INFORME TÉCNICO RED P2P (SISTEMAS DISTRIBUIDOS).

(https://github.com/thomasvanegas/REST_API_P2P_Architecture/**).**

JHON DANIEL GAVIRIA GARCIA

THOMAS CAMILO VANEGAS ACEVEDO

23/09/2025

SISTEMAS DISTRIBUIDOS Y ARQUITECTURAS DE NUBE.

OBJETIVO.

Diseñar e implementar una red P2P no estructurada basada en un servidor de directorio y localización de archivos, en la cual cada nodo integre microservicios que permitan compartir archivos entre nodos haciendo uso del protocolo de comunicación gRPC.

MARCO TEORICO.

Las redes peer-to-peer son arquitecturas distribuidas en las que cada nodo de la red, denominado peer, puede actuar simultáneamente como cliente y servidor. Esto elimina la dependencia de un nodo central para la transferencia de datos y permite la descentralización en la compartición de recursos, siendo ampliamente utilizado en sistemas de compartición de archivos y aplicaciones colaborativas.

En este contexto, un peer es un nodo participante que puede proveer y consumir recursos. Sin embargo, las redes p2p tienen problemas localización de archivos. Para solucionar este problema, se suele incorporar un servidor de directorio y localización, encargado de mantener un índice actualizado de los archivos disponibles en la red y los peers que los contienen.

El diseño del sistema se apoya en la arquitectura de microservicios, donde cada funcionalidad se implementa como un servicio independiente, capaz de escalar y comunicarse con otros servicios. Estos microservicios pueden ejecutarse en distintos nodos o procesos y comunicarse entre sí utilizando protocolos de comunicación estandarizados como HTTP y gRCP. un aspecto clave en este tipo de sistemas es la concurrencia, entendida como la capacidad de los microservicios para atender múltiples solicitudes de procesos remotos de manera simultánea.

En cuanto a la comunicación entre nodos. Por un lado, esta HTTP se utiliza para la autenticación de los nodos. De esta manera, se garantiza que cada peer pueda identificarse ante el sistema antes de participar en el intercambio de archivos. Por otro lado, se implementa gRPC como protocolo para la carga y descarga de archivos entre los nodos, ya que permite llamadas a procedimientos

remotos altamente eficientes, lo que reduce la latencia y optimiza el rendimiento en la transmisión de datos binarios.

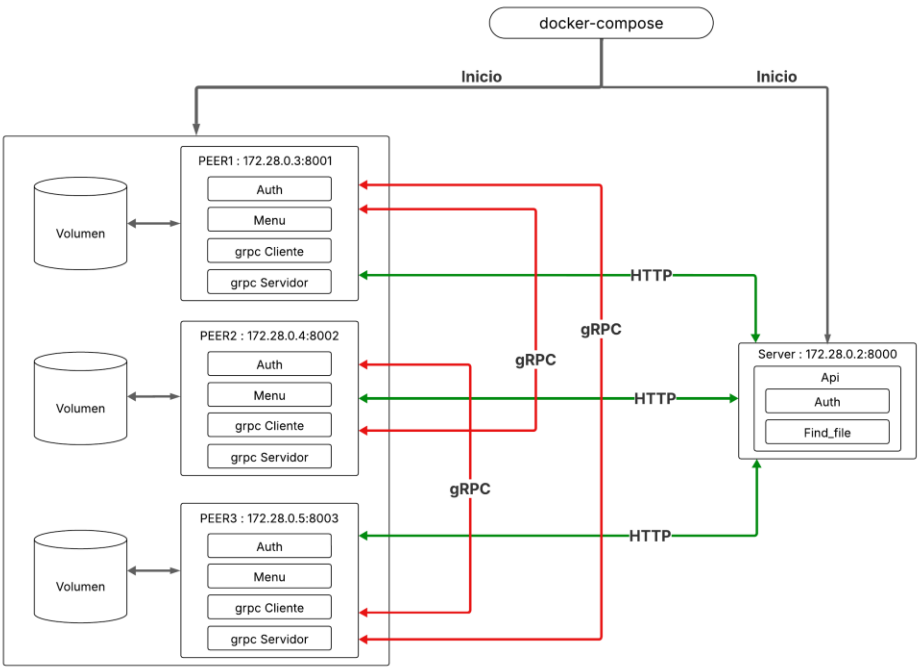
DESCRIPCIÓN DEL SERVICIO Y PROBLEMA ABORDADO.

