Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA

UNIDAD CUAJIMALPA

LICENCIATURA EN TECNOLOGÍAS Y SISTEMAS DE INFORMACIÓN.

PROYECTO FINAL

Mini Plataforma de Video Streaming P2P con Microservicios.

UNIDAD DE ENSEÑANZA

SISTEMAS DISTRIBUIDOS

Dr. Guillermo Monroy

Marin Sanchez Jose Abraham
2233030514

INTRODUCCIÓN

Este proyecto tiene como objetivo diseñar y construir un sistema distribuido que tenga la capacidad de intercambiar fragmentos de un video por medio de nodos usando una arquitectura Peer to Peer. Donde los fragmentos estarán divididos en diferentes nodos y uno a otro podrán solicitarse fragmentos que les falten.

La plataforma a realizar nos pide ciertos requisitos a cumplir:

1. Dividir un video en 10 fragmentos.
2. Implementar nodos que intercambien estos fragmentos mediante peer to peer.
3. Un API o Microservicio que gestione los fragmentos de cada nodo.
4. PubSub para saber qué fragmentos tiene cada nodo.
5. Una interfaz básica mediante consola o logs para pedir o recibir fragmentos de video.
6. Usar Docker y Swagger.

Al existir una cuentas propuestas de proyecto , este debe cumplir con una arquitectura propuesta y utilizar algunas tecnologías sugeridas.

En mi caso el proyecto elegido fue el proyecto en Python de manera individual por lo cual se recomendó usar las siguientes tecnologías:

- Flask
- Sockets TCP simples
- PubSub simulado internamente.

Y para la arquitectura se pidió:

- Hacer uso de dos nodos en el peer to peer los cuales pueden ser dos procesos o dos scripts separados.
- Comunicación por TCP o HTTP.
- Que el PubSub fuera simulado mediante listas internas.
- Que el usuario decida manualmente a qué nodo conectarse.
- Logs Claros.

DESARROLLO

Para la arquitectura de este proyecto que necesita un tipo de mezcla entre peer to peer y pubsub , utilice un broker para la gestión y la búsqueda de los fragmentos del video además de el uso de el framework flask y docker para los contenedores. A continuación explicaré como se intenta hacer el uso de tanto el P2P y el PUBSUB.

Peer to Peer

El sistema tiene un nodo que se ejecuta dos veces con puertos diferentes los cuales operan estos como pares entre ellos. Para que cada uno de los procesos tenga sus fragmentos se divide un video en 10 partes y luego estas se distribuyen de manera manual entre las carpetas de fragmentos de los dos nodos.
Cuando un nodo necesite de un fragmento que no tiene , se comunica para poder obtenerlo del otro nodo si este lo tiene.

El funcionamiento en el intercambio de fragmentos es:

1. El nodo que requiera un fragmento puede consultar en el broker la ubicación del fragmento que quiere.
2. El broker le devuelve el nodo en el que está ese fragmento.
3. Entonces considerando que el fragmento está en el nodo 2, el nodo 1 solicita al nodo 2 el fragmento
4. El nodo 2 le responde mandando el fragmento pedido.

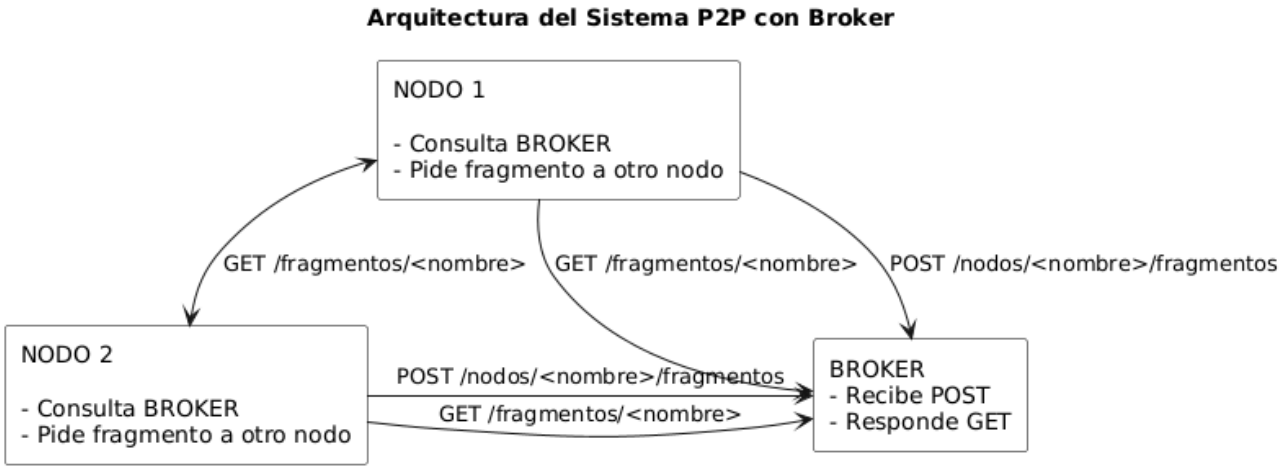
PubSub

El pubSub del sistema se simula en la interacción entre los nodos y el broker, ya que el broker es el que se encarga de gestionar tanto la publicación y suscripción de los datos sobre la disponibilidad de los fragmentos de video de cada nodo.

Publicador: Un nodo será publicador cuando se ejecute o tenga un nuevo fragmento , esto mediante un endpoint de registro que envía al broker los fragmentos que tiene.

