Universitat Oberta de Catalunya

uoc.edu

Big data en seguridad

Carlos de la Cruz Pinto

Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones

Enric Hernández Jimenez

Victor García Font

04 de Junio de 2018

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Seguridad y Big Data*
Nombre del autor:	*Carlos de la Cruz Pinto*
Nombre del consultor/a:	*Victor García Font*
Nombre del PRA:	*Enric Hernández Jimenez*
Fecha de entrega (mm/aaaa):	06/2018
Titulación:	Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones
Área del Trabajo Final:	*TFM Ad-Hoc*
Idioma del trabajo:	*Español*
Palabras clave	*Big Data, Seguridad, Streaming*

El objetivo de este proyecto es, utilizando un escenario concreto real, desarrolla la posible implementación de un conjunto de servicios y aplicativos software relacionados con adquisición, transformación y análisis de cantidades masivas de datos de registro que sirvan como base para un sistema de monitorización que permita detectar anomalías en comportamientos que sirvan para conocer posibles intentos de accesos no autorizados a información o a sistemas, comportamientos no deseados, o intentos de explotación de vulnerabilidades. Se hace un análisis y una discusión de la viabilidad del uso para estos fines de diferentes herramientas relacionadas con Big Data, integración, computación distribuida y procesamiento de streams.

Abstract

The purpose of this project is, using a base real scenario, designing a viable implementation architecture for a set of services and software applications related to adquisition, transformation and analisys of massive amounts of log data, that then can be used for building a monitoring system for detecting anomalies in behaviours that may lead to detect possible unauthorized accesses to pieces of information or systems, undesired behaviours, or attempts to exploit systems vulnerabilities.

Several tools related to Big data, data integration, distributed computation and stream processing are also analyzed and discussed in the context of the project.

Índice

Contenido

1. Introducción ... 1
1.1 Contexto y justificación del Trabajo ... 1
1.3 Enfoque y método seguido ... 2
1.4 Planificación del Trabajo ... 3
1.5 Breve sumario de productos obtenidos ... 5
1.6 Breve descripción de los otros capítulos de la memoria ... 5
2. Primera etapa del pipeline: Extracción y entrega ... 6
3. Indexado de la información ingerida ... 16
4. Generación de la información de prueba: ... 20
5. Transformación de los datos ingeridos ... 22
6. Procesamiento de los datos y frameworks para computación distribuida y manejo de Big Data ... 25
7. Elaboración de la arquitectura para extracción, transformación y carga de la información ... 33
8. Glosario ... 45
9. Bibliografía ... 48
10. Anexos ... 49

1. Introducción

1.1 Contexto y justificación del Trabajo

En entornos productivos en los que es necesaria una garantía de integridad como la de un sistema de notaría electrónica, soluciones de monitorización tradicionales como por ejemplo PRTG, Nagios, Collectd + InfluxDB + Grafana, o sistemas de detección de intrusión no son suficientes.

Hay situaciones de uso del sistema que podrían proceder aparentemente de usuarios legítimos en situaciones normales, pero que puestos en contexto (por ejemplo, tras ser procesados por modelos de ML elaborados a tal efecto), podría descubrirse que ocultan usos no legítimos de un sistema.

Ejemplos de esto los tenemos en el sistema público de notificación telemática de sentencias judiciales (Lexnet), el cual a pesar de estar restringido solo a usuarios relacionados con el Sistema de Justicia, contaba con deficiencias que permitían que usuarios no autorizados a acceder a la información de un caso concreto pudieran aun así acceder a información relacionada con este. Detectar estas deficiencias sólo fue posible de forma manual, debido a que una persona realizó una serie de pruebas.

Un sistema automatizado con unas características similares a las que se describen en este proyecto hubiera podido ayudar a detectar estas anomalías y emitir una alerta en consecuencia (por ejemplo, si un usuario suele acceder diariamente a tres casos concretos durante la mañana desde un lugar concreto, podría considerarse una anomalía que acceda a cinco casos por primera vez desde una zona geográfica totalmente distinta a las seis de la tarde).

