Proyecto Final de Sistemas Distribuidos

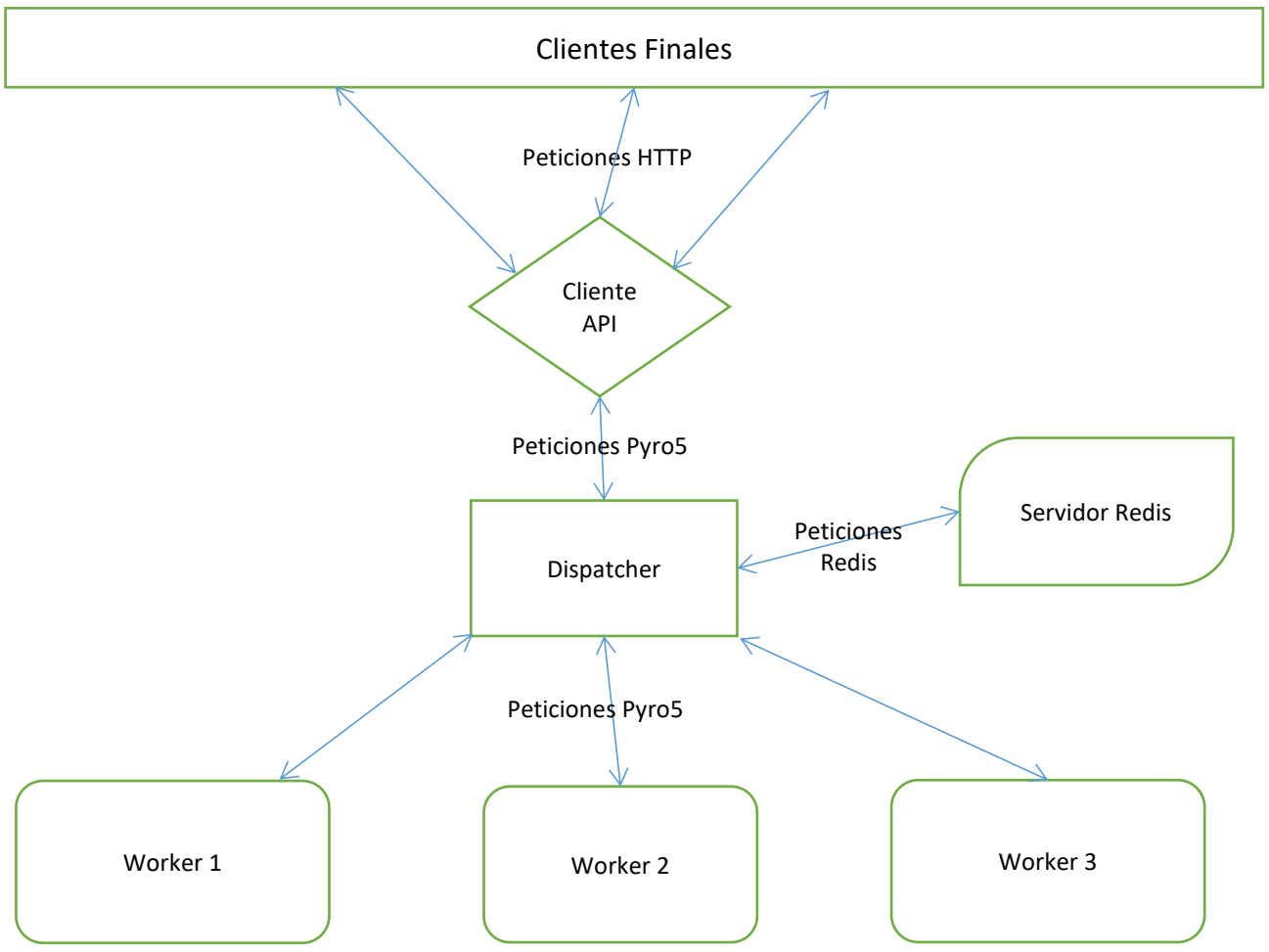
Scrapper

Integrantes:
Deborah Famadas Rodríguez C412
Gabriel Hernández Rodríguez C411
David Manuel García Aguilera C411

Dependencias fundamentales:

python>=3.10.10: para ejecutar y programar el proyecto
Fire>=0.5.0: cuando se crea un nuevo nodo maneja la creación de las instancias y ejecuta automáticamente el rol especificado.
requests>=2.28.2: se encarga de hacer las peticiones HTTP para escrapear páginas.
fastapi>=0.96.0: usado en la implementación del cliente API
Pyro5>=5.14: se encarga de implementar el protocolo RPC de comunicación entre los nodos.
redis>=4.5.5: sirve de base de datos para guardar datos como la caché de páginas escrapeadas y la lista de URLs pendientes.