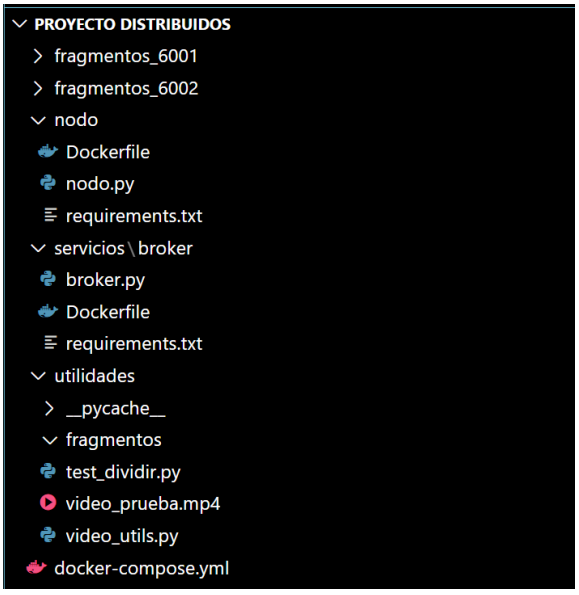
Suscriptor: Un nodo sería suscriptor al necesitar un fragmento , porque consultará al broker que él dirá en cuál nodo está el fragmento de video que necesita.

Diagrama de la arquitectura:



CONSTRUCCIÓN DEL SISTEMA

La estructura de las carpetas y archivos del programa es la siguiente:



Se tienen dos carpetas de fragmentos que es donde se ponen los fragmentos de cada nodo, la carpeta nodo donde se tiene el nodo.py, dockerfile y el requirements.

El código de **nodo.py** establece las funciones de cada nodo del sistema P2P, sus funciones permiten que se puedan compartir y descargar fragmentos del video, además de comportarse como server para el otro nodo levantado y como cliente para el broker.

Fragmentos importantes:

-Función para registrar nodo en el broker.

```python
##registra el nodo en el broker con los fragmentos que tiene
def registrar_en_broker():
    try:
        if not os.path.exists(CARPETA_FRAGMENTOS):
            print(f"Error la carpeta de fragmentos '{CARPETA_FRAGMENTOS}' no existe.")
            return False

        fragmentos = os.listdir(CARPETA_FRAGMENTOS)
        url = f"{broker_URL}/nodos/{NOMBRE_NODO}/fragmentos"

        intentos_max = 10
        for i in range(intentos_max):
            try:
                r = requests.post(url, json={"fragmentos": fragmentos})
                if r.status_code == 200:
                    print(f"Nodo '{NOMBRE_NODO}' registrado con {len(fragmentos)} fragmentos.")
                    return True
                else:
                    print(f"Error al registrar nodo: {r.text}")
                    return False
            except requests.exceptions.ConnectionError:
                print(f"Intentando conectar con el broker... (Intento {i+1}/{intentos_max})")
                time.sleep(2)

        print("No se pudo conectar con el broker después de varios intentos.")
        return False
    except FileNotFoundError:
        print(f"Error: La carpeta de fragmentos '{CARPETA_FRAGMENTOS}' no existe.")
        return False
```

Intenta conectarse al broker para decirle que partes del video tiene, además tiene un bucle para intentar registrar el nodo por si falla al iniciar.

-Función para saber los nodos que tienen algún fragmento.

```python
#Consulta al broker para obtener una lista de nodos que tienen un fragmento
def buscar_nodos_con_fragmento(nombre_fragmento):
    try:
        url = f"{broker_URL}/fragmentos/{nombre_fragmento}/nodos"
        respuesta = requests.get(url)
        if respuesta.status_code == 200:
            datos = respuesta.json()
            return datos.get("nodos", [])
        else:
            print(f"No se pudo consultar el broker: {respuesta.text}")
    except Exception as e:
        print(f"Error consultando broker: {e}")
    return []
```

-Función para descarga de fragmentos.

```python
#Descarga un fragmento desde otro nodo que lo tenga
def descargar_fragmento_de_nodo(nodo, nombre_fragmento):
    try:
        host, port_str = nodo.split(":")
        url = f"http://{host}:{port_str}/fragmentos/{nombre_fragmento}"

        # usa streaming para archivos grandes
        respuesta = requests.get(url, stream=True)

        if respuesta.status_code == 200:
            ruta_guardado = os.path.join(CARPETA_FRAGMENTOS, nombre_fragmento)
            with open(ruta_guardado, 'wb') as f:
                for chunk in respuesta.iter_content(chunk_size=8192):
                    f.write(chunk)
            print(f"Fragmento '{nombre_fragmento}' descargado desde '{nodo}'.")

            # logica p2p, se vuelve a registrar
            registrar_en_broker()
            return True
        else:
            print(f"Fragmento no encontrado en nodo '{nodo}'")
    except Exception as e:
        print(f"Error descargando fragmento de '{nodo}': {e}")
    return False
```

Se conecta a otro nodo con la información conseguida con el broker y descarga el fragmento, a la vez si se descarga bien, el nodo se vuelve a registrar al broker para que sepa que tiene el nuevo fragmento.

ENDPOINTS:

Los endpoints cuentan con un docstring que es necesario para la documentación en swagger.

```python
@app.route('/descargar/<nombre_fragmento>', methods=['GET'])
def descargar_fragmento(nombre_fragmento):
    """
    Endpoint para solicitar la descarga de un fragmento.
    ---
    tags:
      - Cliente
    parameters:
      - name: nombre_fragmento
        in: path
        type: string
        required: true
        description: Nombre del fragmento a descargar
    responses:
      200:
        description: Fragmento descargado exitosamente.
      404:
        description: Fragmento no encontrado en ningún nodo.
      500:
        description: Error al descargar el fragmento.
    """
    nodos = buscar_nodos_con_fragmento(nombre_fragmento)
    if not nodos:
        return jsonify({"error": "El fragmento no se encontro en ningun nodo"}), 404

    # busca y descarga del primer nodo disponible
    for nodo in nodos:
        if nodo != NOMBRE_NODO:
            if descargar_fragmento_de_nodo(nodo, nombre_fragmento):
                return jsonify({"mensaje": f"Fragmento '{nombre_fragmento}' descargado de '{nodo}'."}), 200

    # si no se pudo descargar en ningun nodo
    return jsonify({"error": "No fue posible descargar el fragmento desde los nodos disponibles"}), 500


@app.route("/fragmentos/<nombre>", methods=['GET'])
def enviar_fragmento(nombre):
    """
    Endpoint para que otros nodos descarguen un fragmento.
    ---
    tags:
      - Intercambio P2P
    parameters:
      - name: nombre
        in: path
        type: string
        required: true
        description: Nombre del fragmento a enviar
    responses:
      200:
        description: Archivo fragmento encontrado y enviado
        content:
          application/octet-stream:
            schema:
              type: string
              format: binary
      404:
        description: Fragmento no encontrado
    """
    ruta_fragmento = os.path.join(CARPETA_FRAGMENTOS, nombre)
    if os.path.exists(ruta_fragmento):
        return send_from_directory(CARPETA_FRAGMENTOS, nombre)
    else:
        return jsonify({"error": "Fragmento no encontrado"}), 404
```

Endpoint para la descarga y endpoint para que se conecte el otro nodo.

```python
@app.route("/mis_fragmentos", methods=['GET'])
def listar_fragmentos():
    """
    Lista todos los fragmentos disponibles localmente en este nodo.
    ---
    tags:
      - Cliente
    responses:
      200:
        description: Lista de fragmentos encontrados
        schema:
          type: object
          properties:
            fragmentos:
              type: array
              items:
                type: string
    """
    fragmentos_existentes = os.listdir(CARPETA_FRAGMENTOS) if os.path.exists(CARPETA_FRAGMENTOS) else []
    return jsonify({"fragmentos": fragmentos_existentes})
```

Endpoint que muestra los fragmentos del nodo.

```python
@app.route("/nodos/<nombre_nodo>/fragmentos", methods=['GET'])
def obtener_fragmentos_de_nodo(nombre_nodo):
    """
    Obtiene la lista de fragmentos que posee un nodo específico consultando al broker.
    ---
    tags:
      - Cliente
    parameters:
      - name: nombre_nodo
        in: path
        type: string
        required: true
        description: Nombre del nodo a consultar (ej. 'nodo1:6001')
    responses:
      200:
        description: Lista de fragmentos que tiene el nodo
        schema:
          type: object
          properties:
            fragmentos:
              type: array
              items:
                type: string
      404:
        description: Nodo no encontrado en el broker.
    """
    try:
        url = f"{broker_URL}/nodos/{nombre_nodo}/fragmentos"
        respuesta = requests.get(url)
        if respuesta.status_code == 200:
            datos = respuesta.json()
            return jsonify(datos), 200
        else:
            return jsonify({"error": "Nodo no encontrado en el broker"}), 404
    except Exception as e:
        print(f"Error al consultar el broker: {e}")
        return jsonify({"error": "Error interno al consultar el broker"}), 500
```

Endpoint para buscar los fragmentos según el nodo.

El código de **broker.py** es el que define el servidor que sirve como un directorio, su función principal es saber en qué nodo está algún fragmento.

Cuenta con dos diccionarios :

```
fragmentos_por_nodo = {}
nodos_registrados = {}
```

fragmentos_por_nodo guarda el nombre del fragmento y el nodo o nodos donde esta ese fragmento y nodos_registrados que lleva el registro de los nodos registrados.

ENDPOINTS:

```
@app.route("/nodos/<nombre_nodo>/fragmentos", methods=['POST'])
def registrar_nodo(nombre_nodo):
    """
    Registra un nodo y los fragmentos que posee.
    ---
    tags:
      - Registro de Nodos
    parameters:
      - name: nombre_nodo
        in: path
        type: string
        required: true
        description: Nombre del nodo que se registra (ej. 'nodo1:6001')
      - name: body
        in: body
        required: true
        schema:
          type: object
          properties:
            fragmentos:
              type: array
              items:
                type: string
              description: Lista de nombres de los fragmentos que el nodo posee
    responses:
      200:
        description: Nodo y fragmentos registrados exitosamente
    """
    data = request.json
    fragmentos = data.get("fragmentos", [])

    # elimina registros viejos del nodo
    for f in list(fragmentos_por_nodo.keys()):
        if nombre_nodo in fragmentos_por_nodo[f]:
            fragmentos_por_nodo[f].remove(nombre_nodo)
            if not fragmentos_por_nodo[f]:
                del fragmentos_por_nodo[f]

    # registra nuevos fragmentos
    for fragmento in fragmentos:
        if fragmento not in fragmentos_por_nodo:
            fragmentos_por_nodo[fragmento] = []
        if nombre_nodo not in fragmentos_por_nodo[fragmento]:
            fragmentos_por_nodo[fragmento].append(nombre_nodo)

    nodos_registrados[nombre_nodo] = True
    print(f"Nodo '{nombre_nodo}' registrado. Tiene {len(fragmentos)} fragmentos.")
    return jsonify({"mensaje": "Nodo registrado"}), 200
```

Endpoint principal que recibe la solicitud y registra el nodo y los fragmentos que tiene.

```python
@app.route("/fragmentos/<nombre_fragmento>/nodos", methods=['GET'])
def buscar_nodos_con_fragmento(nombre_fragmento):
    """
    Obtiene la lista de nodos que tienen un fragmento específico.
    ---
    tags:
      - Búsqueda de Fragmentos
    parameters:
      - name: nombre_fragmento
        in: path
        type: string
        required: true
        description: Nombre del fragmento a buscar
    responses:
      200:
        description: Lista de nodos que poseen el fragmento
        schema:
          type: object
          properties:
            nodos:
              type: array
              items:
                type: string
    """
    nodos = fragmentos_por_nodo.get(nombre_fragmento, [])
    return jsonify({"nodos": nodos}), 200
```

Endpoint de búsqueda que se usa para saber en qué nodo está cierto fragmento.