1.2 Objetivos del Trabajo

El objetivo de este trabajo es diseñar una arquitectura software para un sistema de ingesta y recogida de datos que pueda servir como plataforma para un sistema de detección de anomalías en un conjunto grande de servidores que, debido a lo sensible

de la información que albergan y los requerimientos de la legislación española en lo referente a notaría electrónica, no pueden estar alojados en infraestructuras cloud o beneficiarse de servicios ya existentes en este tipo de infraestructuras.

Así pues, se pretende conseguir:

- Hacer un estudio del estado del arte en tecnologías relacionadas con:
  - Transformación y enrutado de datos provenientes de logs
  - Streaming de datos y paso de mensajes
  - Procesado de datos previo a aplicación de algoritmos de ML
  - Frameworks integrados para Big Data

- Elaborar una arquitectura que permita crear un pipeline de transformación, almacenado y análisis de los datos generados en los servidores de las 3000 notarías existentes en el que se garantice la seguridad de la información.

  - Realizar una prueba de concepto de tecnologías que permitan desarrollar modelos para detección de anomalías utilizando técnicas de machine learning sobre los datos recolectaods.
  - Realizar una prueba de concepto de un sistema que permita realizar consultas ad-hoc contra los datos recolectados.

1.3 Enfoque y método seguido

La forma de proceder ha sido estudiar detalladamente las características de cada una de las tecnologías implicadas en el proceso, y valorar y describir tanto su escalabilidad como los requisitos secundarios y consecuencias económicas y de rendimiento que pueden generar escoger una u otra tecnología en cada una de las fases del pipeline.

Se han realizado pequeños tests y pruebas de concepto de varias de las tecnologías que se han considerado más relevantes para el proyecto, tras un estudio de las mismas.

1.4 Planificación del Trabajo

La planificación inicial estimada para el proyecto fue la siguiente:

Planificación temporal:

- Estudio de tecnologías y técnicas de transmisión/transformación/limpieza de logs (Flume, Logstash, Filebeat, Fluentd...) y de su desempeño, escalabilidad y viabilidad para el caso concreto. (10h)

- Estudio de tecnologías para el apoyo a la entrega de los eventos (Redis, Kafka...) (20h)
  - Estudio de desempeño, facilidad de escalabilidad y viabilidad de su aplicación. (10h)
  - Pruebas de integración entre logstash / flume / rsyslogd con Kafka / Redis. Valorar rendimiento / facilidad de despliegue y escalabilidad / recursos necesarios. (10 h)

- Estudio de herramientas de indexado y visualización de datasets (Lucene, Elasticsearch, kibana...) y su utilidad / aplicabilidad en este caso. (30h)
  - Justificación, si cabe, de la incorporación de estos elementos en la arquitectura (20h)
  - Prueba de conexión con el sistema de transmisión de información de logs. (10h)

- Estudio de herramientas de almacenaje, transformación y manipulación de Big Data (Hadoop /HDFS, Spark, Drill, Impala, Hive que permitan aplicar posteriormente modelos de machine learning utilizando diferentes lenguajes de programación. Valorar la posibilidad y utilidad de utilizar datastores NoSQL como MongoDB o Cassandra para almacenar los datos recolectados a largo plazo. (30h)

- Estudio de tecnologías de orquestación, clustering y despliegue automatizado para las herramientas anteriores. Tecnologías de contenedores genéricas (Docker, Kubernetes...) o de clustering más específicas, orientadas a ejecutar

Hadoop (Zookeeper), Distribuciones de Hadoop de terceros como Cloudera. (30h)

- Elaboración final de una arquitectura Software + Hardware que unifique todas las piezas escogidas en las fases anteriores, con una estimación de los costes y recursos que podría conllevar su explotación. (100h)

