
Análisis de logs escalable.

Álvaro García Jaén

Antonio Molner Domenech

Índice

Un problema a solucionar...	3
¿Que es Log Analysis?	3
Importancia	4
Problemas con el análisis en tiempo real	4
Nuestra solución	4
Arquitectura y diseño	5
Implementación	7

Un problema a solucionar...

A lo largo de esta asignatura hemos visto como desplegar servidores web de altas prestaciones. El concepto que hace eco en nuestra cabeza seguramente sea «granja web». Tras usar esta técnica, una de las cuestiones a solucionar es: ¿cómo recopilo información de mi granja? A estas alturas es muy probable que todos tengamos claro que es un log y la importancia que tiene (si no es así, tranquilo, en el siguiente punto daremos un repaso para los más rezagados). Cuando tenemos un solo servidor, simplemente consultamos un único archivo que genera Apache HTTP, Nginx, [inserte aquí su servidor web]... para ver los logs pero, ¿y si nuestra granja web esta compuesta por 10 servidores finales? ¿Y si son 100? Esto rápidamente se nos va de las manos...

En esta memoria os presentaremos la solución que hemos dado nosotros y cómo lo hemos implementado. Obviamente hay muchas otras maneras de hacerlo y os invitamos a que investigueis al respecto.

¿Que es Log Analysis?

Cuando un programa está ejecutándose (este puede ser un servidor web, el propio sistema operativo...) pasan cosas constantemente (clientes abriendo la página web, inicios de sesión, crear un nuevo archivo...). Llamaremos a esto eventos. Estos eventos se almacenan en unos archivos que reciben el nombre de «archivos de log».

En los archivos de log, como ya hemos dicho, podemos encontrar un registro de los eventos que han sucedido en ciertos programas (obviamente cada programa tiene su propio archivo, si no esto sería un caos, aunque hay maneras de consultar todos los logs del sistema de manera simultanea). En el caso que nos interesa a nosotros, los servidores web, veremos información de las peticiones HTTP que han realizado los diferentes clientes. Obtendremos información como la IP desde la que el cliente se conecta, qué parte de la web ha solicitado, el código de error que ha devuelto nuestro servidor (2xx, 3xx, 5xx...), a qué hora realizó la petición, el protocolo usado...

Toda esta información no viene dada en un formato estructurado como por ejemplo JSON, sino que es una línea de información que siempre muestra los mismo campos en el mismo orden y que nosotros deberemos ir extrayendo mediante parsing (Regex por ejemplo). A este tipo de datos se les llaman datos semiestructurados.

Importancia

Analizar los logs es sumamente importante. En la informática, como su propio nombre indica, la información es lo más importante. Cuando más información tengamos de cómo está funcionando nuestro servidor web, más podremos prevenir los futuros problemas. Por ejemplo, si vamos analizando los logs de nuestro servidor en tiempo real, al fin y al cabo estamos monitorizándolo. Podemos poner diferentes alertas, por ejemplo si un mismo cliente (misma IP) está intentando entrar a ciertas rutas que no paran de devolver un código 4xx (aquí se incluyen peticiones que resultan páginas prohibidas, que requieren autenticación...) y actuar al respecto, por ejemplo baneando dicha IP (prohibiendo que vuelva a conectarse más a nuestro servidor) o seguirle de cerca para tratar de averiguar quién es o qué quiere.

Problemas con el análisis en tiempo real

Lo que buscamos entonces es poder analizar estos logs en tiempo real para poder monitorizar bien nuestra granja. Para explicar bien esto introduciremos dos términos: cauce y latencia. Cauce es la cantidad de datos que procesas por unidad de tiempo. Latencia es el tiempo que tardas en procesar un log. Debemos de mantener un equilibrio entre estos términos a la hora de analizar nuestros datos. No tiene sentido ir procesando los logs cada segundo, ya que si no hay muchos clientes, iremos procesando los logs de uno en uno o ninguno; tampoco tiene sentido esperar a tener en cola 100 logs para entonces procesarlos, ya que quizás esperaremos demasiado y perderemos el «tiempo real».