Para ambos archivos se tiene un dockerfile que es el que le dice a docker cómo construir una imagen de la aplicación y el requirements que trae el nombre de las dependencias necesarias a instalar.

```dockerfile
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY nodo.py .

CMD ["python", "nodo.py", "6000"]
```

```
Flask==2.0.3
Flasgger==0.9.5
requests==2.28.1
Werkzeug==2.0.3
```

En la carpeta utilidades tengo los archivos necesarios para dividir el video en fragmentos.

Para dividir el video se instalo la biblioteca ffmpeg que se uso en estos archivos.

video_utils.py cuenta con las funciones para saber la duración del video, dividir el video y guardarlos en la ruta con un nombre especifico.

```python
import os
import subprocess

# usa ffprobe para obtener la duracion del video
def obtener_duracion_video(ruta_video):
    try:
        resultado = subprocess.run(
            ["ffprobe", "-v", "error", "-show_entries", "format=duration",
            "-of", "default=noprint_wrappers=1:nokey=1", ruta_video],
            stdout=subprocess.PIPE,
            stderr=subprocess.STDOUT
        )
        duracion = float(resultado.stdout.decode().strip())
        return duracion
    except Exception as e:
        print(f"No se pudo obtener la duracion del video: {e}")
        return None

#divide un video en fragmentos con ffmpeg
def dividir_video_en_fragmentos(ruta_video, carpeta_salida, cantidad_fragmentos=10):

    duracion = obtener_duracion_video(ruta_video)
    if duracion is None:
        print("No se pudo obtener la duracion del video no se puede dividir")
        return

    duracion_fragmento = duracion / cantidad_fragmentos

    # genera ruta de salida para cada fragmento
    nombre_base = os.path.splitext(os.path.basename(ruta_video))[0]
    salida_fragmentos = os.path.join(carpeta_salida, nombre_base + "_%03d.mp4")

    comando = [
        "ffmpeg",
        "-i", ruta_video,
        "-c", "copy",
        "-map", "0",
        "-f", "segment",
        "-segment_time", str(duracion_fragmento),
        "-reset_timestamps", "1",
        salida_fragmentos
    ]

    print(f"Dividiendo video en {cantidad_fragmentos} fragmentos de aproximadamente {duracion_fragmento:.2f} segundos cada uno")
    subprocess.run(comando)
    print(f"Fragmentos guardados en: {carpeta_salida}")
```

Y el dividir.py el cual hace uso de las funciones para dividir el video en 10 fragmentos.

```python
from video_utils import dividir_video_en_fragmentos

video = "video_prueba.mp4"
carpeta = "fragmentos"
dividir_video_en_fragmentos(video, carpeta, cantidad_fragmentos=10)
```

Finalmente tengo el docker-compose que define cómo se ejecutan los archivos en el contenedor en donde levantamos los servicios del broker , y dos veces el nodo pero con puertos diferentes y que dependen de que el broker este iniciado ya.

```yaml
version: "3.9"

▷Run All Services
services:
  ▷Run Service
  broker:
    build:
      context: ./servicios/broker
    ports:
      - "5000:5000"
    restart: unless-stopped

  ▷Run Service
  nodo1:
    build:
      context: ./nodo
    volumes:
      - ./fragmentos_6001:/fragmentos_6001
    ports:
      - "6001:6001"
    command: ["python", "nodo.py", "6001"]
    restart: unless-stopped
    depends_on:
      - broker
    environment:
      - NODE_NAME=nodo1

  ▷Run Service
  nodo2:
    build:
      context: ./nodo
    volumes:
      - ./fragmentos_6002:/fragmentos_6002
    ports:
      - "6002:6002"
    command: ["python", "nodo.py", "6002"]
    restart: unless-stopped
    depends_on:
      - broker
    environment:
      - NODE_NAME=nodo2
```

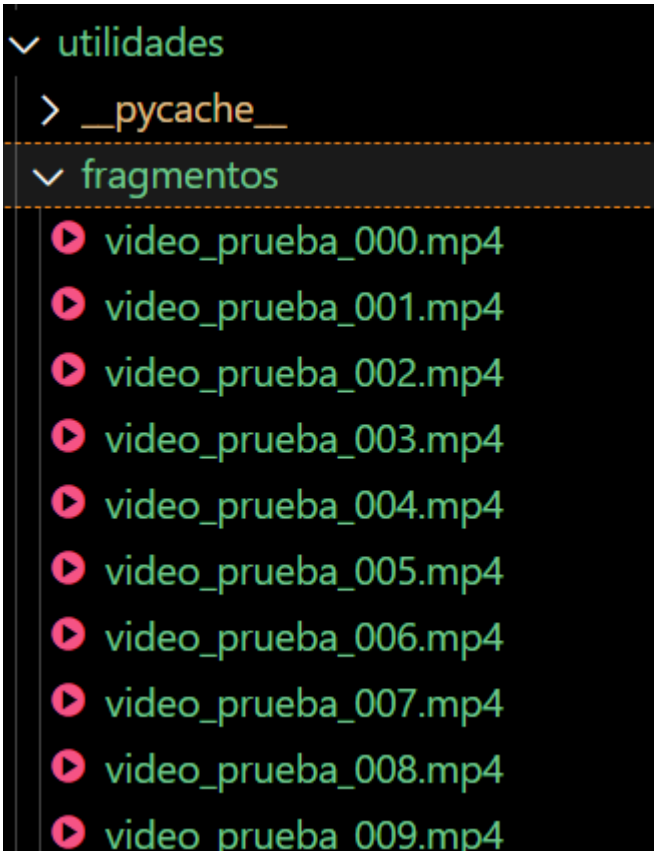
PRUEBAS DE FUNCIONAMIENTO DEL SISTEMA DISTRIBUIDO

En primer lugar necesitamos un video en formato mp4 y ponerlo en la carpeta de utilidades.