    - Estudio de el hardware mínimo que garantice una ejecución adecuada de la arquitectura propuesta. (10h)

    - Elaboración de una prueba de concepto que junte todas las piezas software escogidas para la arquitectura final en los pasos anteriores.
        - Elaboración de scripts de despliegue en entorno de pruebas local (por ejemplo Vagrant +Ansible) (50h)

        - Elaboración de scripts de aprovisionamiento en infraestructura cloud u On Premises (por ejemplo, Digital Ocean o Vmware VCenter) mediante Terraform o similar para las piezas de software que resulten escogidas (10h)

        - Elaboración de una herramienta que genere datasets de prueba con una periodicidad / volumen que pueda equipararse a los volúmenes reales de datos y que permita estudiar el desempeño del sistema. (10h)

- Realizar un ejemplo limitado de detección de anomalías sobre los conjuntos de datos obtenidos utilizando un framework compatible con el stack elegido en la fase de análisis (30h)

- Elaboración de la memoria y el video final (50h)
    - Redacción de la memoria
    - Grabación de screencasts
    - Edición del video final

1.5 Breve sumario de productos obtenidos

- Scripts de despliegue Ansible para varios de los Servicios descritos y scripts para despliegue de la infraestructura para dar soporte a las pruebas de concepto descritas que podrían servir como base para un despliegue para producción.
- Ficheros de configuración utilizados en las pruebas de concepto
- Generador de contenido aleatorio para generar trafico para los sistemes de ingesta de información
- Proyectos de ejemplo utilizando dos de las librerías más utilizadas para tratamiento de datos en streams (Apache Spark Streaming y Apache Kafka Streams)

1.6 Breve descripción de los otros capítulos de la memoria

En la sección 2, se analizan las distintas herramientas que permiten extraer la información de los nodos de la forma más adecuada, discutiéndose los requisitos y las diferentes tecnologías existentes para esto, así como tecnologías para permitir que estos eventos lleguen hasta la fase de carga/almacenamiento de una forma segura y escalable.

En la sección 3, se discuten las tecnologías que permiten realizar un indexado de la información, así como tecnologías que permiten visualizar la información ingerida.

La sección 4 discute algunas tecnologías que, sin ser parte del pipeline, suponen un apoyo.

La sección 5 discute los principales frameworks para Trabajo con big data y algunes de las herramientas que permiten realizar consultas a grandes almacenes de datos.

En la sección 6 se discute la arquitectura elegida, su implementación y algunas posibles mejoras.

2. Primera etapa del pipeline: Extracción y entrega

La idea principal de este proyecto consiste en crear un almacén de datos, por lo que las primeras fases del pipeline consistirán en la extracción, transformación y carga de los datos adquiridos a partir de los servidores monitorizados.

Como primer paso para el estudio de esta información, se extraerán los eventos de los logs de los nodos a monitorizar para su posterior transformación y carga. El requisito más importante en esta fase es el bajo impacto en rendimiento. También debe tenerse en cuenta la robustez, y el nivel de seguridad aportado por la solución escogida.

Idealmente, se debería mantener también una copia local en cada nodo durante algunos días, para que en caso de que el sistema pierda conectividad con el sistema de recogida no haya pérdida de información. Algunos de los requisitos básicos son:

- Posibilidad de conectar directamente con las siguientes etapas del sistema para buffering (Redis, Kafka) o para almacenamiento directamente (Elasticsearch, Hadoop-*)
- Bajo consumo de recursos
- Fácilmente instalable y configurable, idealmente mediante un gestor de configuración o script sencillo.
- Mantenido apropiadamente (proyecto activo)

Para extracción de la información desde los nodos, se han considerado las siguientes piezas de software:

- Filebeat (Elastic)
- Apache Fluentd
- RSyslog / syslog-ng

Para buffering se han considerado:

- Redis
- Kafka

Finalmente, como conectores entre la etapa de buffering/streaming y la de almacenamiento, se han considerado:

- Fluentd
- Flume
- Confluent Kafka connect
- Logstash

Filebeat:

Filebeat forma parte de la plataforma beats de Elastic inc. Es un recolector de logs totalmente integrado con el stack ELK, pero que además permite emplear buffering con herramientas como Redis o Kafka. Es muy flexible, y permite balancear la entrega de eventos desde el cliente entre varios servidores. Además, hace cierto buffering local, y se asegura de que cada evento se entregue al menos una vez. Su footprint (consumo de recursos) es bastante bajo, así que es un gran complemento a rsyslog en los nodos.

Ejemplo de entrega directa a kafka:

```
output.kafka:
  enabled: true
  # Servidores kafka
  hosts: ["prod-k1:9092, prod-k2:9092, prod-k3:9092"]
  # Topic / particion
  topic: es-zgz-not1'

  required_acks: 1
  compression: gzip
  max_message_bytes: 1000000
```

Además, filebeat cuenta con opciones de serie para tratar con entradas de varias lineas como las que genera un core dump de Java, una funcionalidad muy interesante si se utilizan aplicativos basados en esta tecnología.

```
### Multiline options

# Mutiline can be used for log messages spanning multiple lines. This is common
# for Java Stack Traces or C-Line Continuation

# The regexp Pattern that has to be matched. The example pattern matches all lines starting with [
#multiline.pattern: ^\[

# Defines if the pattern set under pattern should be negated or not. Default is false.
#multiline.negate: false

# Match can be set to "after" or "before". It is used to define if lines should be append to a pattern
# that was (not) matched before or after or as long as a pattern is not matched based on negate.
# Note: After is the equivalent to previous and before is the equivalent to to next in Logstash
#multiline.match: after
```

Fluentd

Fluentd es una alternativa a Logstash. Tiene también su propio log shipper similar a filebeat, una amplia cantidad de plugins de entrada y salida y ofrece soporte comercial.

Cuenta con buffering incorporado para dar solución a situaciones en las que los diferentes destinos no están disponibles.

Su rendimiento presenta algunas ventajas frente a logstash. Utiliza C-Ruby en lugar de JRuby, así que su rendimiento mejora bastante respecto a este.Resolución de nanosegundos. Permite forwarding de eventos mediantes ssl, y tiene incluso más plugins de entrada y salida que logstash.

- Outputs destacados: elasticsearch, kafka, mongodb, influxdb, redis, elastic beats,
- Algunos Inputs destacados: snmp, munin, mysql-replicator, docker y Kubernetes

Syslog-NG:

Syslog NG es una evolución del demonio syslog que incluyen la mayoría de sistemas Linux y UNIX por defecto. Cuenta con varios plugins y permite enviar los logs a un sistema syslog remoto.

Aunque Syslog NG permite conectar directamente con Kafka, por ejemplo, la capa que permite hacerlo requiere del uso de un conector escrito en Java, por lo que el consumo de recursos en cada máquina aumentaría considerablemente.

En el escenario planteado, podría configurarse para retener los datos que también se envían a Kafka durante un cierto tiempo, para evitar perder datos en caso de caídas / mantenimiento del cluster Kafka, siempre con una política de rotación razonable para evitar problemas de llenado de disco.

Tecnologías para buffering / streaming:

Kafka:

Kafka es una pieza de software que nos permite almacenar temporalmente información para poder consumirla de forma conveniente, tolerante a fallos, escalable, y eficiente.

A pesar de requerir un cluster Zookeper, incluye una versión de este lista para funcionar, y una instancia simple de Kafka, incluso tratándose de un entorno de pruebas modesto, permite procesar un número bastante elevado de eventos por segundo, con una latencia relativamente baja.

