.5	docker-compose-catalogue-1	"/app -port=80"	24 minutes ago	Up 24 minutes	80/tcp
2d3614b48cc2	weaveworksdemos/orders:0.4.7	docker-compose-orders-1	"/usr/local/bin/java..."	24 minutes ago	Up 24 minutes	
82a9b3583891	weaveworksdemos/user-db:0.4.0	docker-compose-user-db-1	"/entrypoint.sh mong..."	24 minutes ago	Up 24 minutes	27017/tcp
8a1fcd5dc01f	mongo:3.4	docker-compose-carts-db-1	"docker-entrypoint.s..."	24 minutes ago	Up 24 minutes	27017/tcp
6bb2ff349680	mongo:3.4	docker-compose-orders-db-1	"docker-entrypoint.s..."	24 minutes ago	Up 24 minutes	27017/tcp
3b256c678544	weaveworksdemos/edge-router:0.1.1	docker-compose-edge-router-1	"traefik"	24 minutes ago	Up 24 minutes	0.0.0.0:80->8080/tcp, 0.0.0.0:8080->8080/tcp
b4c155b5cc91	weaveworksdemos/queue-master:0.3.1	docker-compose-queue-master-1	"/usr/local/bin/java..."	24 minutes ago	Up 24 minutes	
260f69d8e440	weaveworksdemos/catalogue-db:0.3.0	docker-compose-catalogue-db-1	"docker-entrypoint.s..."	24 minutes ago	Up 24 minutes	3306/tcp
ccbc94a2e110	weaveworksdemos/payment:0.4.3	docker-compose-payment-1	"/app -port=80"	24 minutes ago	Up 24 minutes	80/tcp
b3e8f1c46eee	weaveworksdemos/user:0.4.4	docker-compose-user-1	"/user -port=80"	24 minutes ago	Up 24 minutes	80/tcp
c135712c63f1	rabbitmq:3.6.8	docker-compose-rabbitmq-1	"docker-entrypoint.s..."	24 minutes ago	Up 24 minutes	4369/tcp,
5671-5672/tcp, 8beca3042e15	weaveworksdemos/carts:0.4.8	docker-compose-carts-1	"/usr/local/bin/java..."	24 minutes ago	Up 24 minutes	

Contenedores:

- front-end: Ejecuta la aplicación de front-end, este contenedor ejecuta la aplicación de front-end utilizando el comando npm start.
- edge-router: Enrutador, se encarga de la gestión del tráfico en los puertos 80 y 8080.
- catalogue: Ejecuta la sección de catálogo en la aplicación web y como el contenedor "front-end". El punto de entrada de esta aplicación es /app con el argumento -port=80.
- catalogue-db: Ejecuta una base de datos MySQL para ser utilizada por la aplicación de catálogo. Crea la base de datos llamada "socksdb". Se inicia el servidor de base de datos MySQL con el comando "mysqld".
- carts: Ejecuta la aplicación que gestiona los carritos de compra. Especifica configuraciones como las opciones de memoria y seguridad. Como punto de entrada tiene el servicio de carritos de compra.
- carts-db: Ejecuta una base de datos MongoDB para ser la aplicación de carritos de compra. Inicia el servidor MongoDB con el comando mongod.
- orders: Ejecuta la aplicación de ordenes, una vez completada la compra. Como punto de entrada tiene el servicio de ordenes de compra.