Nuestra solución

Utilizaremos este punto para introducir nuestra solución. En primer lugar, dividiremos este problema en varias partes:

- **Ingestión.** La ingestión es el momento en el que cogemos los datos, los logs, y los movemos al servidor donde se procesarán. Para esto utilizaremos [Apache Flume](#) y [Apache Kafka](#). (No, Apache no patrocina esta memoria)
- **Procesamiento.** El procesamiento es el momento en el que tomamos los logs en crudo, tal y como los produce nuestro servidor web, y obtenemos la información que nos interesa de los eventos. Para esto utilizaremos [Apache Spark](#), [Apache Cassandra](#) y [Apache Hadoop](#) (HDFS).

- Presentación de la información. El visionado es el momento en el que se muestra la distinta información, ya digerida, en un panel para que puedan consultarse. Aquí pueden consultarse diferentes relaciones entre los eventos como el número total de visitantes, los diferentes códigos de error devueltos... Para esto utilizaremos [Dash \(by Plotly\)](#).

Vamos a comentar un poco por qué hemos usado este software y no otro:

En el caso de Kafka, cuenta con una interfaz genérica con todas las fuentes de información. Spark tiene una integración perfecta con Kafka y podemos conectarla directamente con HDFS. Además la paralelización es la base de su desarrollo. En cuanto a Flume, se conecta bien con Kafka y además permite muchas fuentes distintas (más adelante veremos esto).

Todas estas herramientas están pensadas para el BigData, por lo que escala tan bien como una granja web. Además todo es de Apache, por lo que el hecho de que se integre bien no es ninguna sorpresa.

No obstante, cabe mencionar que todas las partes son reemplazables. Kafka podría reemplazarse por RabbitMQ, Spark por Storm (o tu propia implementación)... Aunque tal y como lo hemos hecho todo funciona bien, casi de manera automática. Cassandra podríamos sustituirlo por HBase o cualquier base de datos NoSQL. El panel también podría cambiarse por implementaciones [InfluxDB](#), que aunque te facilita mucho el desarrollo, hace que pierdas cierta flexibilidad.

Arquitectura y diseño

Una vez que tenemos una idea aproximada de cual será el flujo de datos, lo aclararemos con una imagen. Como vemos en la Figura 1, en pocas palabras nuestro diseño hará lo siguiente:

- Un agente de Flume (instalado en cada uno de los servidores finales) enviará los logs a Kafka.
- Spark tomará la información que recibe Kafka y la irá almacenando en crudo en HDFS. También procesará esta información, estructurándola y almacenándola en una base de datos Cassandra.
- El Dashboard irá tomando la información de Cassandra para presentar diferentes datos que nos interesan.

Ahora veremos qué hace exactamente cada programa y como nos permite realizar esto.

- Apache Flume. Servicio que nos permite coger, agregar y mover grandes cantidades de logs de manera eficiente. Tiene una arquitectura simple basada flujo de datos. En la Figura 2 podemos ver como funciona: Un agente toma la información de una fuente (por ejemplo

