Sistemas Distribuidos

Proyecto 2: Consorcio

Diseño, desarrollo y despliegue de un portal de noticias distribuido

Integrantes

Claudio Solis
Micaela Pintos
Juan Huicha
Miguel Bustamante
Facundo Bernardini

Objetivos

El objetivo del proyecto es realizar un consorcio el cual brinda diferentes funciones relacionadas a las noticias. Estas funciones son:

- Administrar la suscripción de los clientes a un servicio de noticias.
- Enviar las nuevas noticias a todos los clientes inscriptos en la sección.
- Recibir una nueva noticia.
- Borrar una noticia, solamente puede realizar está operación el cliente que la envió como nueva noticia.
- Administrar el área de noticias, esto significa agregar y eliminar un área.
- Informar sobre las noticias de las últimas 24hs.

Luego cada cliente puede realizar las siguientes operaciones:

- Suscribirse en cualquiera de los servicios de noticias. (Deportivas, Políticas, Sociales, del Mundo, Económicas)
- Recibir todas las noticias que son enviadas en las cuales está suscrito.
- Borrarse de una suscripción.
- Enviar noticias a una sección en la cual es miembro.

Tecnologías

Las tecnologías acordadas en el equipo fueron las siguientes:

- Para la comunicación entre servicios se utilizó gRPC, ya que es una nueva tecnología explicada en la clase y muy utilizada. Para poder aprenderla mejor, se decidió implementarla.
- Como orquestación de contenedores, al principio en el modelo, se había llegado a la conclusión de utilizar Docker Compose pero ante las limitaciones del mismo y la utilidad de Docker Swarm, se decidió utilizar esté último.
- Se eligió MySQL para poder almacenar la información, tanto los clientes como las noticias.
- El lenguaje de programación que se utilizó para los servicios es Python, ya que nos resultó práctico la cantidad de librerías que implementan para realizar esté estilo de proyectos.