Las redes cliente-servidor tradicionales presentan limitaciones en escalabilidad y tolerancia a fallos, lo que dificulta la compartición eficiente de archivos. Las redes P2P ofrecen una alternativa distribuida y descentralizada, pero requieren mecanismos de localización de recursos y concurrencia.

El sistema propuesto implementa una red P2P no estructurada con un servidor de directorio y localización, donde cada peer integra microservicios que cumplen funciones de cliente y servidor. Estos servicios permiten autenticar nodos, consultar índices de archivos y simular operaciones de descarga y carga entre pares.

La comunicación se organiza de forma híbrida: API REST sobre HTTP para autenticación y consultas, y gRPC para la transferencia simulada de archivos. Con este enfoque, se logra una red descentralizada, concurrente y modular, capaz de soportar la compartición distribuida de recursos.

ARQUITECTURA DEL SISTEMA Y DIAGRAMAS.



En primer lugar, el proyecto se construye bajo una arquitectura orientada a servicios (SOA) haciendo uso del patrón REST, para la implementación de una arquitectura (P2P) basada en servidor.

Del mismo modo, la arquitectura anterior, está constituida por 4 nodos o peers, los cuales se distribuyen de la siguiente manera: un nodo representando el servidor (arquitectura p2p basada en servidor) y 3 peers, cada uno de los anteriores se despliegan utilizando Docker, siendo cada peer un contenedor diferente, mediados por Docker-Compose. Los peers, poseen cada uno un volumen, en los cuales se almacenarán los archivos a localizar e intercambiar mediante gRPC.

Por otro lado, es importante destacar que el peer servidor en la presente arquitectura soporta concurrencia, es decir permite invocar (llamada) a más de un procedimiento remoto, gracias al establecimiento de protocolos REST y gRPC.

ESPECIFICACION DE PROTOCOLOS Y API.

La API, esta distribuida de una manera escalable, estableciendo módulos independientes para favorecer el rendimiento de la aplicación, haciendo uso del patrón REST, mediado por el lenguaje de programación Python y su framework para la creación de APIs FastAPI (https://fastapi.tiangolo.com/) (ver requirements.txt del proyecto).

El API se compone de (routes, services y un archivo main). En los servicios se establecen los métodos de la interface que luego consumirán las rutas. Los métodos definidos en los servicios son: login_peer(), buscar_archivo(), limpiar_peers_inactivos(). Por el lado de las routes, se definen tres rutas principales para el funcionamiento del sistema, las cuales son: /login, /buscar_archivos, /peers_activos, /ver_archivos.

En esta API se usa el protocolo HTTP, para permitir principalmente, la gestión de peers e indexacion de archivos. Para el apartado de transferencia de archivos se utilizará gRPC como protocolo de comunicación (grpcio).

Por otro lado, en el proyecto se estructura el directorio Peer, en cual los archivos principales son: bootstrap.py, dockerfile y peer.py. El orden de ejecución es que el bootstrap es el encargado de

ejecutar/levantar la autenticación, el servidor y el menú para la interacción con Docker (revisar /peer/Dockerfile), para la autenticación e inicialización de los peers, bootstrap invoca el método start_peer() del archivo peer, el cual es el método principal que construye un peer, indexa sus archivos, establece el menú y hace que el peer se autentique de manera recurrente cada 30 segundos (establecido en el método auth_loop() del archivo peer).

De este modo, utilizando docker-compose se establece una construcción automática y monitoreada de peers, permitiendo construir la arquitectura (establecimiento de red, IPv4 privada de los contenedores/peers del sistema P2P, la generación de volúmenes (persistencia de archivos) en cada peer), etc. (https://github.com/thomasvanegas/REST_API_P2P_Architecture/blob/main/docker-compose.yml)

Del mismo modo, es importante destacar, que en la siguiente ruta se encuentra la configuración y desarrollo del archivo .proto para el protocolo gRPC: REST_API_P2P_Architecture/Peer/peer.proto at main · thomasvanegas/REST_API_P2P_Architecture

Adicionalmente, en el mismo directorio (/Peer) se encuentra la definición de la autenticación, métodos e interfaces del proyecto, especialmente, el archivo del gRPC Client y el gRPC Server.

DESCRIPCIÓN DEL ENTORNO EN DOCKER.