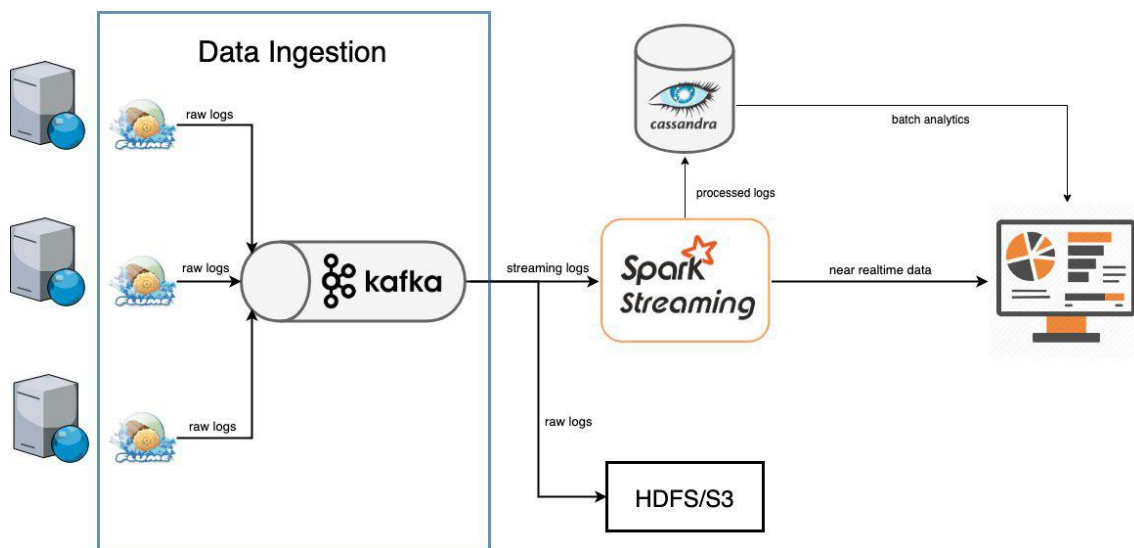


Figura1: Diseño que implementaremos

leer un archivo), la mueve mediante un canal (por ejemplo la memoria principal) y lo deposita en una pila (por ejemplo otro servicio como Kafka).

- Apache Kafka. Plataforma de streaming distribuido. La idea básica es que tenemos canales, llamados «topics», por donde fluirá la información. Hay dos tipos de usuarios, los productores (en nuestro caso los agentes Flume) que enviarán la información y los consumidores que serán los que tomarán esa información. Además tiene particiones, lo cual da paralelismo tanto a los consumidores como a los productores sobre el mismo topic.
- Apache Spark. Framework open-source que permite hacer cómputo en clusters, en nuestro caso para procesar logs de manera rápida y eficiente.
- Apache Cassandra. Sistema de almacenamiento de bases de datos NoSQL. Permite manejar grandes cantidades de datos de manera eficiente y escalable. Lo utilizaremos para almacenar la información ya procesada, para luego ser mostrada a nuestra voluntad en el panel.
- Apache Hadoop. Colección de herramientas que facilita el manejo de grandes cantidades de datos. Nosotros más concretamente proponemos HDFS, que es un sistema de archivos distribuidos basado en Java para almacenar grandes cantidades de datos. Esto es esencial en entornos de producción donde es necesario almacenar los logs en crudo, sin procesar.
- Dash by Plotly. Framework escrito en Python que nos permite de manera sencilla y elegante desarrollar aplicación web para analisis. Es así como presentamos los datos una vez han

sido procesados.

Implementación

Hablar sobre la configuración de los clusters, agentes, y base de datos. Hablar del consumo de datos por parte del Dashboard (peticiones a cassandra y consumo de Kafka).

Para implementar esta arquitectura y el diseño que mostramos en la Figura 1, lo primero que hemos hecho es aprovisionarnos de servidores. Hemos utilizado los VPS (Virtual Private Server) de [DigitalOcean](#). Hemos elegido este proveedor porque cuenta con un plan de referidos en el que a nuevos clientes les dan 100 euros para gastar en servidores, por lo que podremos hacernos con un buen arsenal:

- 4 x Servidores web básicos finales. Aquí tendremos corriendo el agente de Flume y [un generador de logs](#) que simulará gran cantidad de tráfico.
- 1 x Servidor para procesar toda la información. Se trata de un servidor más potente (4 núcleos). Es por eso que hemos escogido usar 4 servidores web finales, para aprovechar al máximo el paralelismo. Aquí estará Kafka, Spark, Cassandra y el Dashboard. En nuestro ejemplo no instalaremos HDFS por motivos de espacio. En un entorno de real cada uno de estos servicios correría en servidores diferentes y sería obligatorio, como ya comentamos anteriormente, el uso de HDFS para almacenar los logs en crudo.

