

INSTITUTO POLITECNICO NACIONAL

**UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA
Y TECNOLOGIAS AVANZADAS - UPIITA**

EXTRAORDINARIO

Materia: Aplicaciones Distribuidas 24/1

Docente: Dr. Félix Rivera Mata

Alumno: Gallardo Cervantes Mario Abraham

Grupo: 4TV3

Fecha: 17/01/2024

INDICE

INTRODUCCIÓN	3
Servicios web/HTTP	3
Tolerancia a fallos	3
Herramientas	4
Web sockets.....	4
Flask	4
Web sockets.....	5
DESARROLLO	5
PRUEBAS Y RESULTADOS	6
OBSERVACIONES Y CONCLUSIONES FINALES	9
REFERENCIAS	10

INTRODUCCIÓN

Servicios web/HTTP

Los servicios web y HTTP son componentes fundamentales de la web. Los servicios web son aplicaciones que proporcionan funcionalidad a otras aplicaciones o sitios web a través de Internet. HTTP (Hypertext Transfer Protocol) es el protocolo que permite a los servicios web comunicarse entre sí e intercambiar datos. Es como un lenguaje que utilizan los ordenadores para hablar entre sí. Sin HTTP, no podríamos acceder ni interactuar con sitios y aplicaciones web.

Para transferir datos en volumen, los servicios web pueden utilizar diversos protocolos y tecnologías, como:

1. HTTP: HTTP es la base de la web y se utiliza para transferir datos entre servidores y clientes web. Admite varios métodos para enviar datos, como GET, POST, PUT y DELETE.
2. HTTPS: HTTPS (HTTP Secure) es una extensión de HTTP que añade una capa extra de seguridad cifrando los datos que se transmiten. Esto es especialmente importante cuando se transfieren datos sensibles, como información financiera o datos personales.
3. FTP (Protocolo de Transferencia de Archivos): FTP es un protocolo para transferir archivos a través de Internet. Se utiliza habitualmente para transferir grandes cantidades de datos, como imágenes, vídeos y documentos.
4. SFTP (Protocolo Seguro de Transferencia de Archivos): SFTP es una versión segura de FTP que cifra los datos que se transmiten. Suele utilizarse para transferir datos sensibles, como información financiera o datos personales.
5. API (Interfaz de Programación de Aplicaciones): Una API es un conjunto de reglas y protocolos que permite a diferentes sistemas de software comunicarse entre sí. Las API pueden utilizarse para transferir datos entre servicios web y suelen emplearse para integrar diferentes sistemas y servicios.
6. WebSockets: WebSockets es un protocolo que permite la comunicación bidireccional entre servidores web y clientes web. Puede utilizarse para transferir datos en tiempo real, como actualizaciones en directo o chat en directo.

Tolerancia a fallos

La tolerancia a fallos en servicios web se refiere a la capacidad de un sistema de software o aplicación web para continuar funcionando correctamente y proporcionar una experiencia de usuario aceptable, incluso cuando se producen fallos o errores en el sistema. La tolerancia a fallos es importante en servicios web porque los usuarios esperan que los sitios web y las aplicaciones web sean disponibles y funcionales en todo momento. Si un sitio web o aplicación web se queda atascado o falla constantemente, los usuarios pueden perder la confianza en la plataforma y buscar alternativas.

Para implementar una buena tolerancia a fallos en servicios web, se pueden seguir algunos best practices, como:

1. Diseñar un sistema escalable: Diseñar un sistema que pueda escalar fácilmente y manejar un gran tráfico de usuarios y solicitudes.

2. Implementar mecanismos de recuperación de errores: Implementar mecanismos de recuperación de errores para que el sistema pueda reaccionar y recuperarse de los fallos de manera eficiente.
3. Implementar monitoreo y seguimiento: Implementar un sistema de monitoreo y seguimiento para detectar fallos y errores en tiempo real y tomar medidas para solucionarlos.
4. Implementar pruebas de fallos: Implementar pruebas de fallos para detectar y solucionar problemas antes de que afecten a los usuarios.
5. Utilizar tecnologías de redundancia: Utilizar tecnologías de redundancia para garantizar que el sistema siga funcionando incluso si se produce un fallo en una parte del sistema.