- orders-db: Este contenedor ejecuta una base de datos MongoDB para ser utilizada por la aplicación de ordenes.
- shipping: Ejecuta una aplicación para el envío de ordenes. Como punto de entrada tiene el servicio de envío de compras.
- queue-master: Este contenedor actúa como un maestro de cola y se comunica con otros contenedores a través del socket de Docker. Se establecen ciertas configuraciones de seguridad.
- rabbitmq: Este contenedor ejecuta el servidor de cola RabbitMQ, que se utiliza para la comunicación y la gestión de colas. Este contenedor ejecuta el servidor RabbitMQ con el comando rabbitmq-server.
- payment: Ejecuta una aplicación de pagos. El punto de entrada de esta aplicación es /app con el argumento -port=80.
- user: Este contenedor ejecuta una aplicación de usuario y se comunica con la base de datos MongoDB "user-db". El punto de entrada de esta aplicación es /user y se usa con el argumento -port=80.
- user-db: El contenedor "user-db" ejecuta una base de datos MongoDB para ser utilizada por el servicio "user". Inicia el servidor MongoDB con la configuración ubicada en /etc/mongodb.conf y la opción --smallfiles.
- user-sim: Ejecuta una herramienta para simular cargas de usuarios. Se ejecuta con un comando específico para controlar el comportamiento de la simulación.

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

```
PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-demo> git clone https://github.com/microservices-demo/front-end.git
Cloning into 'front-end'...
remote: Enumerating objects: 1236, done.
remote: Total 1236 (delta 0), reused 0 (delta 0), pack-reused 1236
Receiving objects: 92% (1138/1236), 45.68 MiB | 5.47 MiB/s
Receiving objects: 93% (1150/1236), 45.68 MiB | 5.47 MiB/s
Receiving objects: 100% (1236/1236), 47.90 MiB | 5.35 MiB/s, done.
Resolving deltas: 100% (687/687), done.
PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-demo> git clone https://github.com/microservices-demo/user.git
Cloning into 'user'...
remote: Enumerating objects: 1063, done.
remote: Total 1063 (delta 0), reused 0 (delta 0), pack-reused 1063
Receiving objects: 92% (978/1063), 172.83 KiB | 921.00 KiB/s, done.
Receiving objects: 100% (1063/1063), 172.83 KiB | 921.00 KiB/s, done.
Resolving deltas: 100% (601/601), done.
PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-demo> git clone https://github.com/microservices-demo/edge-router.git
Cloning into 'edge-router'...
remote: Enumerating objects: 50, done.
remote: Total 50 (delta 0), reused 0 (delta 0), pack-reused 50
Receiving objects: 54% (27/50), 14.51 KiB | 353.00 KiB/s, done.
Receiving objects: 100% (50/50), 14.51 KiB | 353.00 KiB/s, done.
Resolving deltas: 100% (12/12), done.
PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-demo> |
```

¿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. La separación de repositorios para el código y la configuración del sistema puede tener ventajas y desventajas, y la elección de adoptar esta práctica depende de distintos factores, incluyendo la estructura del proyecto, las necesidades específicas y las preferencias del equipo de desarrollo.

Ventajas:

- Organización: Mantener el código y la configuración del sistema en repositorios separados puede hacer que el proyecto sea más organizado, se puede identificar claramente dónde se encuentra cada componente y qué cambios se realizan en cada repositorio.
- Reutilización: La configuración del sistema, como archivos de Docker Compose, se puede reutilizar en múltiples proyectos. Al mantenerlos en un repositorio independiente, es más fácil compartir y reutilizar estas configuraciones en diferentes proyectos.
- Control de acceso: La separación de repositorios permite un mejor control de acceso. Los equipos pueden otorgar permisos de lectura y escritura de manera más específica según las necesidades.
- Escalabilidad: Cuando el proyecto crece, la separación de código y configuración puede facilitar la escalabilidad. Los equipos pueden trabajar en diferentes aspectos del proyecto sin interferir entre sí.

Desventajas de utilizar repositorios separados:

- Complejidad inicial: La separación de repositorios puede aumentar la complejidad inicial del proyecto. Los desarrolladores deben gestionar múltiples repositorios, lo que podría requerir más tiempo y esfuerzo en la configuración inicial.
- Más trabajo administrativo: Mantener varios repositorios implica más tareas de administración, como la gestión de ramas, etiquetas y flujos de trabajo. Esto puede aumentar la carga de trabajo para los equipos.
- Dificultades en la colaboración: Si los equipos de desarrollo necesitan realizar cambios tanto en el código como en la configuración del sistema en paralelo, la separación de repositorios puede dificultar la colaboración efectiva.