Arquitectura general del sistema

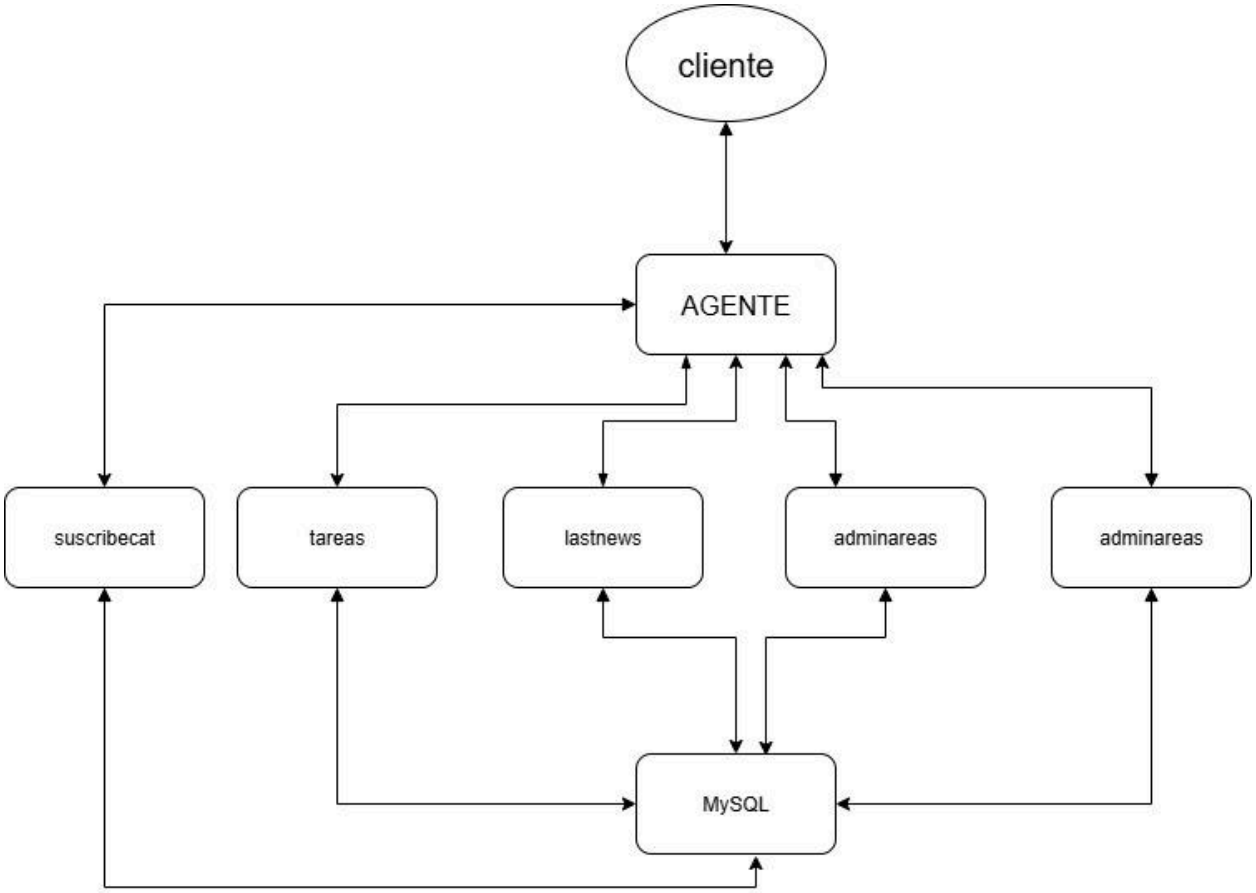

Figura 1: Arquitectura planteada

En la figura 1 se puede visualizar la arquitectura planteada para el proyecto en el modelo pero la implementación que se llevó a cabo, no difiere tanto de la misma. Se decidió seguir con la idea pero se le ha cambiado el nombre al middleware, ya que se decidió llamarlo "*Agente*", el cual cumple la misma función que el middleware de la figura 1.

Funcionalidad

El servicio Agente dentro del clúster Swarm se encarga de manejar las operaciones solicitadas por los clientes relacionadas con la gestión de noticias y suscripciones a categorías. El servicio implementa las siguientes responsabilidades:

- Login: Verifica las credenciales del cliente (DNI y contraseña) y retorna el resultado del inicio de sesión, utiliza servicios que brinda el miembro **tareas.**
- ObtenerNoticiasUltimas24hs: Retorna las noticias publicadas en las últimas 24 horas para un usuario autenticado, utiliza servicios del miembro **lastnews**

- ObtenerUltimasNoticias: Permite obtener las últimas noticias disponibles para una categoría específica, utiliza servicios del miembro **tareas.**
- AgregarCategoria: Habilita al cliente a registrar una nueva categoría en el sistema, utiliza servicios del miembro **adminareas**
- SuscribirNuevaCategoria: Suscribe al cliente a una nueva categoría de interés, utiliza servicios del miembro **suscribecat.**
- BorrarSuscripcionCategoria: Elimina la suscripción de un cliente a una categoría determinado, utiliza servicios del miemrbo **tareas.**
- BorrarArea: Permite al cliente eliminar una categoría previamente registrada, utiliza servicios del miembro **adminareas.**
- VerCategoriasInscripto: Muestra todas las categorías a las que el cliente está actualmente suscripto, utiliza servicios del miembro **adminareas.**
- EnviarNoticia: Permite al cliente enviar una nueva noticia, indicando título, contenido y categoría de destino, utiliza servicios del miembro **enviarnoticia.**

Diseño de la base de datos

Cuando se debatió el modelo, se decidió realizar una base de datos como se muestra en la figura 2, ya que contempla todas las operaciones que brinda el consorcio. Esta es accedida por medio de los miembros, ya que son ellos los que procesan y solicitan una solicitud particular del cliente.