Apache Kafka: rendimiento:

Kafka incluye una herramienta que permite evaluar su rendimiento en el entorno en el que vayamos a emplearlo con relativa sencillez.

```
[carlos@localhost bin]$ ./kafka-producer-perf-test.sh --topic localhost --num-records 10000000 --throughput 500000 --record-size 150 --producer.config ../config/producer.properties


2499348 records sent, 499869,6 records/sec (71,51 MB/sec), 103,7 ms avg latency, 374,0 max latency.
2499735 records sent, 499947,0 records/sec (71,52 MB/sec), 0,9 ms avg latency, 11,0 max latency.
2499956 records sent, 499991,2 records/sec (71,52 MB/sec), 0,9 ms avg latency, 8,0 max latency.
2500457 records sent, 500091,4 records/sec (71,54 MB/sec), 0,9 ms avg latency, 6,0 max latency.
10000000 records sent, 499850,044987 records/sec (71,50 MB/sec), 26,57 ms avg latency, 374,00 ms max latency, 1 ms 50th, 241 ms 95th, 342 ms 99th, 371 ms 99.9th.
```

También incluye varias herramientas de lineas de comandos para gestionar la creación de topics, consumidores/emisores, etc.

El topic es un concepto utilizado en sistemas publicador / subscriptor. Las piezas de software que serán emisoras de información escriben datos en un topic, y los subscriptores podrán leer a partir de este. Dependiendo de la configuración / contexto del sistema, estos mensajes estarán disponibles más o menos tiempo, utilizarán diferentes formatos, etc.

La transmisión de información en Kafka sigue un modelo productor consumidor. En este caso, es responsabilidad del cliente hacer un seguimiento de la información que ya ha consumido. Será importante configurar correctamente la retención que queremos. Kafka nos da la opción de configurarla en base a consumo de espacio en disco o en base a tiempo.

Dada la cantidad de servidores de los que recibiremos datos y la importancia de mantener la información, elegido Kafka, serán necesario un mínimo de tres nodos para ejecutar Zookeper de una forma fiable, y otros tantos dedicados a nodos de Kafka, lo cual lamentablemente complica la arquitectura.

Apache Kafka: Topics:

La forma de funcionar de Kafka implica la creación de una serie de Topics en los que los colectores de cada servidor enviarán información a través de FileBeat.

A su vez, estos topics pueden particionarse para optimizar la escritura y lectura en ellos.

Podemos crear topics con el siguiente comando:

```
kafka-topics --zookeeper confluent-0:2181 --create --topic test-rep3 --partitions 24 --replication-factor 3
```

Para un topic no replicado, el troughput (cantidad de mensajes entregados por unidad de tiempo) parece aceptable para el sistema que pretendemos configurar. La latencia se estabiliza con el tiempo:

```
root@confluent-0:~# kafka-run-class
org.apache.kafka.tools.ProducerPerformance --topic test-one --num-records 1500000 --record-size 200 --throughput 50000 --producer-props acks=1 bootstrap.servers=confluent-0:9092 buffer.memory=37108864 batch.size=8196
204251 records sent, 40850.2 records/sec (7.79 MB/sec), 1111.1 ms avg latency, 1379.0 max latency.
296221 records sent, 59244.2 records/sec (11.30 MB/sec), 363.7 ms avg latency, 949.0 max latency.
250156 records sent, 50031.2 records/sec (9.54 MB/sec), 12.7 ms avg latency, 66.0 max latency.

(...)
```

