Trabajo práctico N°1: Logging-Pipeline

Sistemas Distribuidos I (75.74)

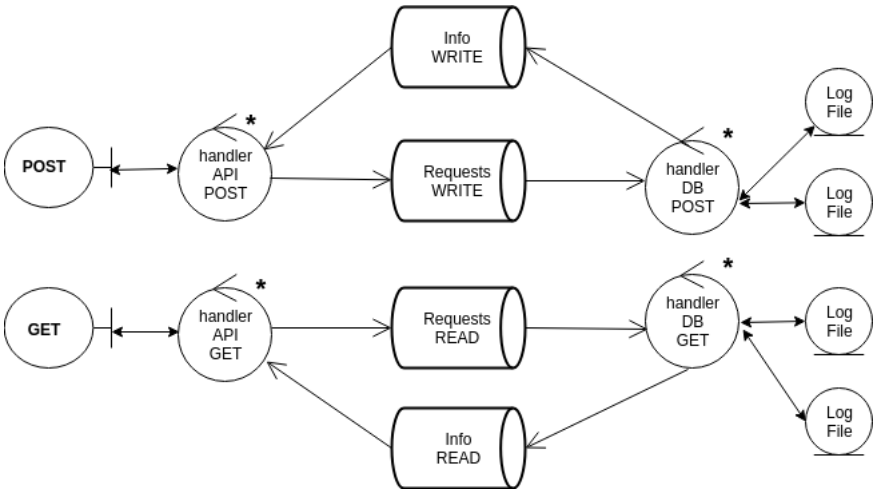
Ana Czarnitzki

11 de abril de 2019

Para este trabajo se solicitó el diseño y la implementación de un sistema distribuido que ofrezca un servicio HTTP para el guardado y consulta de logs generados por diferentes aplicaciones. Se espera que el sistema responda a una cantidad masiva de pedidos y de escrituras y que se encuentre disponible en todo momento.
Los logs guardados cuentan con un mensaje, una fecha y un conjunto de tags y deben poder pedirse al sistema por estos atributos.

1. Arquitectura

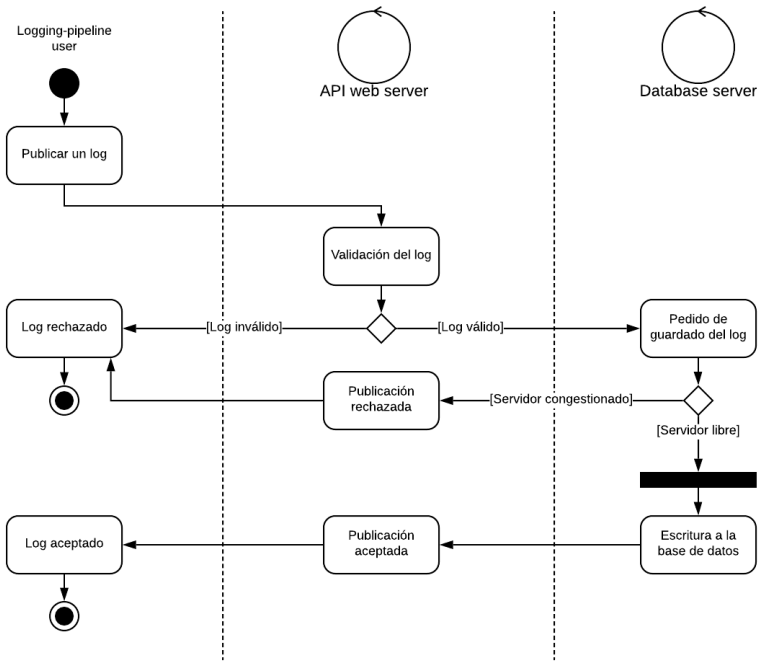
La arquitectura propuesta para este trabajo fue la siguiente:



Debido a que la prioridad del sistema es estar disponible en todo momento y brindar *fairness*, se dividió al mismo en dos conjuntos de procesos, uno para recibir los *POSTs* (pedidos de guardado de un log) y otro para recibir los *GETs* (consulta de logs). Los *handlers* tanto del lado del servidor web como del lado

