Proyecto Consorcios

Sistemas Distribuidos 2023
Departamento de Ciencias e Ingeniería de la Computación

Integrantes:

Galiana, Albano.
Iurman, Franco.
Mazzarello, Nicolás.
Miguel, Agustín.

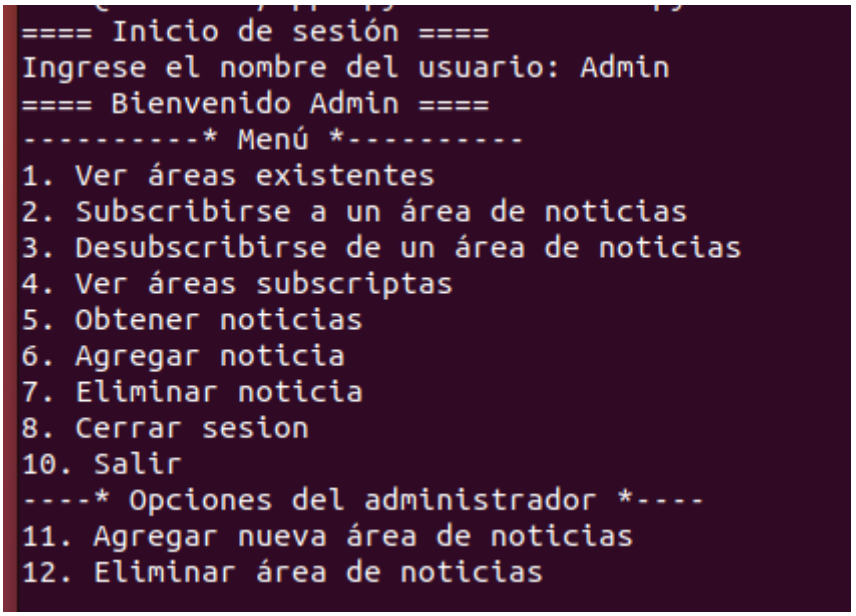
Índice

Introducción

Este proyecto busca aplicar varios de los conceptos aprendidos en el transcurso de la materia Sistemas Distribuidos, tales como comunicación directa e indirecta, sincronización y la utilización de Docker para la orquestación, a un sistema de gestión de usuarios y noticias con modelo de suscripción.

Requerimientos Funcionales

- Ver todas las áreas de noticias disponibles.
- Aceptar la suscripción de un nuevo cliente.
- Aceptar la finalización de la suscripción por parte del cliente.
- Mostrar las áreas suscriptas.
- Entregar las noticias de un área especificada, con posibilidad de entregar todas las noticias o solamente las noticias no leídas.
- Agregar una noticia a un área especificada.
- Eliminar una noticia, solamente puede realizar esta operación el cliente que la envió como nueva noticia.
- Cerrar sesión.
- El administrador puede agregar una nueva área de noticias.
- El administrador puede eliminar un área de noticias.



Menú de Usuario Administrador

Requerimientos No Funcionales

Los nodos deben poder crearse para otorgar mayor capacidad de carga y en caso de que uno falle, otro pueda reemplazarlo y el sistema siga funcionando.

Toda la información debe ser accesible y este acceso debe ser transparente para el cliente sin importar en qué nodo se encuentre conectado.

Las conexiones deben ser cortas.

El cliente se comunica con el servicio de noticias, y este mismo se encarga de seleccionar el nodo correspondiente según la solicitud del cliente, luego le entrega al cliente el nombre del nodo (comisión) correspondiente. Luego, el cliente se comunica con el nodo y este le devuelve la respuesta de la solicitud.

La comunicación entre cliente, nodo y servidor es a través de sockets.

Piezas de software

- Agente: Encargado de realizar la comunicación inicial con el cliente. Se encarga del registro e inicio de sesión de los usuarios, si estos son administradores además otorga la posibilidad de agregar o eliminar áreas.
  Por otro lado, se encarga de manejar a nivel aplicación la distribución de las áreas entre los nodos, devolviendo al usuario el nodo al cual solicitar una operación.
  Periódicamente realiza un healthcheck de los nodos registrados, para eliminar aquellos que se desconecten.
  Este agente puede replicarse para permitir la consulta de más usuarios en simultáneo.

- Nodo: Realizan las operaciones referidas a noticias, tales como:
  Agregar Noticias
  Eliminar Noticias
  Suscribir Usuarios a un Área
  Desuscribir Usuarios a un Area

  Los nodos pueden replicarse para permitir soportar una mayor capacidad de solicitudes, en especial para áreas más demandadas.

