Universidad Pontificia Bolivariana

Facultad de Ingeniería de Sistemas e Informática
Arquitecturas de nube y Sistemas distribuidos
Proyecto 1 (10%) - septiembre 5 de 2024
Fecha de entrega: septiembre 22 de 2024 GMT-5

Estudiante: Samuel Pérez Hurtado ID: 000459067

Plantilla de autoevaluación y requisitos del proyecto

1. Breve descripción de la actividad
   A. Que aspectos cumplió o desarrolló de la actividad propuesta por el profesor (requerimientos funcionales y no funcionales)
   B. Que aspectos NO cumplió o desarrolló de la actividad propuesta por el profesor (requerimientos funcionales y no funcionales)
2. información general de diseño de alto nivel, arquitectura, patrones, mejores prácticas utilizadas
3. Descripción del ambiente de desarrollo y técnico: lenguaje de programación, librerías, paquetes, etc, con sus números de versiones:
   A. cómo se compila y ejecuta
   B. detalles del desarrollo.
   C. detalles técnicos
   D. descripción y como se configura los parámetros del proyecto (ej: ip, puertos, conexión a bases de datos, variables de ambiente, parámetros, etc)
   E. opcional - detalles de la organización del código por carpetas o descripción de algún archivo. (ESTRUCTURA DE DIRECTORIOS Y ARCHIVOS IMPORTANTE DEL PROYECTO, comando 'tree' de linux)
   F. una mini guía de como un usuario utilizaría el software o la aplicación
   G. opcionalmente - si quiere mostrar resultados o pantallazos
4. otra información que considere relevante para esta actividad
5. referencias
   A. debemos siempre reconocer los créditos de partes del código que reutilizaremos, así como referencias a youtube, o referencias bibliográficas utilizadas para desarrollar el proyecto o la actividad

Éxitos !!!

Samuel Pérez Hurtado – 000459067

Autoevaluación Proyecto 1 P2P – Sistemas Distribuidos

1. **Breve descripción de la actividad**

   El proyecto consistió en el diseño e implementación de un sistema P2P para la compartición de archivos de manera descentralizada utilizando microservicios con comunicación basada en API REST y gRPC. Cada nodo de la red actúa tanto como cliente (enviando solicitudes) como servidor (respondiendo solicitudes de otros nodos). El sistema está diseñado para permitir la localización de archivos en otros nodos de la red y la transferencia de archivos reales.

   A. **Aspectos cumplidos (requerimientos funcionales y no funcionales)**

      Funcionales:

      - Implementación de una red P2P no estructurada con nodos que actúan como clientes y servidores simultáneamente.
      - Uso de API REST para la localización de recursos en la red.
      - Implementación de comunicación mediante gRPC para la interacción entre peers y transferencia de archivos.
      - Separación de los diferentes microservicios, cada uno con funcionalidades distintas para manejo de archivos, localización y simulación de transferencia.
      - Configuración dinámica mediante comandos de configuración, donde cada nodo especifica IP, puerto, directorio de archivos, etc.
      - Despliegue en contenedores utilizando Docker para la ejecución y pruebas en un ambiente virtualizado.
      - Implementación de concurrencia en el servidor gRPC para permitir que múltiples peers interactúen de manera simultánea.

      No funcionales:

      - **Escalabilidad:** El sistema puede escalar fácilmente al añadir más peers a la red sin necesidad de modificar el código base.

- **Concurrencia:** Los servidores gRPC permiten que varios peers interactúen simultáneamente sin bloqueos.
- **Despliegue con Docker:** Utilización de contenedores para asegurar la portabilidad y simplicidad en el despliegue del sistema en diferentes entornos.
- **Uso de tokens para seguridad básica:** Se implementó la autenticación basada en tokens para controlar el acceso a los recursos compartidos.