Con un topic con replicación factor 3, tenemos algo más de latencia, pero es la única forma que tenemos de garantizar que nuestro sistema funcionará correctamente en caso de la caida de un nodo:

```
root@confluent-0:~# kafka-run-class org.apache.kafka.tools.ProducerPerformance --topic test-rep3 --num-records 1500000 --record-size 200 --throughput 50000 --producer-props acks=1 bootstrap.servers=confluent-0:9092 buffer.memory=37108864 batch.size=8196
238903 records sent, 47457.9 records/sec (9.05 MB/sec), 670.5 ms avg latency, 1203.0 max latency.
263113 records sent, 52601.6 records/sec (10.03 MB/sec), 34.5 ms avg latency, 635.0 max latency.
249882 records sent, 49976.4 records/sec (9.53 MB/sec), 15.1 ms avg latency, 138.0 max latency.
249718 records sent, 49943.6 records/sec (9.53 MB/sec), 14.6 ms avg latency, 139.0 max latency.
250756 records sent, 50151.2 records/sec (9.57 MB/sec), 13.5 ms avg latency, 122.0 max latency.
```

Resumidamente, dependiendo del número de nodos que se desee emplear en el clúster, se deberán crear topics con un número de particiones acorde. Idealmente se debería establecerá un mecanismo para automatizar cambios en este tipo de configuraciones (incrementar el número de particiones al añadir nuevos nodos, etc). En [6] se puede encontrar información detallada sobre el rendimiento de Kafka.

Redis:

La principal ventaja de Redis es ser un almacen en memoria y permitir clustering y sharding sin recurrir a agentes externos. En general está muy bien considerado en cuanto a rendimiento.

Redis incluye un benchmark que hemos ejecutado en la misma máquina que el resto de los tests para hacernos una idea de su capacidad de despachar mensajes a alta velocidad.

```
[root@localhost bin]# redis-benchmark -t set,lpush -n 100000 -q
SET: 121065.38 requests per second
LPUSH: 130378.09 requests per second
```

Redis cuenta con varias ventajas a la hora de su despliegue. En primer lugar, está empaquetado para las distribuciones mayoritarias, y cuenta con el soporte de varias empresas, aparte de llevar en el mercado varios años.

En el caso de la seguridad, la principal opción que tenemos respecto a asegurar Redis es seguridad perimetral, ya que Redis ha sido diseñado para ser rápido, no seguro. No tiene soporte para cifrado TLS / SSL, por ejemplo.

Escalabilidad: Redis soporta clustering y varios nodos como maestros / esclavos, así que su tolerancia a fallos y escalabilidad son bastante buenas.

Redis	Kafka
Mayor consumo de memoria	Mayor consumo de disco e IOPS
El broker se encarga de asegurar la entrega al menos una vez	El receptor es el encargado de verificar que ha recibido todos los mensajes disponibles.
Clustering out of the box	Requiere Apache Zookeeper
Pérdida de datos en caso de fallo -por defecto no se persiste necesariamente la información a disco-	Mayor fiabilidad en caso de fallos

Conectores entre el sistema de buffering / streaming y la etapa de almacenamiento:

Logstash

Logstash es una solución muy potente, gratuita, open source y que permite enrutar eventos entre diferentes entradas y salidas y transformarlos con diferentes filtros (posibilidad de añadir anonimización o enriquecer la información entrante, por ejemplo).

La forma de trabajo de logstash consiste en una o varias entradas de información (input plugins), que posteriormente pasan por una etapa de filtrado (filter plugins) y son conducidos a una o varias salidas (output plugins).

Algunos ejemplos de inputs son snmptrap –para recibir traps snmp desde dispositivos que lo soporten-, beats –para recibir métricas y logs vía las utilidades proporcionadas por elastic, como filebeat o metricbeat-, log4j….

Para el filtrado existen opciones que permiten cifrar información sensible, añadir campos a eventos json... también posibilita realizar transformación de formatos, por ejemplo desde formato collectd –recolector de métricas- a formato avro –formato binario utilizado por Hadoop-.

Las salidas son muy variadas, facilitando la integración de un número elevado de tecnologías como Redis,

Dado que utilizaremos un método de buffering previo, una solución a la falta de soporte para clustering de logstash podría pasar por repartir los topics entre distintas instancias de logstash y utilizar un orquestador para aumentarlas en número en caso de ser necesario. Un gestor de configuración como Chef permitiría especificar a qué instancia conectaría cada servidor.