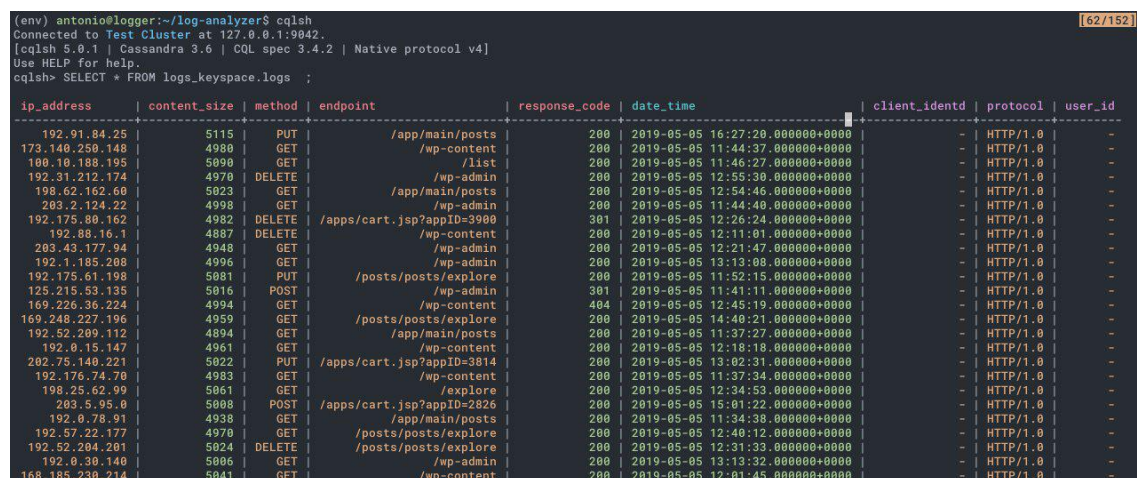
Para llevar a cabo la instalación, comenzaremos por configurar los servidores web (aquí lo explicaremos de manera muy general, cada uno luego podrá consultar de manera más específica lo que tenga dudas viendo [el repositorio](#)). Para ello pondremos primero a generar los logs de manera automática. Una vez que se están generando, debemos instalar Apache Flume (que lamentablemente no está en los repositorios, tendremos que descargar el código fuente e instalarlo nosotros mismos). Una vez instalado y configurado para que envíe los logs a nuestro servidor de procesamiento, lo dejaremos a la espera de ejecutar (primero debemos instalar y configurar Kafka).

Cuando ya tenemos todos los servidores web preparados, nos vamos al servidor de procesamiento. Aquí en primer lugar instalaremos Apache ZooKeeper para facilitarnos la instalación. Una vez hecho esto, simplemente iniciaremos el servicio. Es turno de instalar Apache Kafka. Todos estos servicios, al igual que Flume, no están en el repositorio sino que debemos descargarlo de la web de Apache. Tras instalarlo, debemos crear los topics por donde fluirá la información (en nuestro caso dos, uno para Flume y otro para el Dashboard).

Una vez hecho esto, ya podemos iniciar Flume en los servidores web y veremos como empiezan a llegar los logs (se crea un archivo en /tmp). Como dato curioso, habia que añadir el hostname del servidor de procesamiento en el /etc/hosts de cada uno de los servidores web, aún no sabemos muy bien por qué.

Ha llegado el momento de procesar la información. Para ello en primer lugar instalaremos Cassandra, ya que será en una de sus bases de datos donde iremos guardando esta información. Para ello primero añadiremos el repositorio donde se encuentra para poder instalarlo con APT. Ejecutamos el servicio y ya podemos crear nuestra tabla. Para ello debemos crear un *key space* que será nuestro espacio de trabajo para cada tabla.

Cuando ya tenemos listo nuestro almacenamiento, debemos procesar la información. Para ello usaremos Spark, que se conectará a Kafka para recibir los batches (entradas de log) y procesarlos de forma paralela. Cada CPU tiene su conexión a la base de datos Cassandra para poder ir añadiendo entradas a la tabla de forma paralela. En la Figura 3 podemos ver un ejemplo de como se nos queda la tabla. En la Figura 4, podemos ver en tiempo real como Spark va recibiendo y procesando los diferentes batches.



```
(env) antonio@logger:~/log-analyzer$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.6 | CQL spec 3.4.2 | Native protocol v4]
Use HELP for help.
cqlsh> SELECT * FROM logs_keyspace.logs ;
```