Herramientas

Web sockets

Los web sockets son una potente herramienta para crear aplicaciones distribuidas. Permiten la comunicación bidireccional en tiempo real entre el cliente y el servidor, lo que hace posible una experiencia de usuario más interactiva y con mayor capacidad de respuesta. Estas son algunas de las principales ventajas de utilizar web sockets para aplicaciones distribuidas:

1. Comunicación en tiempo real: Los web sockets permiten la comunicación en tiempo real entre el cliente y el servidor, permitiendo actualizaciones y respuestas instantáneas. Esto es especialmente útil para aplicaciones que requieren intercambio de datos en tiempo real, como aplicaciones de chat, juegos y actualizaciones en directo.
2. Comunicación bidireccional: A diferencia de las peticiones HTTP tradicionales, que son unidireccionales, los web sockets permiten la comunicación bidireccional. Esto significa que el cliente y el servidor pueden enviarse datos en cualquier momento, lo que permite aplicaciones más interactivas y con mayor capacidad de respuesta.
3. Baja latencia: Los web sockets pueden proporcionar una latencia más baja en comparación con las peticiones HTTP tradicionales, ya que no requieren la sobrecarga de establecer una nueva conexión para cada petición. Esto puede mejorar el rendimiento de las aplicaciones en tiempo real y reducir el tiempo de espera percibido por los usuarios.
4. Escalabilidad: Los web sockets pueden escalarse fácilmente para acomodar un gran número de conexiones concurrentes, lo que los hace adecuados para aplicaciones de alto tráfico.

Flask

Flask es un micro framework web en Python que permite construir aplicaciones web. Es ligero y flexible, por lo que es una opción popular para proyectos de desarrollo web.

Para manejar peticiones HTTP en Flask, puedes utilizar el objeto ``request`` que proporciona el framework. El objeto ``request`` contiene información sobre la solicitud HTTP actual, como el método de solicitud (por ejemplo, GET, POST, PUT, DELETE, etc.), la URL de la solicitud, las cabeceras de la solicitud, y el cuerpo de la solicitud (si procede).

Web sockets

Los sockets web en Flask pueden implementarse usando la librería `websocket-flask`. Esta librería proporciona una forma sencilla de establecer y gestionar sockets web en aplicaciones Flask.

Para usar web sockets en Flask, necesitarás instalar la librería `websocket-flask` usando `pip install websocket-flask`. Una vez instalada la librería, puedes usarla en tu aplicación Flask importándola y creando un objeto `WebSocket`.

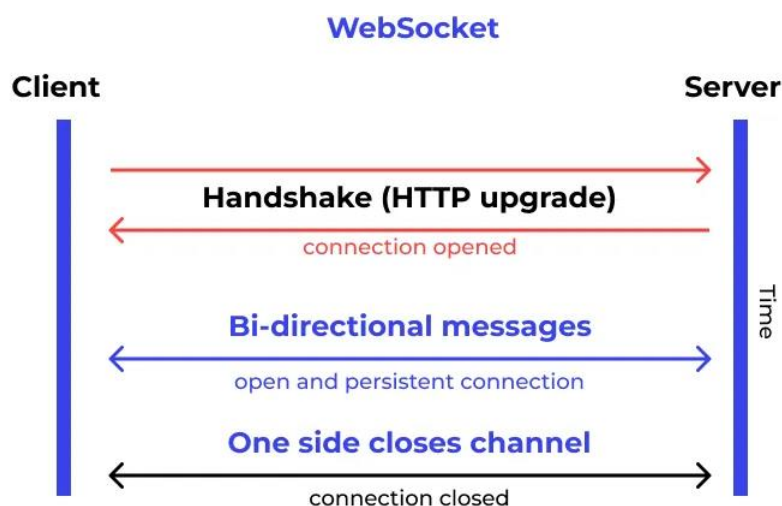


Tabla 1. Códigos utilizados en la realización del proyecto.

ID	Descripción	Referencia
Codigo 1.1 Archivo app.py Ubicado en la carpeta final-project	Código que se encarga de las rutas del servidor web.	Código adaptado del sitio: https://github.com/migfel/Apps-Distribuidas
Codigo 1.2 Archivo ServidorMP3Video.py Ubicado en la carpeta final-project del repositorio "distributed-applications"	Servidor con web sockets para la transferencia de video.mp4	Código adaptado del sitio: https://github.com/migfel/Apps-Distribuidas
Codigo 1.3 BalanceadorDeCarga.py Ubicado en la carpeta del repositorio	Balanceador de carga para manejar las peticiones de Cliente(Balanceador).py	Código adaptado del sitio: https://github.com/migfel/Apps-Distribuidas
Codigo 1.4 Cliente(Servidor).py Ubicado en la carpeta del repositorio	Solicitud del cliente desde terminal.	Codigo 1.3 BalanceadorDeCarga.py Ubicado en la carpeta del repositorio

DESARROLLO

En la figura 1 se muestra la arquitectura que tiene la aplicación, en primera instancia tenemos a los usuarios que desde su máquina cargan el archivo que desean transferir, el archivo se almacenará en el servidor y posteriormente un cliente hará la petición para descargar el archivo en su ordenador.