Luego en terminal se ejecuta el dividir.py para que se divida en 10 fragmentos los cuales se guardan en la carpeta fragmentos dentro de utilidades.

```
PS C:\Users\joms_\OneDrive\Escritorio\PROYECTO DISTRIBUIDOS\utilidades> python dividir.py
Dividiendo video en 10 fragmentos de aproximadamente 17.99 segundos cada uno
ffmpeg version 2025-08-04-git-9a32b86307-essentials_build-www.gyan.dev Copyright (c) 2000-2025 the FFmpeg developers
  built with gcc 15.1.0 (Rev6, Built by MSYS2 project)
```

```
  Stream #0:1(eng): Audio: aac (LC) (mp4a / 0x6134706D), 44100 Hz, stereo, fltp, 128 kb/s (default)
    Metadata:
      creation_time   : 2023-11-07T11:27:06.000000Z
      handler_name    : ISO Media file produced by Google Inc. Created on: 11/07/2023.
      vendor_id       : [0][0][0][0]
Press [q] to stop, [?] for help
[segment @ 000002838b62eac0] Opening 'fragmentos\video_prueba_001.mp4' for writing
[segment @ 000002838b62eac0] Opening 'fragmentos\video_prueba_002.mp4' for writing
[segment @ 000002838b62eac0] Opening 'fragmentos\video_prueba_003.mp4' for writing
[segment @ 000002838b62eac0] Opening 'fragmentos\video_prueba_004.mp4' for writing
[segment @ 000002838b62eac0] Opening 'fragmentos\video_prueba_005.mp4' for writing
[segment @ 000002838b62eac0] Opening 'fragmentos\video_prueba_006.mp4' for writing
[segment @ 000002838b62eac0] Opening 'fragmentos\video_prueba_007.mp4' for writing
[segment @ 000002838b62eac0] Opening 'fragmentos\video_prueba_008.mp4' for writing
[segment @ 000002838b62eac0] Opening 'fragmentos\video_prueba_009.mp4' for writing
[out#0/segment @ 000002838b6385c0] video:5233KiB audio:2811KiB subtitle:0KiB other streams:0KiB global headers:0KiB muxing overhead: unknown
frame= 4313 fps=0.0 q=-1.0 Lsize=N/A time=00:02:59.84 bitrate=N/A speed=1.24e+03x elapsed=0:00:00.14
Fragmentos guardados en: fragmentos
```



Ahora levantamos el contenedor en la terminal con **docker compose up –build**

```
PS C:\Users\joms_\OneDrive\Escritorio\PROYECTO DISTRIBUIDOS> docker compose up --build
time="2025-08-07T18:54:07-06:00" level=warning msg="C:\\Users\\joms_\\OneDrive\\Escritorio\\PROYECTO DISTRIBUIDOS\\docker-compose.yml:
it will be ignored, please remove it to avoid potential confusion"
[+] Building 3.2s (23/23) FINISHED
 => [internal] load local bake definitions
 => => reading from stdin 1.20kB
 => [broker internal] load build definition from Dockerfile
 => => transferring dockerfile: 190B
```

```
+] Running 7/7
✓ nodo1                                  Built
✓ nodo2                                  Built
✓ broker                                 Built
✓ Network proyectodistribuidos_default   Created
✓ Container proyectodistribuidos-broker-1  Created
✓ Container proyectodistribuidos-nodo2-1   Created
✓ Container proyectodistribuidos-nodo1-1   Created
```

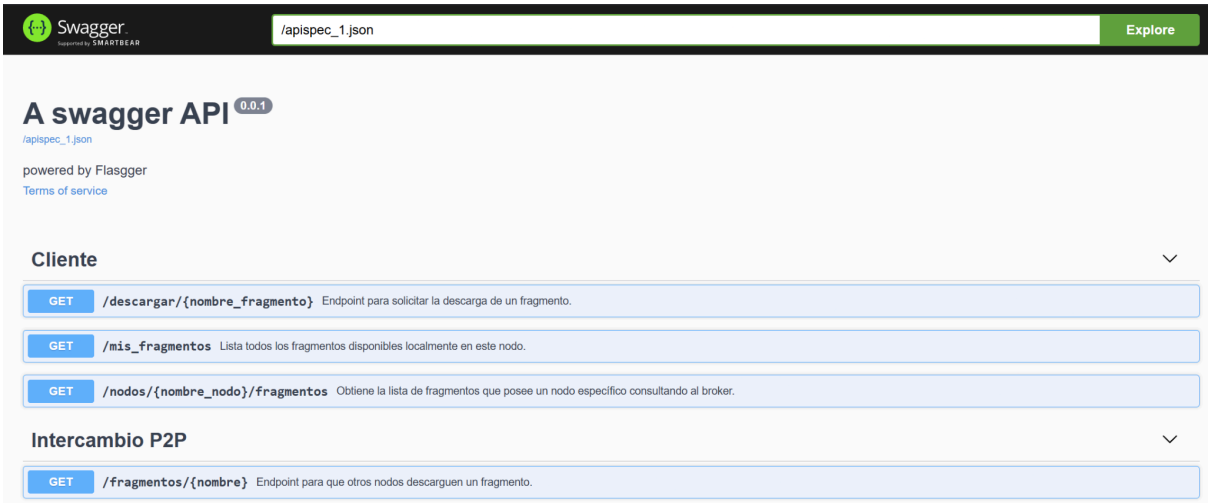
Es cuando los nodos se registran en el broker

```
 * Running on http://172.21.0.2:5000/ (Press CTRL+C to quit)
172.21.0.3 - - [08/Aug/2025 00:54:18] "POST /nodos/nodo1:6001/fragmentos HTTP/1.1" 200 -
172.21.0.4 - - [08/Aug/2025 00:54:18] "POST /nodos/nodo2:6002/fragmentos HTTP/1.1" 200 -
Nodo 'nodo1:6001' registrado con 0 fragmentos.
Nodo 'nodo2:6002' registrado con 0 fragmentos.
 * Serving Flask app 'nodo' (lazy loading)
```