B. Aspectos sin cumplir

No funcionales:

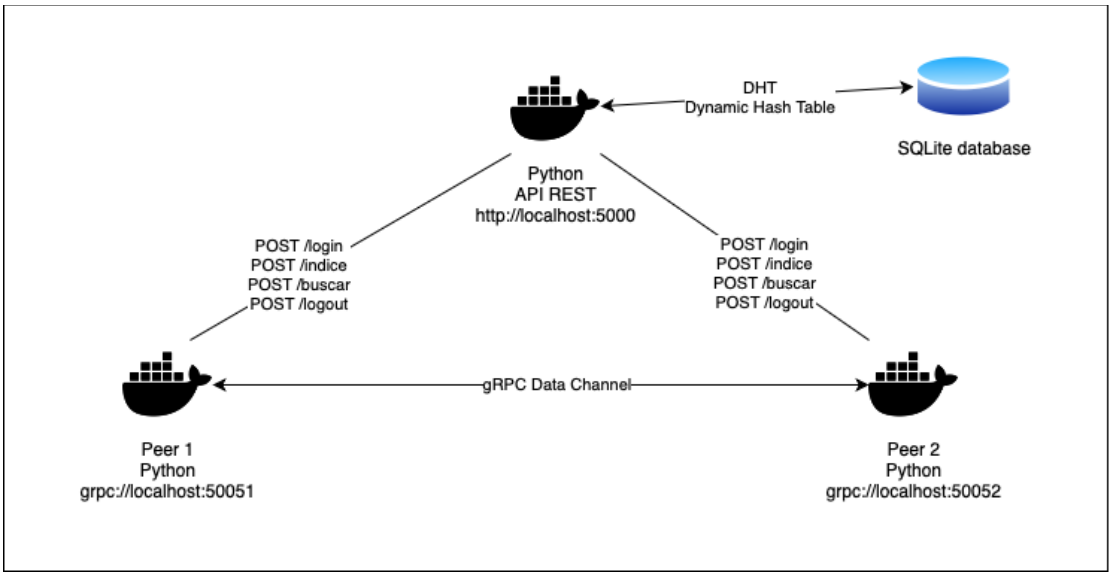
- **Seguridad:** Uso de HTTPS para la API REST, que aunque no se pedía en el Proyecto es un factor importante a tener en cuenta porque deja las transmisiones expuestas a potenciales ataques de intermediarios (MITM).

2. Información general de diseño de alto nivel, arquitectura, patrones, mejores prácticas utilizadas

El diseño sigue una arquitectura distribuida donde cada peer puede actuar tanto como cliente como servidor, creando una estructura de red P2P. Los patrones y mejores prácticas utilizados incluyen:

- **Lenguaje de programación:** Python, ya que la disponibilidad de bibliotecas como Flask (para REST APIs) y grpcio (para gRPC) simplificó el desarrollo del sistema, permitiendo integrar fácilmente servicios de red y comunicación entre los nodos.
- **Microservicios:** Cada funcionalidad (localización, transferencia) está separada en microservicios que interactúan entre sí.
- **Comunicación mediante gRPC:** Para la transferencia de archivos simulada y consultas entre peers.
- **API REST:** Para la localización de recursos y consultas sobre los archivos disponibles.

- **Docker:** Se utiliza para garantizar la portabilidad del sistema y ejecutar múltiples peers de manera aislada y controlada.
- **Concurrencia:** Implementada en el servidor gRPC para permitir que varios peers interactúen al mismo tiempo.



3. Descripción del ambiente de desarrollo y técnico

El proyecto está desarrollado en Python 3.11.5, utilizando una variedad de librerías y paquetes que permiten la comunicación entre peers, el manejo de microservicios y la seguridad básica. A continuación, se detallan las principales dependencias y sus versiones:

- **Bcrypt 4.2.0:** Utilizado para el manejo de contraseñas seguras mediante hash.
- **Blinker 1.8.2:** Soporte de señales para Flask, utilizado para gestionar eventos.
- **Certify 2024.8.30:** Certificados para solicitudes seguras en Python.
- **Charset-normalizer 3.3.2:** Librería para normalización de caracteres en respuestas HTTP.
- **Click 8.1.7:** Librería para crear comandos de línea en Flask.
- **Flask 3.0.3:** Framework para crear aplicaciones web y API REST.
- **Grpcio 1.66.1:** Librería para implementar servicios gRPC en Python.