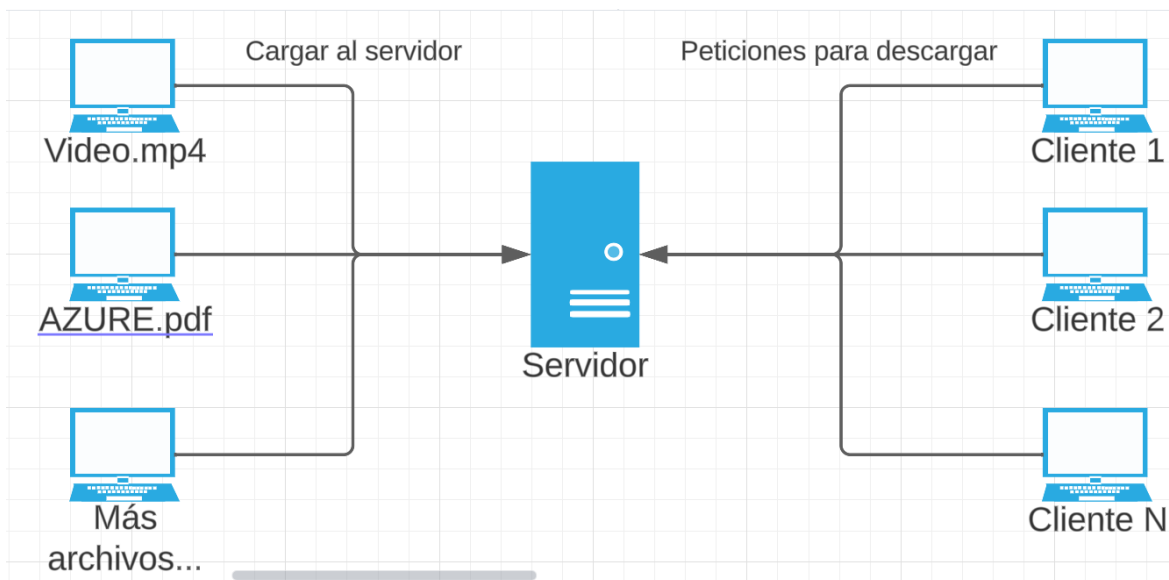


Figura 1.

PRUEBAS Y RESULTADOS

Producto	No se logro funcionalidad	Motivo o Circunstancia
Carga de documentos al servidor	Sí se logró cualquier tipo de documento pero de cualquier tamaño	No se parametrizó
Descarga del documento	No se logró hace stream d video durante la descarga	Rápida descarga

Ejecución

1. Inicar aplicación Flask

Mediante el comando `$flask run` se iniciará la ejecución de la aplicación:

```
~/De/D/U/distributed-applications/final-project master !2 ?1 flask run
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a pro
duction WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Figura 2.

2. Ir a 127.0.0.1:5000 y cargar un documento

En la figura 3 se muestra la interfaz de usuario donde se debe cargar el documento que se desea transferir:



Figura 3.



Figura 4.

En la figura 4 podemos observar que una vez que se carga el documento se almacena en la carpeta **/uploads**, para que posteriormente sea transferido el archivo.

3. Transferencia de video mediante ServidorMP3Video.py

En la figura 5 observamos que el web socket del servidor ya está listo para funcionar y en la figura 6 se accede a la ruta de flask **/client**, que al acceder a esta ruta, se realiza la descarga del video.mp4:

```
~/De/D/U/distributed-applications/final-project master !2 ?2 python3 ServidorMP3Video.py
Servidor esperando conexiones...
```

Figura 5.

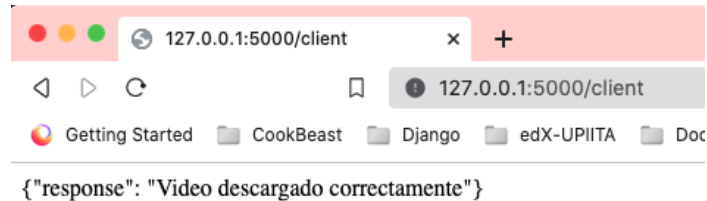


Figura 6.

En la figura 7 se muestra la carpeta /uploads, donde se cargan los archivos para transferir, y la carpeta principal, el destino final de los archivos transferidos.

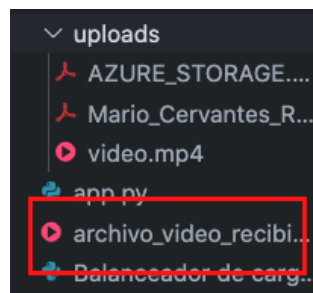


Figura 7.

4. Iniciar Balanceador de Carga

Creación de hilos a través de la dirección 127.0.0.1/12346:

```
~/De/D/U/distributed-applications/final-project ma
ster !2 ?2 python3 Balanceador\ de\ carga.py
* Serving Flask app 'Balanceador de carga'
* Debug mode: off
WARNING: This is a development server. Do not use it in a produc
tion deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:12346
* Running on http://192.168.0.18:12346
Press CTRL+C to quit
```

Figura 8.

5. Hacer petición del cliente desde la terminal


```
~/De/D/U/distributed-applications/final-project mast
er !2 ?2 python3 Cliente\ \balanceador\).py
{'message': 'Redirección iniciada'}
```

Figura 9.

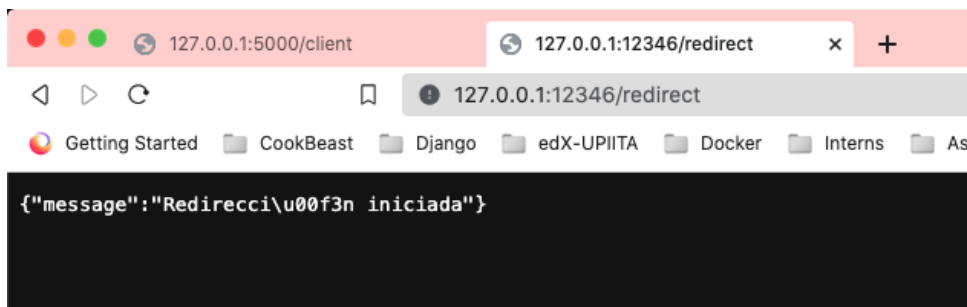
```
Archivo archivo_video_recibido.mp4 recibido.
Conexión cerrada.
Servidor esperando conexiones...
Conexión establecida desde: ('::1', 53074, 0,
0)
```

Figura 10.

En la figura 10 vemos que al ejecutar el *ClienteBalanceador.py*, se hace una petición por medio de un hilo a la dirección **127.0.0.1:5000/client**

6. Hacer petición del cliente desde la web

Aquí la petición se realiza desde el navegador al entrar a la url **127.0.0.1:12346/redirect**



The screenshot shows a web browser window with two tabs. The active tab is titled '127.0.0.1:12346/redirect'. The address bar shows the URL '127.0.0.1:12346/redirect'. Below the address bar, there is a navigation bar with several folder icons labeled 'Getting Started', 'CookBeast', 'Django', 'edX-UPIITA', 'Docker', 'Interns', and 'As'. The main content area of the browser displays a JSON message: `{"message": "Redirecci\u00f3n iniciada"}`.

Figura 11.

OBSERVACIONES Y CONCLUSIONES FINALES

1. Los WebSockets proporcionan un canal de comunicación bidireccional entre el cliente y el servidor, permitiendo la comunicación en tiempo real y posibilitando aplicaciones más interactivas y con mayor capacidad de respuesta.
2. La tolerancia a fallos es crucial en las aplicaciones basadas en WebSocket, ya que un solo fallo puede provocar la pérdida de toda la conexión.
3. Existen varias técnicas para lograr la tolerancia a fallos en las aplicaciones basadas en WebSocket, entre ellas:
 - Equilibrio de carga: distribución de las conexiones WebSocket entrantes entre varios servidores para mejorar la disponibilidad y la escalabilidad.
 - Estrategias de failover y fallback: implementar estrategias de failover y fallback para gestionar fallos y mantener la integridad de la aplicación.
4. Los WebSockets ofrecen varias ventajas para los sistemas tolerantes a fallos, entre las que se incluyen:

- * Escalabilidad mejorada: WebSockets puede manejar un gran número de conexiones concurrentes, lo que los hace adecuados para aplicaciones de alto tráfico.
- * Mejor rendimiento: Los WebSockets proporcionan una comunicación de baja latencia y en tiempo real, lo que resulta crítico para muchas aplicaciones.
- * Mejora de la experiencia del usuario: Los WebSockets permiten aplicaciones más interactivas y con mayor capacidad de respuesta, lo que puede mejorar la experiencia del usuario.

REFERENCIAS

- Liu, Q., & Sun, X. (2012). Research of web real-time communication based on web socket.
- Grinberg, M. (2018). *Flask web development: developing web applications with python*. " O'Reilly Media, Inc."
- Tanenbaum, A. S., Guerrero, G., & Velasco, Ó. A. P. (1996). *Sistemas operativos distribuidos* (No. QA76. 76063. T35. 3 1996.). México;; Prentice Hall.
- Mullender, S. (Ed.). (1990). *Distributed systems*. ACM.