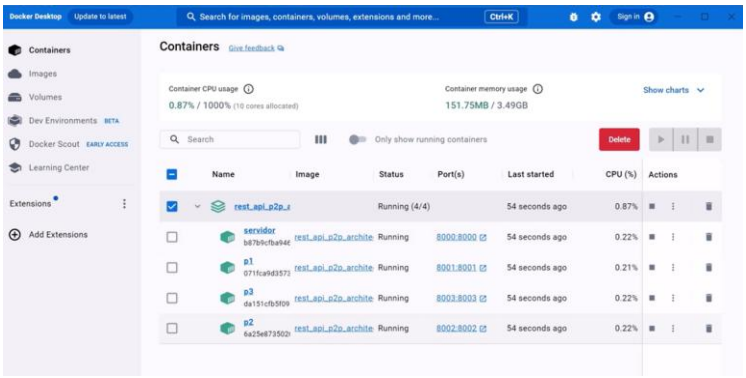
Para la ejecución en el entorno de Docker, se compone el proyecto de dos Dockerfile, uno para la API y otro para la construcción de PEERS. Así pues, en la carpeta raíz del proyecto se define el docker-compose.yml que permite la creación de la arquitectura.

Por otro lado, para la interacción con el sistema y ejecutar las acciones propuestas y establecidas: login, visualizar archivos, indexar archivo, descargar archivo (gRPC), cargar/subir archivo (gRPC), etc.

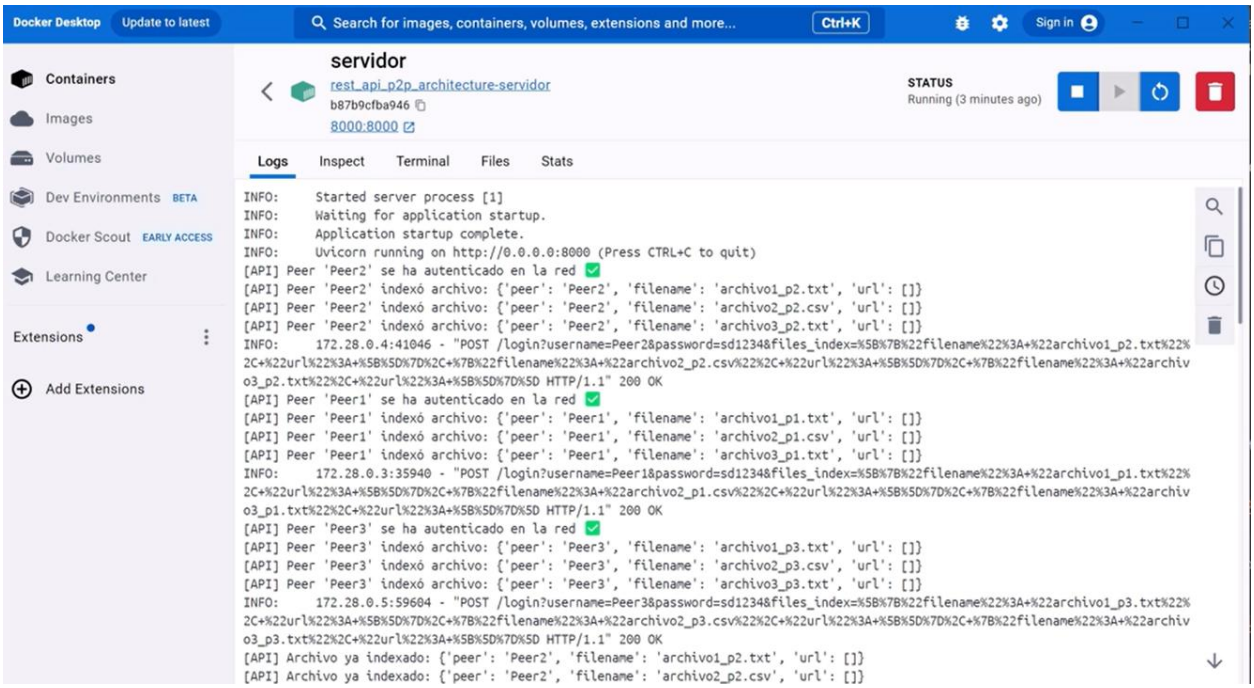
En la siguiente ruta se presenta la definición del Docker-compose: REST_API_P2P_Architecture/docker-compose.yml at main · thomasvanegas/REST_API_P2P_Architecture

PRUEBAS Y ANÁLISIS DE RESULTADOS.