¿Cuál contenedor hace las veces de API Gateway?

El contenedor que actúa como API Gateway es el contenedor "edge-router". Se encarga de dirigir y administrar el tráfico de red dentro del sistema, y expone los puertos 80 y 8080 para redirigir las solicitudes entrantes a los

servicios correspondientes. Funciona como un punto de entrada único para el tráfico entrante y es responsable de enrutar las solicitudes a los servicios, lo que lo convierte en un componente esencial de la arquitectura de microservicios y un tipo de API Gateway.

Cuando ejecuto este comando:

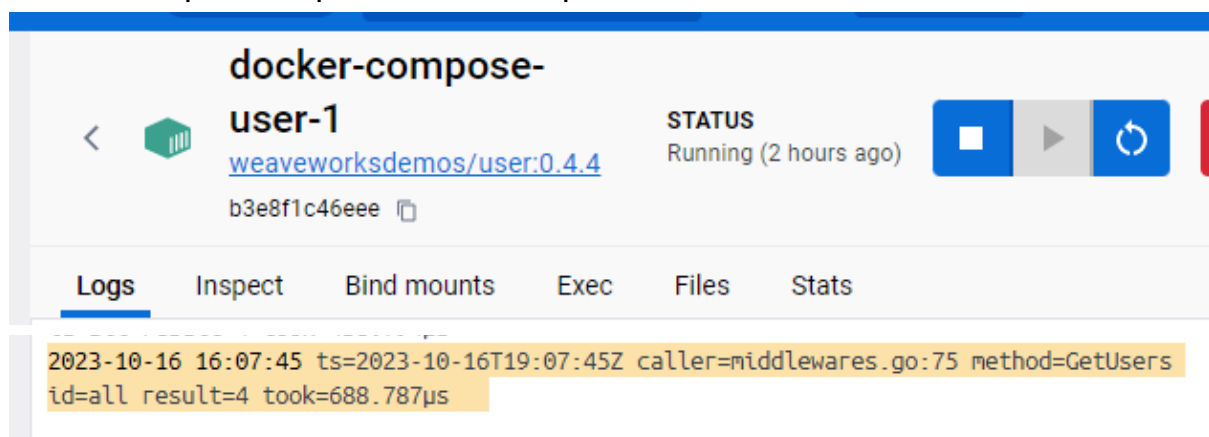
curl <http://localhost/customers>

```
PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-demo> curl http://localhost/customers

StatusCode      : 200
StatusDescription : OK
Content         : {"_embedded":{"customer":[{"firstName":"Eve","lastName":"Berger","username":"Eve_Berger","id":"57a98d98e4b00679b4a830af","_links":{"addresses":{"href":"http://user/customers/57a98d98e4b00679b4a830af/a...
RawContent      : HTTP/1.1 200 OK
                  Date: Mon, 16 Oct 2023 18:53:25 GMT