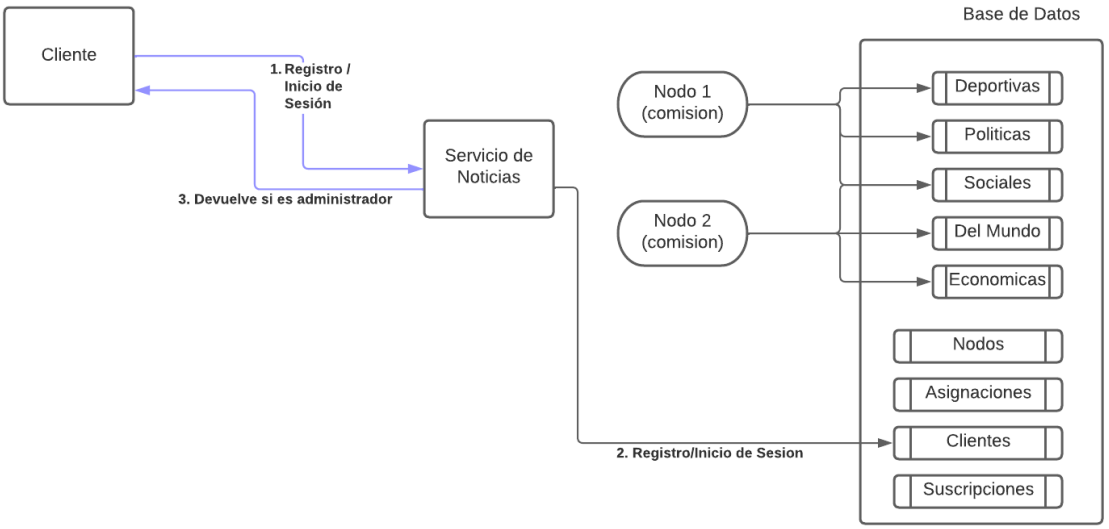
- Cliente: Encargado de los requerimientos funcionales de la aplicación y con quién tendrá acceso el usuario final. Solicitará al usuario su nombre para poder luego obtener su información, como áreas suscritas y si es administrador, además de realizar las operaciones a su nombre.
  Cuenta con un menú para permitir realizar las funciones del sistema. En caso de ser administrador, también permitirá al usuario agregar y eliminar áreas.

- Base de Datos Postgres: Encargada de persistir toda la información de la aplicación.

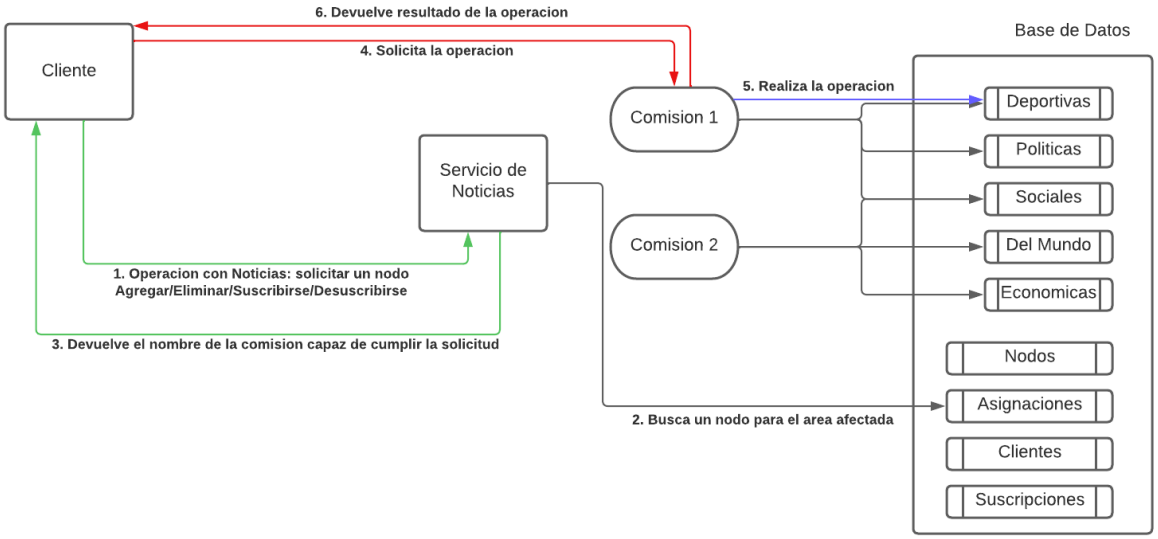
  Almacena:
  - Usuarios
  - Noticias
  - Comisiones
  - Áreas
  - Suscripciones de Usuarios a Áreas
  - Asignaciones de Áreas a Comisiones