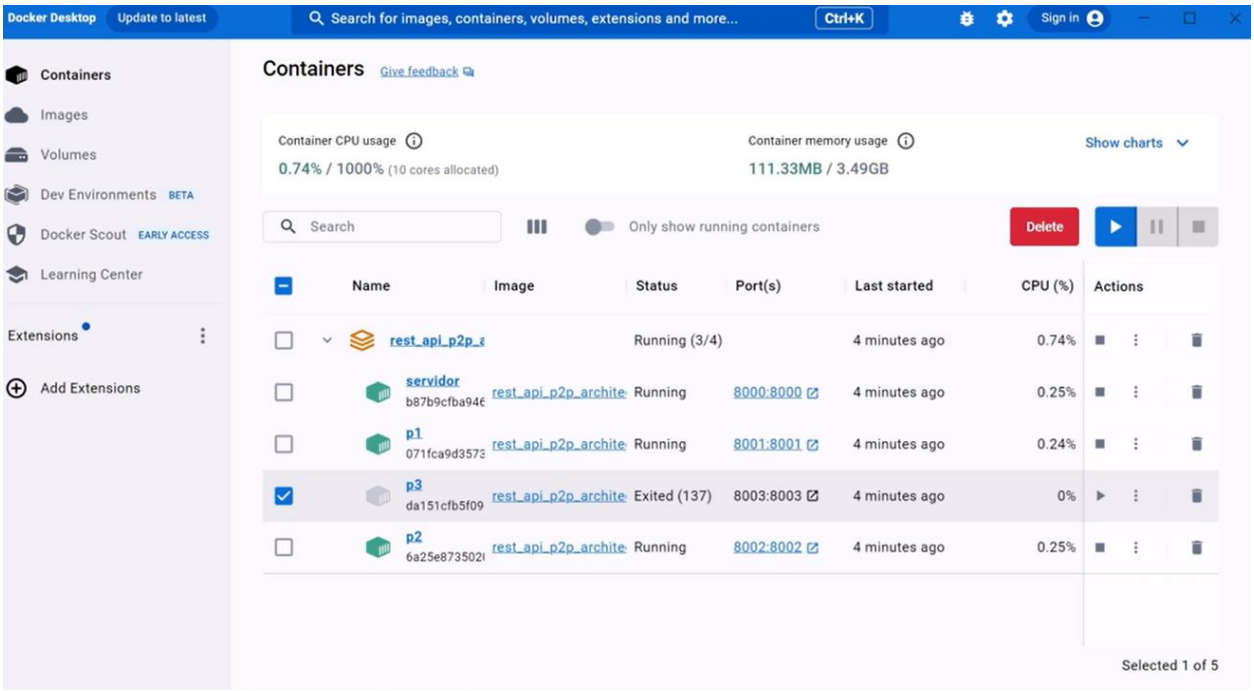
Cuando se ejecuta la instrucción “docker-compose up --build” se crean los cuatro contenedores (PServidor y 3 Peers clientes).



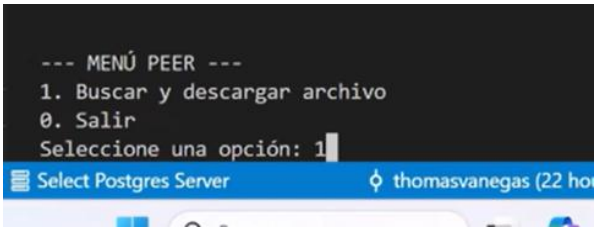
Luego, se puede visualizar en el Peer Servidor que se sube correctamente la API desarrollada en FastAPI (uvicorn main:app --reload) y como se tiene en el docker-compose se construyen los 3 peers, se autentican de manera cíclica (loop) cada 30 segundos (como se indica en el vídeo demostrativo) y se indexan los archivos, es decir, cada peer le indica al servidor que archivos posee, todo esto, de manera automatizada.



Luego, para probar que el servidor es capaz de administrar/gestionar la caída de uno de los peers, apagamos/stop el contenedor asociado al peer3.