                  Set-Cookie: _TRAEFIK_BACKEND=http://front-end:8079,md.sid=s%3AzKCitxPZ8WiYrA8n0HoM9Iv5uCOenRJ3.NdRjdtv9M2t9NER1b%2BvZ7k%2BIpEC8K8c33JsAKE80yH8; Pa...
Forms           : {}
Headers         : {[Date, Mon, 16 Oct 2023 18:53:25 GMT], [Set-Cookie, _TRAEFIK_BACKEND=http://front-end:8079,md.sid=s%3AzKCitxPZ8WiYrA8n0HoM9Iv5uCOenRJ3.NdRjdtv9M2t9NER1b%2BvZ7k%2BIpEC8K8c33JsAKE80yH8; 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 esta procesando la operación es user



The screenshot shows the Docker Desktop interface for a container named 'user-1'. The container is running, as indicated by the 'STATUS' section which says 'Running (2 hours ago)'. Below the status, there are tabs for 'Logs', 'Inspect', 'Bind mounts', 'Exec', 'Files', and 'Stats'. The 'Logs' tab is selected, showing a log entry: '2023-10-16 16:07:45 ts=2023-10-16T19:07:45Z caller=middlewares.go:75 method=GetUsers id=all result=4 took=688.787µs'.

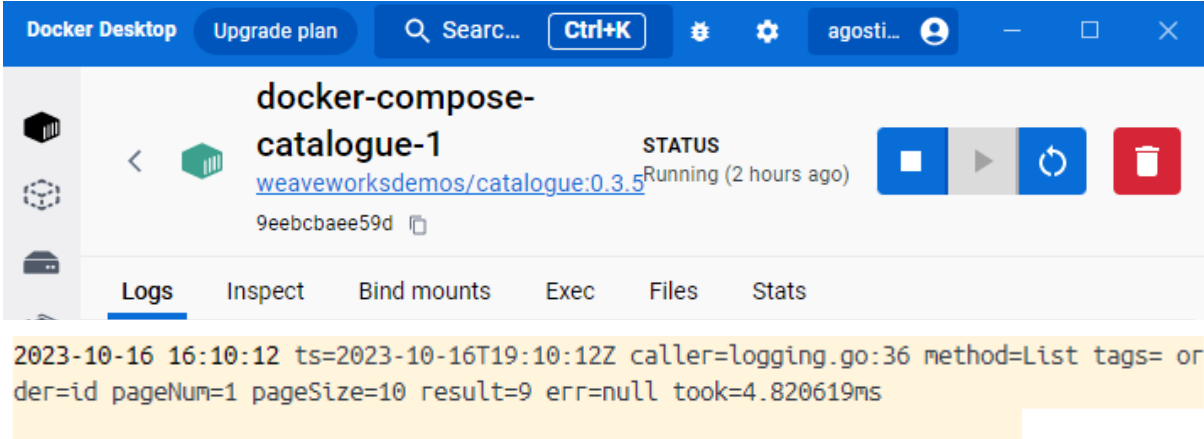
¿Y para los siguientes casos?

curl <http://localhost/catalogue>