ip_address	content_size	method	endpoint	response_code	date_time	client_id	protocol	user_id
192.91.84.25	5115	PUT	/app/main/posts	200	2019-05-05 16:27:20.000000+0000	-	HTTP/1.0	-
173.140.250.148	4980	GET	/wp-content	200	2019-05-05 11:44:37.000000+0000	-	HTTP/1.0	-
180.10.188.195	5090	GET	/list	200	2019-05-05 11:46:27.000000+0000	-	HTTP/1.0	-
192.31.212.174	4970	DELETE	/wp-admin	200	2019-05-05 12:55:30.000000+0000	-	HTTP/1.0	-
198.62.162.60	5023	GET	/app/main/posts	200	2019-05-05 12:54:46.000000+0000	-	HTTP/1.0	-
203.2.124.22	4998	GET	/wp-admin	200	2019-05-05 11:44:40.000000+0000	-	HTTP/1.0	-
192.175.80.162	4982	DELETE	/apps/cart.jsp?appID=3900	301	2019-05-05 12:26:24.000000+0000	-	HTTP/1.0	-
192.88.16.1	4887	DELETE	/wp-content	200	2019-05-05 12:11:01.000000+0000	-	HTTP/1.0	-
203.43.177.94	4940	GET	/wp-admin	200	2019-05-05 12:21:47.000000+0000	-	HTTP/1.0	-
192.1.185.200	4996	GET	/wp-admin	200	2019-05-05 12:13:00.000000+0000	-	HTTP/1.0	-
192.175.61.198	5081	PUT	/posts/posts/explore	200	2019-05-05 11:52:15.000000+0000	-	HTTP/1.0	-
125.215.53.135	5016	POST	/wp-admin	301	2019-05-05 11:41:11.000000+0000	-	HTTP/1.0	-
169.226.36.224	4994	GET	/wp-content	404	2019-05-05 12:45:19.000000+0000	-	HTTP/1.0	-
169.248.227.196	4959	GET	/posts/posts/explore	200	2019-05-05 14:40:21.000000+0000	-	HTTP/1.0	-
192.52.209.112	4894	GET	/app/main/posts	200	2019-05-05 11:37:27.000000+0000	-	HTTP/1.0	-
192.0.15.147	4961	GET	/wp-content	200	2019-05-05 12:18:18.000000+0000	-	HTTP/1.0	-
202.75.140.221	5022	PUT	/apps/cart.jsp?appID=3814	200	2019-05-05 13:02:31.000000+0000	-	HTTP/1.0	-
192.176.74.70	4983	GET	/wp-content	200	2019-05-05 11:37:34.000000+0000	-	HTTP/1.0	-
198.25.62.99	5061	GET	/explore	200	2019-05-05 12:34:53.000000+0000	-	HTTP/1.0	-
203.5.95.0	5008	POST	/apps/cart.jsp?appID=2826	200	2019-05-05 15:01:22.000000+0000	-	HTTP/1.0	-
192.0.78.91	4938	GET	/app/main/posts	200	2019-05-05 11:34:38.000000+0000	-	HTTP/1.0	-
192.57.22.177	4970	GET	/posts/posts/explore	200	2019-05-05 12:40:12.000000+0000	-	HTTP/1.0	-
192.52.204.201	5024	DELETE	/posts/posts/explore	200	2019-05-05 12:31:33.000000+0000	-	HTTP/1.0	-
192.0.30.140	5006	GET	/wp-admin	200	2019-05-05 13:13:32.000000+0000	-	HTTP/1.0	-
168.185.230.214	5041	GET	/wp-content	200	2019-05-05 12:01:45.000000+0000	-	HTTP/1.0	-

Figura2: Tabla de la BD Cassandra

Llegó el momento de presentar la información. Para ello implementaremos un panel sobre Dash. Cuando se genera una acción sobre el Dashboard, se genera la petición sobre la base de datos para obtener la información que se ha pedido. Tiene una parte de tiempo real en la que el panel obtiene los datos directamente de Spark que están en cola en Kafka, lo que nos permite ver todo en tiempo real en un intervalo de 5 a 10 segundos. En la Figura 5 podemos ver como nos ha quedado nuestro panel.