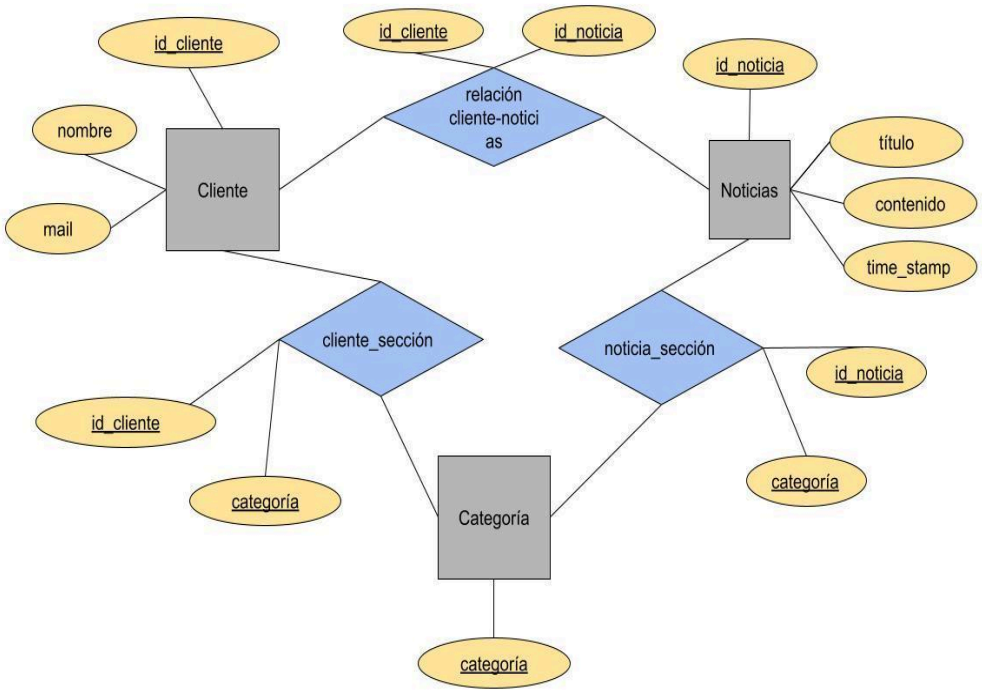

Figura 2: Modelo entidad-relación de la base de datos.

Elección de imagen Python

En un principio se utilizó la imagen python:3.10-slim, con la cual se obtenía una imagen de servicios con un tamaño aproximado de 390 MB. Esta variante slim contiene los paquetes mínimos de Debian necesarios para ejecutar Python.

Luego se verificó el funcionamiento con la imagen python:3.10-alpine, basada en Alpine Linux. Con esta imagen se obtuvo un tamaño de 156 MB. Al no encontrarse diferencias en el funcionamiento, se optó por la imagen Alpine para optimizar el uso de recursos.

También se consideró la construcción de imágenes en etapas (multi-stage builds), pero no se logró obtener una imagen funcional, ni se observó una reducción significativa en su tamaño.

Despliegue en swarm

Desde el nodo que actuará como máster del clúster, se inicializa el entorno de Swarm ejecutando el siguiente comando:

```
docker swarm init --advertise-addr <ip_master>
```

Este comando crea el clúster Swarm y genera un token de unión, el cual debe ser ejecutado en cada uno de los nodos trabajadores para que se integren al clúster.

Para el despliegue de los servicios en el entorno Swarm, es necesario distribuir las imágenes Docker utilizadas por los distintos contenedores (por ejemplo, miembros y agentes). Para ello, se debe exportar cada imagen desde el nodo donde fue construida, utilizando el siguiente comando:

```
docker save -o <nombre_archivo.tar> <nombre_imagen>
```

Luego, en cada uno de los nodos que forman parte del Swarm, se debe cargar dicha imagen con:

```
docker load -i <ruta_archivo.tar>
```

De esta manera, todos los nodos disponen localmente de las imágenes necesarias para ejecutar los servicios correspondientes.
Una vez que todos los nodos del clúster Swarm tienen las imágenes necesarias cargadas, se procede a iniciar el servicio desde el nodo máster. Para ello, se utiliza un archivo **docker-compose.yml** con la definición del stack y se ejecuta el siguiente comando:

```
docker stack deploy -c docker-compose.yml consorcio_stack
```

Este comando despliega el stack llamado consorcio_stack utilizando las configuraciones definidas en el archivo **docker-compose.yml.**
El archivo docker-compose.yml está configurado para cargar un script de inicialización SQL (init.sql) en el contenedor de MySQL mediante el uso de la instrucción configs, propia del modo Swarm.
Por ello, es fundamental que en el nodo máster, desde donde se ejecuta docker stack deploy, exista la siguiente estructura de carpetas y archivos:

```
.
├── docker-compose.yml
└── base-de-datos/
    └── init.sql
```

A continuación se muestra el fragmento de configuración relacionado con MySQL y el script de inicialización:

```
version: "3.8"

configs:
  init-sql:
    file: ./base-de-datos/init.sql

services:
  mysql:
    image: mysql:8.4
    environment:
      MYSQL_ROOT_PASSWORD: admin
      MYSQL_DATABASE: consorcio
    configs:
      - source: init-sql
        target: /docker-entrypoint-initdb.d/init.sql
```

- configs: esta sección define archivos de configuración que se pueden montar en los contenedores del stack. En este caso, se especifica el archivo init.sql ubicado en ./base-de-datos/init.sql.
- En el servicio mysql, se utiliza el bloque configs para montar este archivo en la ruta /docker-entrypoint-initdb.d/init.sql dentro del contenedor.
- MySQL detecta automáticamente los archivos .sql ubicados en esa carpeta durante el primer arranque del contenedor, y los ejecuta para inicializar la base de datos.

Para la ejecución del cliente se debe realizar dentro de la carpeta “cliente” el comando

```
python cliente.py
```