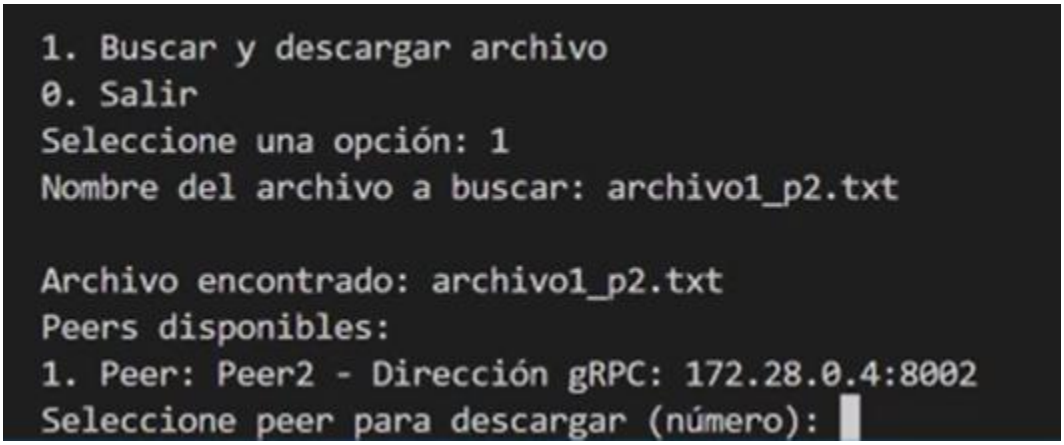
Luego, se ingresa a uno de los peers por consola (CLI) y se ejecuta el menú interactivo.



Cabe destacar, que cuando se apaga el contenedor de Docker, en la visualización de los logs del Peer Servidor se visualiza la desconexión del Peer3, en este caso.

En la imagen anterior se evidencia, que el Peer1 hace la autenticación cíclica e indexa nuevamente sus archivos (los cuales están la carpeta /data del repositorio).

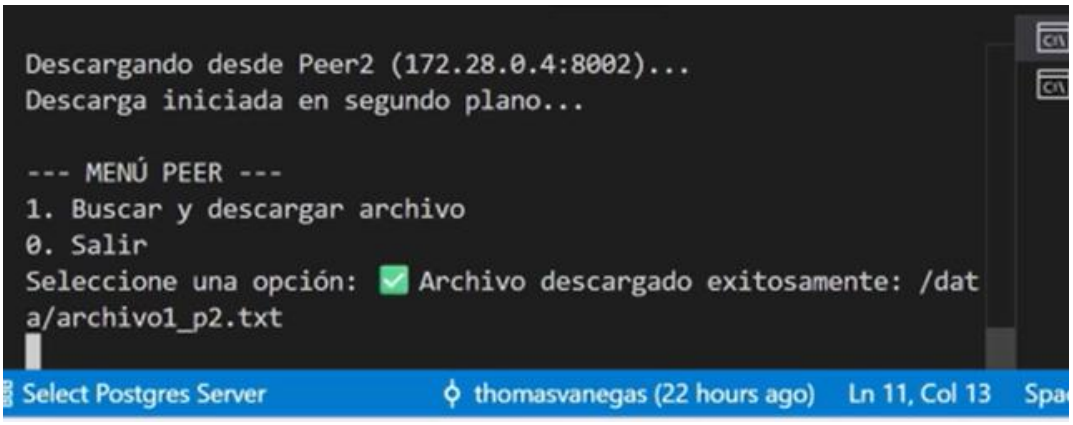
Luego, en el menú interactivo se selecciona la opción 1, correspondiente a buscar y descargar archivo. En este caso, se realizan pruebas desde Peer1 para buscar un archivo que se encuentra en peer2.



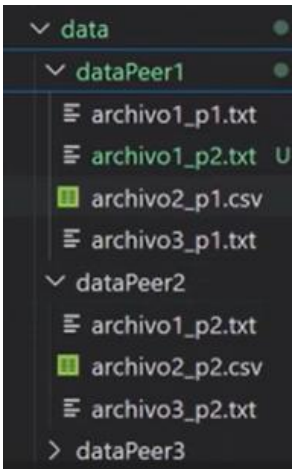
Se evidencia que el servidor encuentra el archivo y brinda la dirección utilizando el protocolo gRPC del Peer2, quién es el Peer que posee el archivo.

Del mismo modo, cabe destacar que el sistema entrega la lista de los Peers disponibles, es decir, quienes tienen el archivo, en este caso solo aparece Peer2 debido a que se utiliza el directorio /data/ y por cuestiones de pruebas se crearon archivos únicos que los identificara.

Luego, se ingresa en que Peer se quiere descargar el archivo y el sistema P2P gracias al protocolo gRPC descargará el archivo.



Posteriormente, se verifica el volumen del contenedor de Docker, que en este caso esta asociado, a la carpeta /data y se evidencia la correcta descarga del archivo desde Peer2 al Peer1.



Así pues, se evidencia el correcto funcionamiento de la arquitectura P2P, haciendo uso de HTTP usando REST y el protocolo gRPC, permitiendo concurrencia, manejo de fallos mediante una arquitectura de gran escalabilidad.