Diagramas de interacción

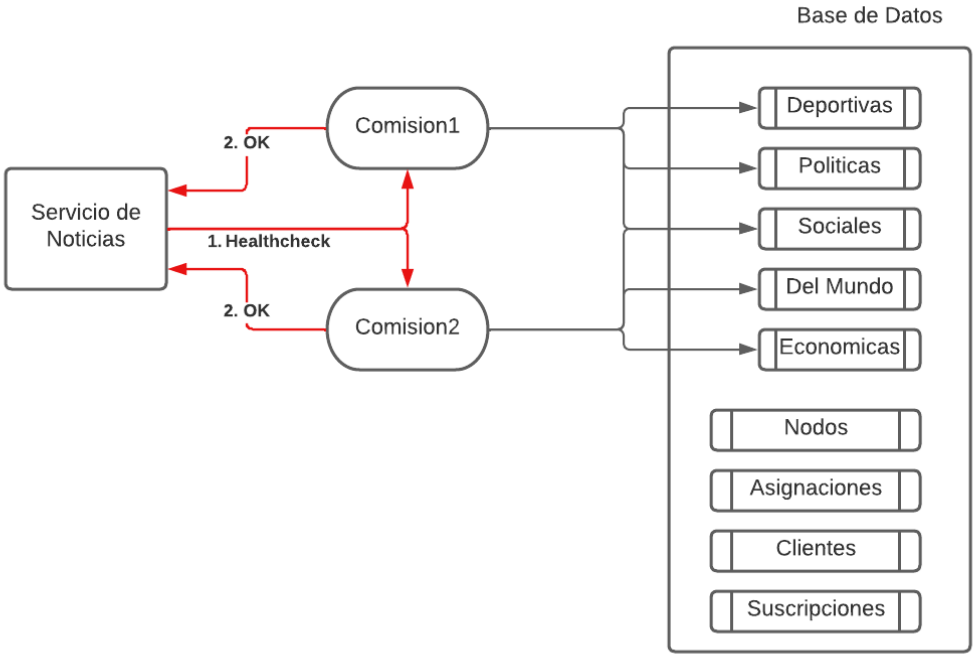
Registro o Inicio de Sesión, del cliente:



Operaciones Suscribirse/Desuscribirse/Agregar/Eliminar, del Cliente:



Chequeo de salud, del Servicio de Noticias al buscar la comisión:



Consideraciones de Diseño

Utilizamos Python como lenguaje de programación para las aplicaciones gracias a su característica de no-tipado, ya que es un lenguaje moderno y permite rápidamente abstraerse de la implementación de sockets, hilos y distintas llamadas al sistema. Además su contenerización es rápida y sencilla.

Despliegue

Cada aplicación está pensada para existir en un container Docker y cuenta con un Dockerfile, que guarda las instrucciones de imágenes base y como crear la imagen de la aplicación que será levantada en un container, a la vez que guarda ciertas variables de entorno con información requerida por las aplicaciones, como nombres de host de las otras aplicaciones.

Los Dockerfile de Nodo y Agente se basan en la imagen oficial de python y cuentan solo con su compilador y la posibilidad de ejecutarlo, dejándonos así imágenes muy livianas de aproximadamente 60mb. Solo se le agregan las variables de entorno consumidas por los programas en Python.

El Dockerfile de Database estaba basado en la imagen oficial de Postgres, con las variables de entorno para configurar el acceso, un healthcheck y lo mas importante, copiar al contexto de este un archivo .sql para crear la base de datos a utilizar por el sistema.

El Dockerfile de Cliente está basado en una imagen de Debian al cual se le agregan SSH para conectarse y Python para poder ejecutar el cliente. Tomamos esta decisión ya que para conectarnos al Agente y a las Comisiones debemos estar dentro de la red de Docker, y nuestra aplicación al ser un cliente de consola y utilizar el standard input, requiere de bash ejecutándose para que podamos interactuar.

Creamos a su vez un archivo de Docker Compose, permitiendo desplegar el sistema localmente de manera rápida y sencilla, tanto las aplicaciones como la base de datos.

Por último, para permitir la implementación en un cluster o sistema distribuido, utilizamos Docker Swarm, dónde creando una red y servicios asociados a esta, permite de manera transparente que las aplicaciones se conecten entre sí y a la base de datos.

Para realizar el deploy en distintos nodos, utilizamos Docker Hub para almacenar las imágenes.