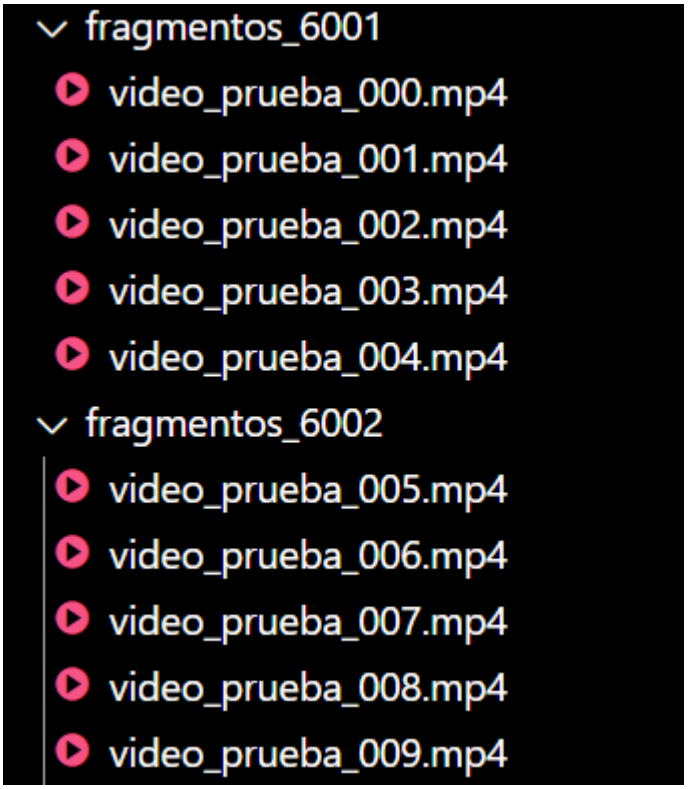
de momento ninguno tiene fragmentos porque no los hemos movido a las carpetas.

Ahora nos podemos meter a los Swagger tanto de los nodos como del broker:

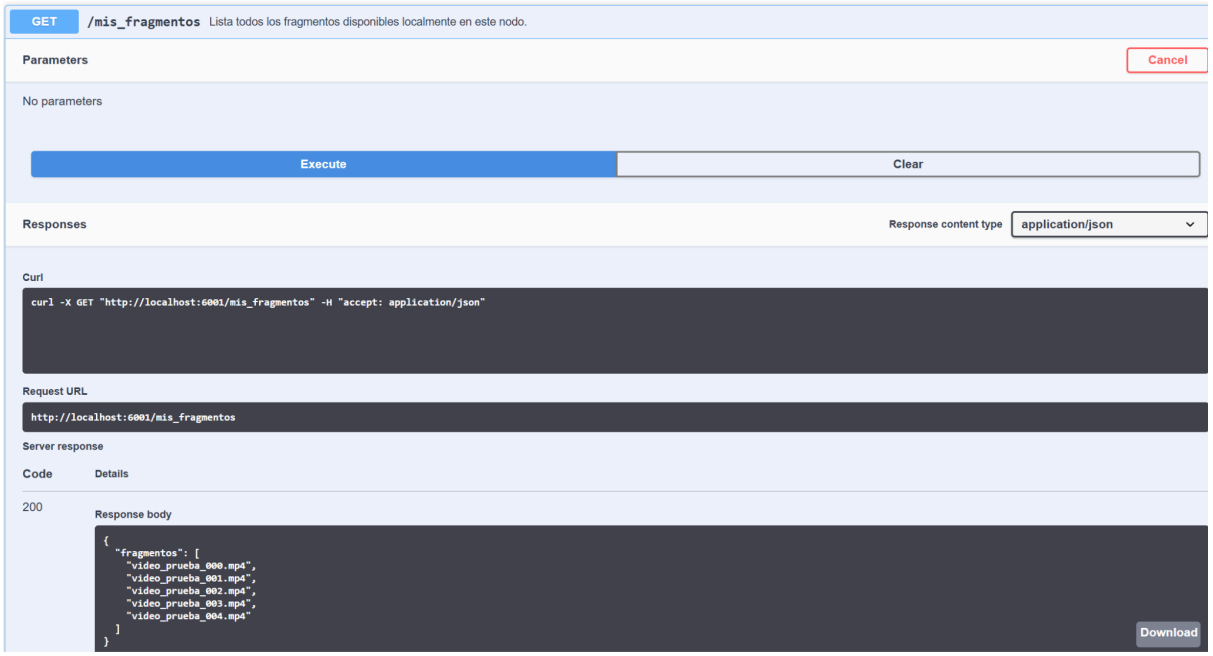
http://localhost:6001/apidocs/ (nodo1) o http://localhost:6002/apidocs/ (nodo2)