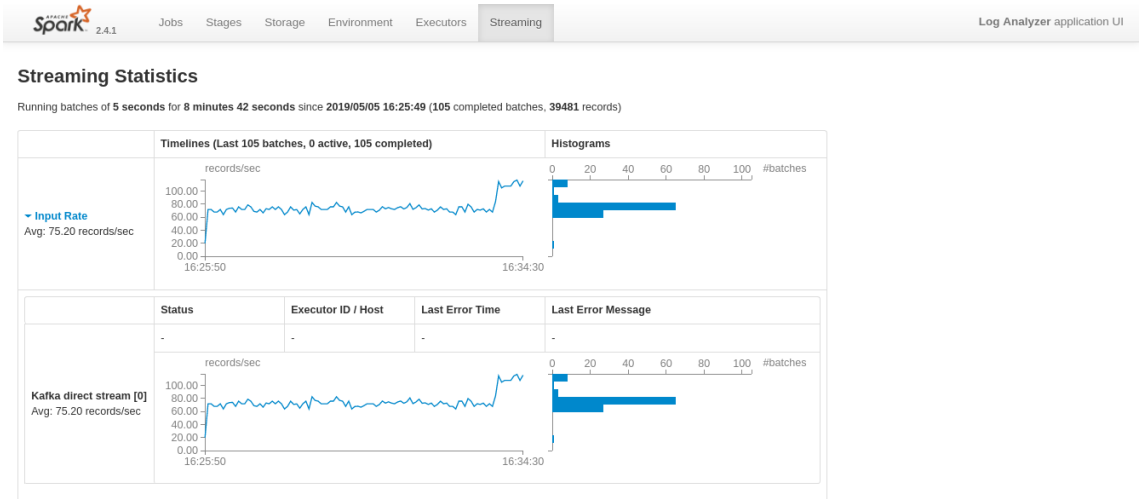


Figura3: Streaming de Apache Spark

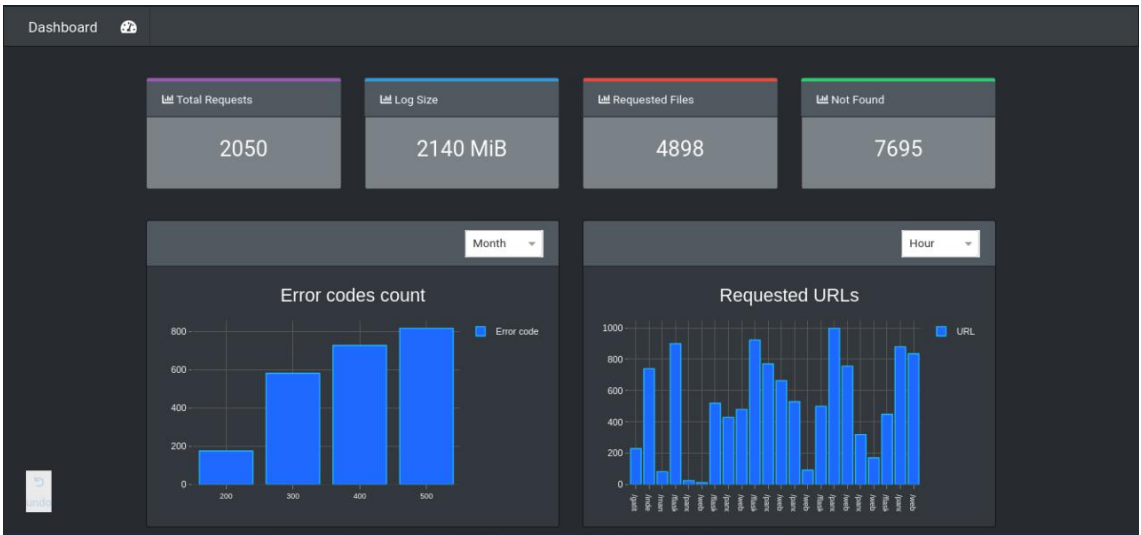


Figura4: Dashboard