Todo el código esta subido en el siguiente repositorio, y gracias a GitHub Actions, el proceso de creacion de las imagenes y su subida a Docker Hub se encuentra automatizado.

https://github.com/nicomazzarello/Distributed-Systems-Final-Project/

Creamos las imágenes a desplegar
sudo docker build Cliente/Dockerfile - -tag nicomazzarello/proyecto-distribuidos:cliente
sudo docker build Agente/Dockerfile - -tag nicomazzarello/proyecto-distribuidos:agente
sudo docker build Nodo/Dockerfile1 - -tag nicomazzarello/proyecto-distribuidos:comision1
sudo docker build Nodo/Dockerfile2 - -tag nicomazzarello/proyecto-distribuidos:comision2
sudo docker build Database/Dockerfile - -tag nicomazzarello/proyecto-distribuidos:database

Subimos las imágenes a DockerHub (Requiere haber iniciado sesión con docker login)
sudo docker push nicomazzarello/proyecto-distribuidos:comision1
sudo docker push nicomazzarello/proyecto-distribuidos:comision2
sudo docker push nicomazzarello/proyecto-distribuidos:agente
sudo docker push nicomazzarello/proyecto-distribuidos:nodo
sudo docker push nicomazzarello/proyecto-distribuidos:database

Creamos el cluster de Docker Swarm en el nodo que será maestro
sudo docker swarm init --advertise-addr 192.168.99.100 # IP donde escuchará

En cada nodo trabajador, lo unimos al Swarm
sudo docker swarm join - -{TOKEN} 192.168.99.100:2377 # El Token será dado por el nodo maestro y así también la IP y Puerto donde estará escuchando

Traer imágenes a los nodos (Este paso puede obviarse ya que se utiliza Docker Hub)
sudo docker pull nicomazzarello/proyecto-distribuidos:agente
sudo docker pull nicomazzarello/proyecto-distribuidos:comision1
sudo docker pull nicomazzarello/proyecto-distribuidos:comision2
sudo docker pull nicomazzarello/proyecto-distribuidos:postgres
sudo docker pull nicomazzarello/proyecto-distribuidos:cliente

Desde el nodo principal
Crear Red
sudo docker network create --driver overlay proyectonet

Crear servicios
sudo docker service create --name database --hostname database --network proyectonet --publish 5432:5432 nicomazzarello/proyecto-distribuidos:postgres
sudo docker service create --name agente --hostname agente --publish 8000:8000 --network proyectonet nicomazzarello/proyecto-distribuidos:agente
sudo docker service create --replicas 3 --name comision1 --hostname comision1 --publish 8005:8005 --network proyectonet nicomazzarello/proyecto-distribuidos:comision1
sudo docker service create --replicas 3 --name comision2 --hostname comision2 --publish 8006:8005 --network proyectonet nicomazzarello/proyecto-distribuidos:comision2
sudo docker service create --name cliente --hostname cliente --publish 22:22 --network proyectonet nicomazzarello/proyecto-distribuidos:cliente

Escalar Servicio de Comisiones
sudo docker service scale comision(1/2) = 5

Conclusiones

A la hora de armar la arquitectura de un sistema distribuido, con todos los beneficios que este otorga, deben tomarse en cuenta muy seriamente las complejidades que agrega.

Desde problemas de comunicación y sincronización, donde debe ser transparente la ubicación de los distintos servicios, hasta posibles condiciones de carrera y formas de disminuir los puntos de fallos del sistema.

Agregar servicios que dependan de otros, nos obliga a tener recaudos a la hora de que alguno de estos falle y produzca el colapso del sistema.

La comunicación, si bien a velocidad humana son rápidas, debemos tener en cuenta que es varias órdenes de magnitud más lenta que una ejecución local y esto puede llegar a provocar problemas de sincronización y condiciones de carrera, sobre todo si varios servicios acceden de manera concurrente.

Es por todo esto que a la hora de desarrollar un software distribuido debemos tener una perspectiva diferente a la del clásico programa monolítico que se ejecuta en un servidor.

Con esto, ganamos escalabilidad, permitiéndonos soportar mayor cantidad de solicitudes. Otra de las mejoras es la redundancia, ya que, por ejemplo, teniendo dos agentes corriendo en dos nodos físicos distintos, nos permite asegurar que la caída de uno no afecte al resto del sistema. Lo mismo con los nodos que acceden a la base de datos, permiten darle redundancia al acceso a los datos.
Por último, destacamos que todo el proceso de despliegue podría automatizarse permitiendo lanzar el sistema y actualizaciones de una manera mucho más sencilla.