Movi 5 fragmentos a cada carpeta

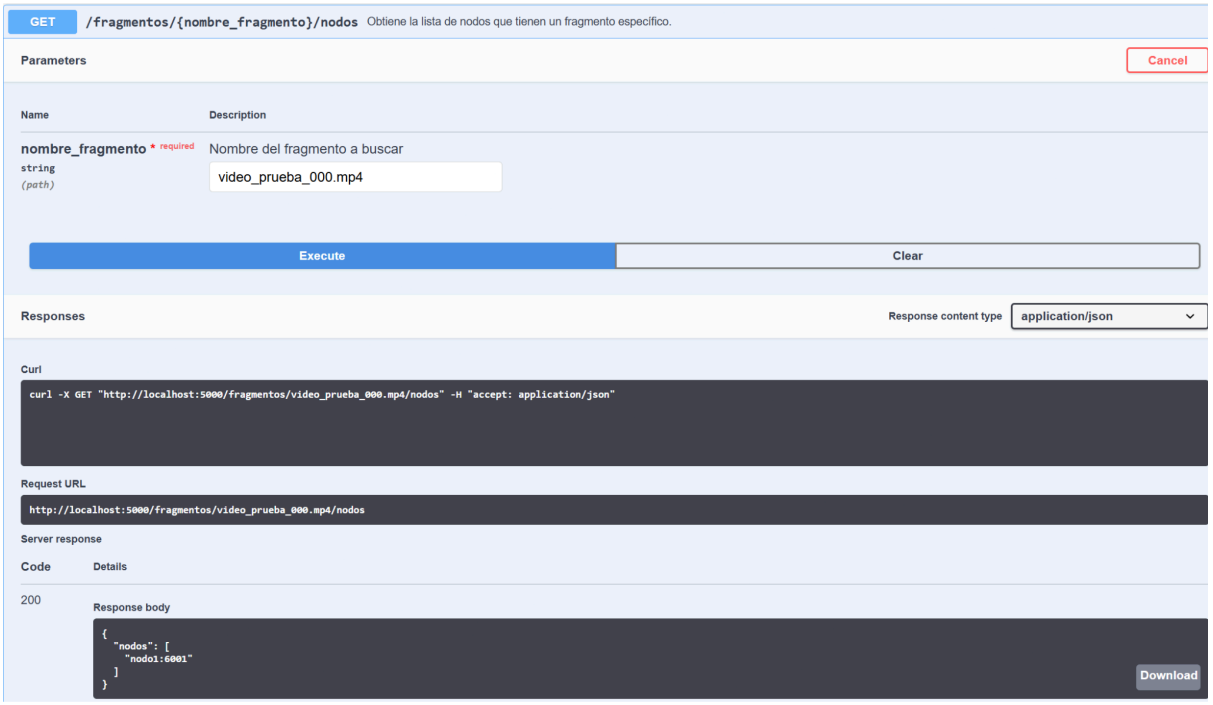


y en el swagger del nodo1 busque que fragmentos tiene con el endpoint de /mis_fragmentos



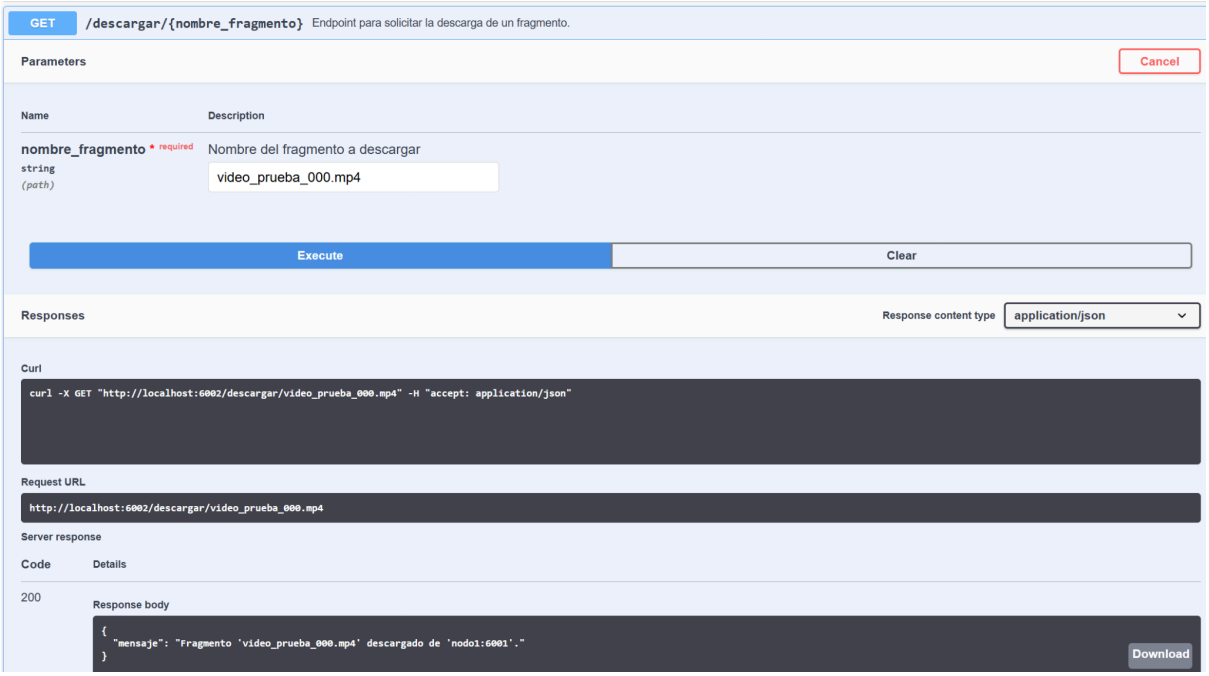
para el nodo 2 se puede hacer lo mismo en su swagger correspondiente.

Ahora desde el broker buscamos la ubicación de algún fragmento , para la prueba buscaré este fragmento video_prueba_000.mp4

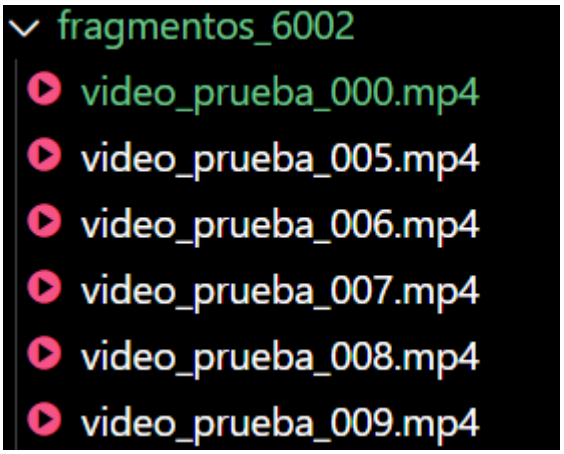


nos dice que ese fragmento está solo en el nodo1.