Arquitectura del proyecto:



La comunicación entre los nodos de todo el sistema distribuido, es decir, sin contar los usuarios finales, sigue un modelo RPC. Está implementado de forma automática por el módulo Pyro5 entre el dispatcher, y los workers y el cliente API, y principalmente consiste en llamar funciones del dispatcher y usar los resultados recibidos para cumplir con un rol. La comunicación entre el dispatcher y el servidor Redis está manejada por el paquete Redis y solo consiste en realizar los cambios necesarios en su base de datos.

Flujo de ejecución de la petición de escrapear una página:

- se genera la petición en los clientes finales, la que se transforma en una petición HTTP al cliente API. Digamos que una URL a escrapear es una *query*.
- en el cliente API se reciben todas las queries y por cada petición se genera una conexión con el dispatcher, por la cual primero se pide escrapear una página y se espera hasta que se termina de procesar. Esta última parte se gestiona por la comunicación Pyro5 anteriormente mencionada.
- en el dispatcher se recibe cada query y puede suceder uno de los siguientes casos:
    - de la URL a escrapear ya se tiene en caché un resultado, este se devuelve directamente
    - no se tiene ningún resultado en caché para esta URL, en este caso se pone al final de la cola de URLs pendientes
- cuando la query ya está en la primera posición de la cola de pendientes y un worker la pide, este la procesa, entiéndase que hace la petición a través del módulo requests de escrapear la página, y le retorna el HTML de la página además del status de la petición de escrapeo. A partir de este momento comienza el proceso en reversa. Notar que cuando se dice que se retornan estos valores a través de Pyro5, se refiere a que estos son serializados y deserializados en el proceso y que el dispatcher al final recibe los resultados esperados.
- el dispatcher cuando recibe la respuesta procedente del worker crea/sobreescribe la caché de esta URL guardando tanto el cuerpo HTML como el estatus del proceso de escrapear.
- el cliente API, se da cuenta de que lo anterior terminó de ejecutarse al recibir una que existe una entrada en la caché y este resultado es lo que devuelve a los clientes finales

Sobre los usuarios del proyecto:

El admin que instala el proyecto es el encargado de poner a correr los nodos del sistema (cliente API, el dispatcher, el servidor Redis y los workers). Este usuario también puede configurar las variable de entorno del sistema, de la forma que se configuren las variables de sistema, sea virtualenv o Docker. Algunas de la variables a configurar son:

- HOSTNAME, HOSTPORT: define el ip y el puerto del host en un nodo del sistema, especialmente para el cliente API y el dispatcher sirve para definir por cual puerto escuchan
- DISPATCHER, DISPATCHER_PORT: configura el ip y el puerto del dispatcher al cual conectarse por parte de otro nodo del sistema. Análogamente BACKUP_DISPATCHER y BACKUP_DISPATCHER_PORT definen el ip y el puerto del dispatcher de respaldo a usar.
- API_HOST, API_PORT: configura el ip y el puerto del cliene API al cual conectarse por parte de un cliente final del sistema.
- REDIS_URL, REDIS_HOST, REDIS_PORT: configura la dirección del servidor Redis para los nodos dispatcher. Análogamente BACKUP_REDIS_URL, BACKUP_REDIS_HOST, BACKUP_REDIS_PORT configuran la dirección del servidor Redis de respaldo.
- MBB_RETRIES, MBB_TIME: configura la cantidad de intentos reconexión y el tiempo de espera entre cada intento de reconexión al dispatcher y al servidor Redis.

Los clientes finales pueden hacer dos tipos de peticiones al sistema:
- fetch: pide el contenido de una URL, puede devolver el contenido guardado en caché o escrapear la página si no está.
- reset: pide borrar la caché de una URL en especifico, permitiendo que al realizar una nueva petición fetch se tenga el resultado actualizado.

Features:

El sistema diseñado tiene las siguientes características de utilidad:
- **estabilidad**: se puede desconectar cualquier componente del sistema que los demás esperarán a que esté disponible para continuar el trabajo. Las queries no se pierden al ser tragadas por un worker que pierda conexión y pare su ejecución, sino que puestas de vuelta en la cola de pendientes y son procesadas luego por algún worker.
- **dispatcher y servidor Redis de respaldo**: cuando alguno de estos componentes no están disponibles, se comienza a trabajar con el de respaldo. También relacionado con la estabilidad del sistema.

Otra característica interesante es que se usaron los locks provistos por el paquete Redis para evitar problemas de concurrencia.