- **grpcio-tools 1.66.1:** Herramientas para generar código Python a partir de archivos .proto para gRPC.
- **Idna 3.10:** Implementación de nombres de dominio internacionalizados en Python.
- **Itsdangerous 2.2.0:** Para la gestión de firmas seguras en aplicaciones Flask.
- **Jinja2 3.1.4:** Motor de plantillas utilizado por Flask.
- **MarkupSafe 2.1.5:** Proporciona una representación de texto seguro para plantillas Jinja2.
- **Protobuf 5.28.2:** Usado para la serialización de datos en gRPC.
- **Requests 2.32.3:** Librería para realizar solicitudes HTTP.
- **urllib3 2.2.3:** Herramienta para gestionar conexiones HTTP y solicitudes.
- **Watchdog 5.0.2:** Monitorización de cambios en el sistema de archivos, útil para desarrollo.
- **Werkzeug 3.0.4:** Servidor web de desarrollo y librería WSGI para aplicaciones Flask.

A. Cómo se compila y ejecuta

El proyecto está diseñado para ser ejecutado dentro de contenedores Docker. Para configurar y ejecutar el sistema, se deben seguir los siguientes pasos:

1. Clonar el repositorio.
2. Construir los contenedores Docker:
   docker compose build
3. Levantar los servicios:
   docker compose up
4. Este proceso ejecutará los microservicios (peers y API) y los conectará entre sí en una red virtualizada.

B. Detalles del Desarrollo

El desarrollo se realizó utilizando un enfoque de microservicios, separando la lógica de los peers en clientes y servidores que se comunican entre sí a través de gRPC. Además, se implementó una API REST utilizando Flask para facilitar la localización de archivos entre los peers.

C. Detalles técnicos

El sistema de comunicación entre peers utiliza gRPC, donde el archivo file_transfer.proto define los servicios que permiten realizar consultas y transferencias simuladas de archivos. A nivel de API, Flask se utiliza para exponer los puntos de acceso REST que permiten la consulta y localización de archivos en otros peers.