```
PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-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: Mon, 16 Oct 2023 18:53:54 GMT
                  Set-Cookie: _TRAFFIK_BACKEND=http://front-end:8079,md.sid=s%3AeNqD3TB_T0x9cDV-gT-OxrTHg3I0ZWc1.
                  MCNeUEUCLChFI55xvZ0QD8P96epiQC2idSxJuQvELz4; Path=/...
Forms           : {}
Headers         : [{"Date, Mon, 16 Oct 2023 18:53:54 GMT}, [Set-Cookie, _TRAFFIK_BACKEND=http://front-end:8079,md.
                  sid=s%3AeNqD3TB_T0x9cDV-gT-OxrTHg3I0ZWc1.MCNeUEUCLChFI55xvZ0QD8P96epiQC2idSxJuQvELz4; 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 catalogue



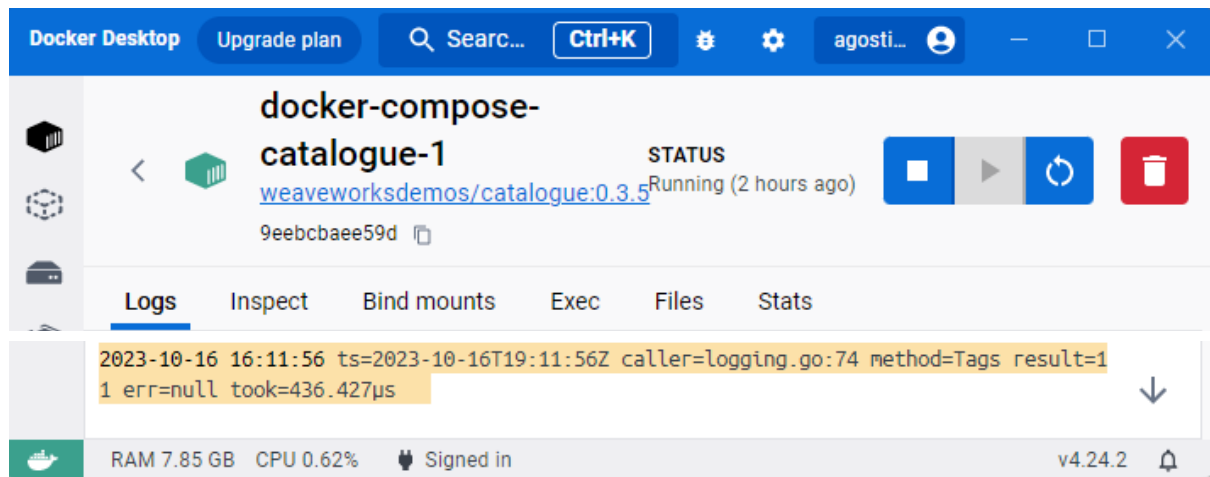
The screenshot shows the Docker Desktop interface. At the top, there's a navigation bar with 'Docker Desktop', 'Upgrade plan', a search bar, and a user profile 'agosti...'. Below this, the main area shows a service named 'docker-compose catalogue-1' with a status of 'Running (2 hours ago)'. There are buttons for stopping, starting, refreshing, and deleting the service. Below the service name, there's a tabbed interface with 'Logs', 'Inspect', 'Bind mounts', 'Exec', 'Files', and 'Stats'. The 'Logs' tab is selected, showing a log entry: '2023-10-16 16:10:12 ts=2023-10-16T19:10:12Z caller=logging.go:36 method=List tags= or der=id pageNum=1 pageSize=10 result=9 err=null took=4.820619ms'.

curl <http://localhost/tags>

```
PS C:\Users\Usuario\OneDrive\Escritorio\IngSwIII\Tp_4\socks-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: Mon, 16 Oct 2023 18:54:02 GMT
                  Set-Cookie: _TRAFFIK_BACKEND=http://front-end:8079,md.sid=s%3AwzEc6gRyU05DCA-v93C338KG85MyN3kl.
                  d%2BVAyOZX0zVygRd2YJYTr8QpIZRsB37MYvTgy%2FV4o8; Pa...
Forms           : {}
Headers         : [{"Date, Mon, 16 Oct 2023 18:54:02 GMT}, [Set-Cookie, _TRAFFIK_BACKEND=http://front-end:8079,md.
                  sid=s%3AwzEc6gRyU05DCA-v93C338KG85MyN3kl.d%2BVAyOZX0zVygRd2YJYTr8QpIZRsB37MYvTgy%2FV4o8;
                  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 catalogue



¿Cómo persisten los datos los servicios?

Utilizando volúmenes, son una forma de almacenar y administrar datos de manera persistente fuera del sistema de archivos del contenedor. Cuando el volumen está montado en el contenedor, cualquier dato almacenado en la ruta especificada en el contenedor se almacenará en el volumen. Esto significa que los datos persisten incluso si se detiene o elimina el contenedor. Puedes acceder a los datos en el volumen desde otros contenedores que también montan ese volumen.

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

El componente encargado del procesamiento de la cola de mensajes es el contenedor rabbitmq que corresponde a un servidor RabbitMQ, un sistema de mensajería de código abierto que actúa como un intermediario para el intercambio de mensajes, se utiliza comúnmente en arquitecturas de microservicios para implementar colas de mensajes y patrones de comunicación asincrónica.

¿Qué tipo de interfaz utilizan estos microservicios para comunicarse?

En esta arquitectura de microservicios, estos se comunican a través de protocolos basados en HTTP. El protocolo HTTP es ampliamente utilizado en arquitecturas de microservicios debido a su simplicidad y su capacidad para funcionar en la mayoría de las redes y entornos. Los microservicios exponen API (Interfaz de Programación de Aplicaciones) basadas en HTTP para permitir la comunicación entre sí.