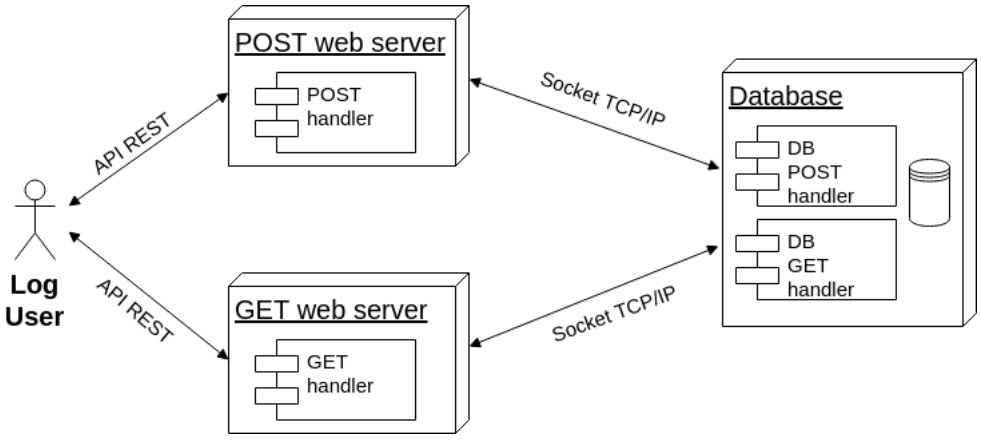
del de la base de datos pueden ser múltiples, permitiendo responder a varios llamados a la vez. En caso de congestión, se descartan los pedidos en las colas que van desde los *handlers* de la *API* a la base de datos.
Por otro lado, para mejorar los tiempos de escritura y lectura a memoria, se optó por guardar archivos diferentes para cada aplicación que haga escrituras al sistema. Además, se tiene esa misma información agrupada tanto por *tags* como por fechas, replicando la misma para evitar tener que hacer *locks* masivos y agilizando los pedidos.

El flujo para publicar un log se puede visualizar en el siguiente diagrama de actividades. El flujo para obtener un conjunto de logs es análogo a este primero.



2. Despliegue e implementación

El despliegue de los distintos procesos en este trabajo fue la siguiente:



En particular, se utilizaron tres containers de *Docker*, uno para cada uno de los servidores web y otro para los procesos relacionados con la base de datos. Esto último fue así para facilitar el guardado y la lectura de un mismo archivo. El usuario se comunica con los servidores webs mediante una *API REST* y estos con la base de datos con *sockets TCP/IP* que implementan un protocolo simple propio. El descarte de mensajes se hace de forma implícita fijando un máximo de conexiones a aceptar por parde los servidores de la base de datos.
Por otro lado, debido a la falta de tiempo, los dos servidores web están hechos en *flask* en vez de estar implementados desde cero y los handlers de la base de datos son hilos en vez de procesos.

2.1. Código

El código almacenado en el repositorio de github se divide en tres módulos principales:

- common: código que utilizan tanto los servidores web como el de la base de datos, contiene librerías de sockets que manejan el protocolo entre los dos servidores y objetos de tipo *wrapper* para simplificar el manejo de la información de los *requests*.
- api: contiene el código de los servidores webs, sencillamente son dos servidores *flask*, un parser y funciones de validación.
- db: contiene el código de los servidores de la base de datos, incluye el código de los dos servidores que aceptan conexiones entrantes y de los *workers* que escriben y leen a la base de datos. Además, se provee una abstracción para los archivos que guardan logs y para el conjunto de los mismos, que implementan lecturas y escrituras *thread safe*.

Todo el código se encuentra documentado en el repositorio y cuenta con un pequeño *readme* que explica el uso del mismo.

3. Mejoras y trabajo futuro

A continuación se listan las posibles mejoras para este trabajo.

- Cambiar los servidores webs *flask*. Esto implica la implementacion del protocolo *HTTP REST* y un manejo de procesos o hilos equivalente al ya hecho para los servidores de la base de datos.
- Utilizar procesos en vez de hilos en el servidor de la base de datos. El mayor trabajo requerido para este cambio es en el manejo de *locks*, razón por la cuál en esta entrega se utilizaron hilos.
- Utilizar un pool de containers tanto para los servidores web como para los de la base de datos, para esto se necesitaría:
  - Para los servidores web, la creación de un pequeño cliente que realice *load balancing* para los mismos.
  - Para la base de datos, generar una lógica de *load balancing* en los servidores web. La forma más sencilla sería que al recibir respuestas de un servidor, se lo guarde en una cola de servidores disponibles los que en los siguientes pedidos se llamará. Este balanceo sólo podría usarse para la escritura, en el caso de la lectura se deberá hacer pedidos a todos los servidores y luego juntar la información.
- Mejora en el filtrado por fechas. En este momento la organización de archivos por fecha mejora los tiempos de *lock* pero no tiene optimizaciones para la obtención de los archivos requeridos en cada caso, para esto se podría, por ejemplo, se podría tener guardados a los mismos en una lista ordenada y realizar búsqueda binaria. Esto no ocurre para los *tags* dado que se accede a uno en particular y no a un rango de estos.
- Mejora en el filtrado por patrones. En este momento se realiza una búsqueda líneal sobre los mismos en los servidores web, esto podría hacerse de forma concurrente.
- Extender la funcionalidad de filtrado, permitiendo filtrar por más de un tag o buscar el patrón no sólo en los mensajes de los logs.
- Evitar el almacenamiento de información repetida o proponer una mejor estructura para el guardado de los archivos de logs.
- Realizar *load balancing* inteligente y dinámico para el guardado en los archivos.