Entonces desde el nodo 2 hacemos la descarga con el GET /descargar/{nombre_fragmento}



nos dice que se descargo y al revisar la carpeta de fragmentos del nodo2 o el /mis_fragmentos del nodo 2 ya muestra el fragmento descargado.



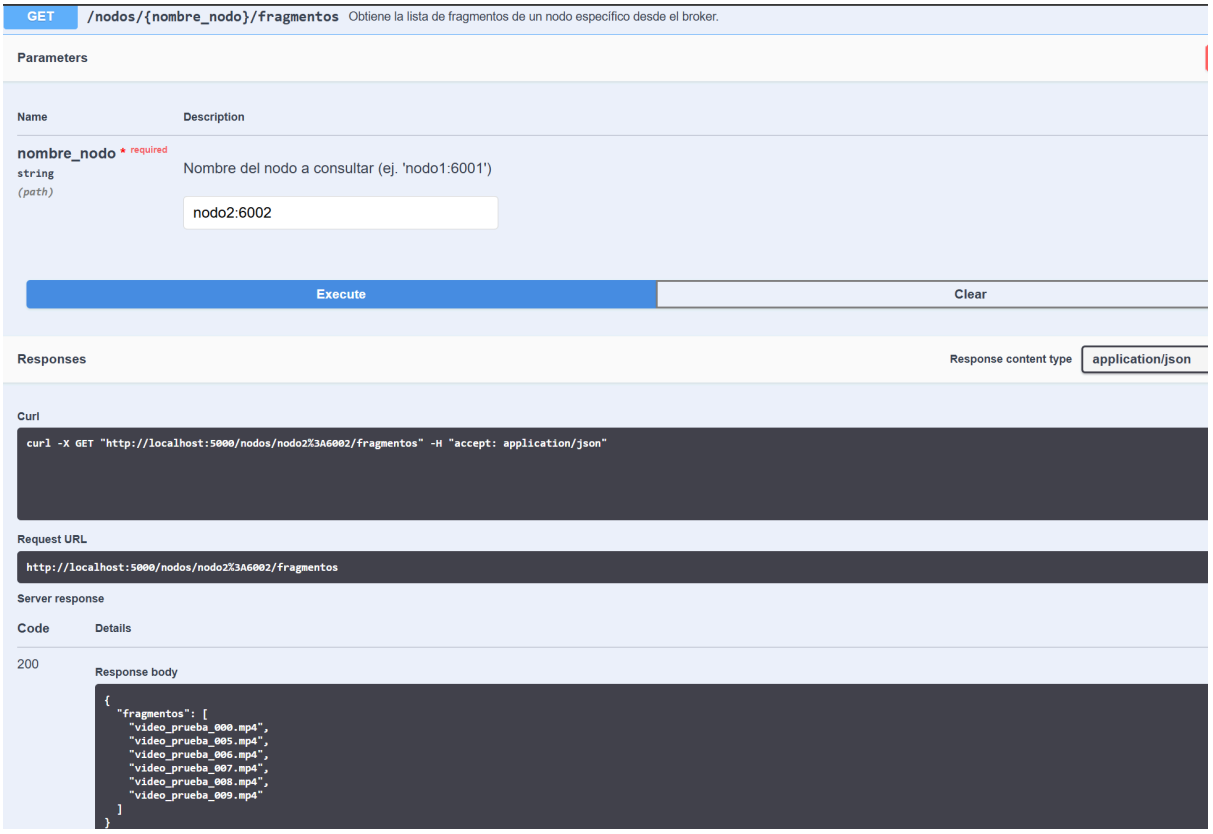
Response body


```
{
  "fragmentos": [
    "video_prueba_000.mp4",
    "video_prueba_005.mp4",
    "video_prueba_006.mp4",
    "video_prueba_007.mp4",
    "video_prueba_008.mp4",
    "video_prueba_009.mp4"
  ]
}
```


Response headers

```
content-length: 155
content-type: application/json
```

También en el swagger del broker se pueden hacer las consultas de los fragmentos para conocer en qué nodo se encuentran.



y si hacemos la búsqueda del fragmento ahora nos dice que está en ambos nodos.


```
{
  "nodos": [
    "nodo2:6002",
    "nodo1:6001"
  ]
}
```

CONCLUSIONES

Este proyecto me sirvió principalmente para reforzar conocimientos acerca de los sistemas distribuidos, si bien en cuanto a teoría creía ya entender el cómo era el funcionamiento de un peer to peer y un pubsub , el ponerlo más en práctica creo que ayuda más a comprender arquitecturas como estas. Además el uso de frameworks como flask se me hace muy interesante ya que durante todo el curso creo ha sido algo fundamental para ver las diferentes arquitecturas.

Considero que el intentar hacer una mezcla entre estas dos arquitecturas fue un reto algo complicado pero interesante por lo que me pondré a la tarea de buscar más recursos de aprendizaje para comprender mejor el proceso de análisis y construcción de sistemas como este.

Creo que el proyecto cumple el cometido principal de transmitir los fragmentos del video mediante los nodos con la arquitectura P2P y la simulación del publicador suscriptor también funciona al hacer consultas a los nodos.

Finalmente como resumen el aprender e implementar estas arquitecturas y el usar diferentes tecnologías como python, flask, github , docker es muy importante ya que seguramente estas tecnologías son muy usadas en la industria tecnológica y el ir obteniendo conocimientos de ellas será clave al momento de adentrarse al mundo laboral.