D. Descripción y cómo se configuran los parámetros del Proyecto

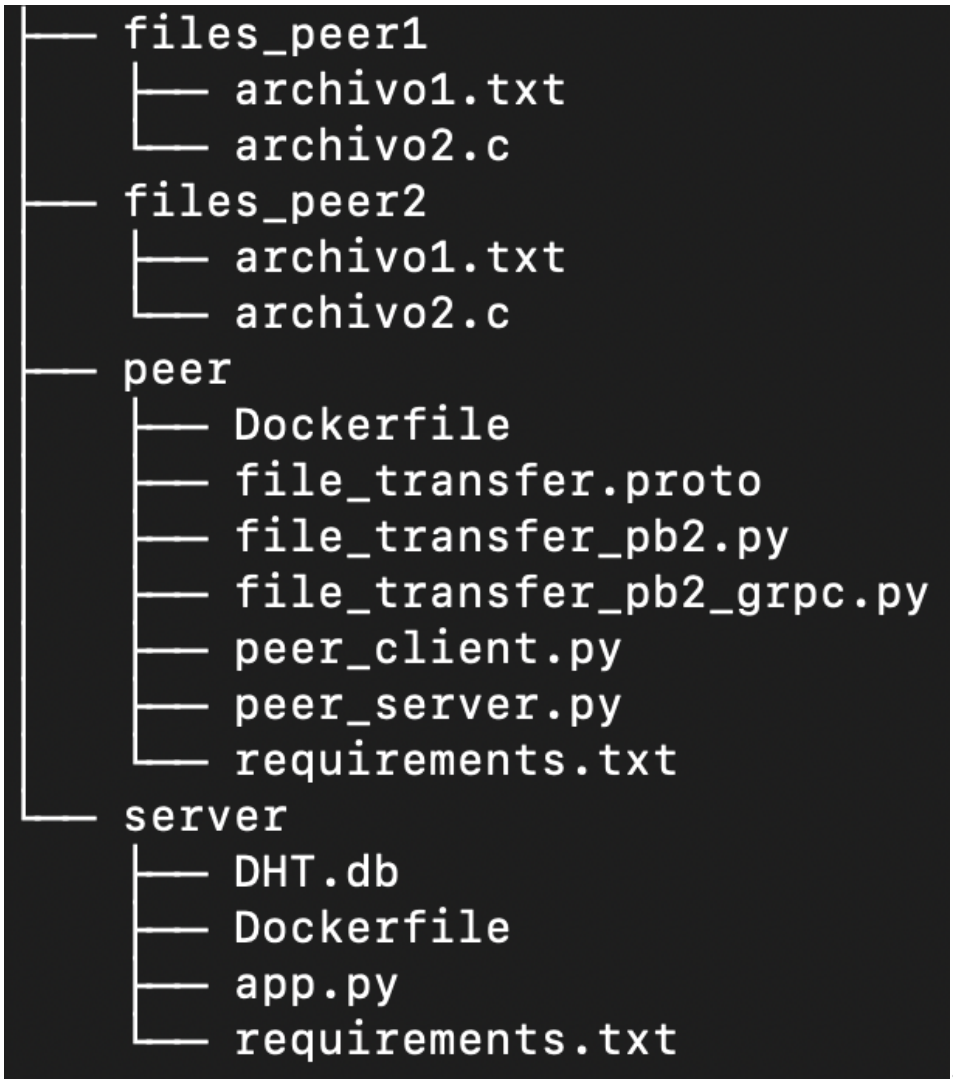
Los parámetros del proyecto se configuran en el archivo docker-compose.yml, donde se especifican las variables de entorno para cada peer:

- GRPC_PORT: El puerto en el que cada peer escuchará solicitudes gRPC.
- USERNAME y PASSWORD: Credenciales básicas para cada peer.
- Volúmenes: Se configuran volúmenes para cada peer, donde se almacenan los archivos que se compartirán.

Ejemplo de configuración en docker-compose.yml:

```
environment:
- GRPC_PORT=50051
- USERNAME=peer1
- PASSWORD=12345
```

E. Detalles de la organización del código por carpetas



F. Mini guía de cómo un usuario utilizaría el software o la aplicación

- Clonar el repositorio del proyecto.
- Configurar el entorno virtual o utilizar Docker para construir y ejecutar los contenedores.
- Construir y ejecutar los contenedores con Docker:

  docker compose build

  docker compose up
- Interactuar con el sistema a través de la API REST para localizar archivos en los peers o utilizar gRPC para simular la descarga/carga de archivos entre nodos.

- El sistema se levantará en los puertos configurados para los peers y la API, permitiendo realizar solicitudes a través de HTTP o gRPC.

4. Referencias

La documentación de gRPC fue consultada para la correcta implementación del servicio de transferencia de archivos.

Se reutilizaron partes del código del archivo file_transfer.proto de un tutorial de gRPC.

Referencias a las siguientes librerías y documentación oficial:

Cpurta. (s. f.). *GitHub - cpurta/p2p-grpc: Simple p2p network using gRPC*. GitHub. https://github.com/cpurta/p2p-grpc

Documentation. (s. f.). gRPC. https://grpc.io/docs/

LondonComputadores. (s. f.). *GitHub - LondonComputadores/grpc-python-flask-microservices.* GitHub. https://github.com/LondonComputadores/grpc-python-flask-microservices

NetMentor. (2024, 10 septiembre). *gRPC para desarrolladores: Explora su poder y cómo aplicarlo en C#* [Vídeo]. YouTube. https://www.youtube.com/watch?v=7g7-_1Fztmc