Por ejemplo, crear una instancia de logstash que gestione grupos de 50 topics de kafka y los escriba en nuestro cluster HDFS. En caso de que la instancia falle, el orquestador se encargaría de levantar otra automáticamente para esos 50 topics.

```
input { kafka { zk_connect => "localhost:2181" topic_id => "event" } } output{
stdout{ codec => rubydebug } elasticsearch{ index => "event-%{+YYYY.MM.dd}"
hosts => ["localhost:9201"] codec => json } }
```

Ejemplo de un pipeline sencillo para logstash que consume datos desde Kafka y los envía a un topic de Elasticsearch.

Kafka Connect:

Kafka connect forma parte de la plataforma Confluent (una distribución de Kafka elaborada por confluent inc).

Esta pieza de software es escalable, puede configurarse mediante una interfaz REST y facilita la integración de varios componentes software.

Confluent Kafka Connect permite realizar transformaciones básicas sobre la información ingerida mediante SMT (Single Message Transformations), si bien la propia compañía sugiere no utilizarlos para transformaciones complejas, y en su lugar desplegar aplicaciones Kafka Streams que transformen la información ingerida y la reescriban a otro topic Kafka. Algunos ejemplos de SMT son:

- InsertField – Añade un campo estático o procedente de metadatos a cada registro
- TimestampRouter – Permite enrutar eventos a diferentes sinks o índices dependiendo del tiempo
- RegexpRouter – Permite enrutar eventos a diferentes topics en base a una expresión regular

Flume:

Flume es una herramienta de entrega de información con la idea de utilizar destinos basados en herramientas para big data.

- Principalmente optimizado para almacenar información en almacenes HDFS y HBase
- Tiene un source para Kafka, lo cual podría permitir usarlo como conector Kafka -> HDFS.

Se trata de un conector muy similar a Kafka connect, y tiene como punto a su favor no requerir de infraestructura externa para funcionar, como Zookeeper.

3. Indexado de la información ingerida

Una vez extraída la información de los nodos, es deseable tener la posibilidad de realizar un indexado de esta, al menos a corto/medio plazo. Esto permite analizar y visualizar la información, realizar consultas ad-hoc sencillas para detectar sucesos extraños / problemas que en principio sean detectables fácilmente de forma visual.

En nuestra arquitectura, determinar cómo indexar la información será clave a la hora de poder realizar una consulta eficiente de la información

Motores de indexado:

- Solr

Solr es uno de los motores de indexado más extendido. Está basado en Java y se integra muy bien con aplicaciones escritas en este lenguaje. Analizando varias comparativas y estudios, llegamos a la conclusión de que Solr es una solución robusta, bien documentada, con una gran comunidad, y además hay soluciones de visualización que permiten utilizarlo como backend.

Además, tiene un número diverso de formatos de entrada, como CSV, XML, Microsoft Word, PDF…. Este tipo de formatos dejan claro además que se trata de un motor de indexado orientado a búsquedas textuales, lo cual en el caso considerado no supone una ventaja.

En cuanto a escalabilidad, Solr permite configurarse en clúster utilizando ZooKeeper como almacén de datos distribuido.

- Elasticsearch

Elasticsearch es la herramienta de Elastic inc. Para indexar información y realizar búsquedas de un modo bastante avanzado. Utiliza Apache Lucene internamente.

Cuenta con un API rest que lo hace fácil de conectar con un amplio abanico de piezas software. Permite realizar búsquedas avanzadas en series de tiempo, agregación, etc.

En cuanto a seguridad, permite utilizar certificados y cifrado TLS/SSL para su manejo. Sin embargo, las opciones de autenticación y control de acceso se tratan de un añadido comercial.

Su escalabilidad es uno de sus puntos fuertes, ya que Elasticsearch fue diseñado para ser utilzado en entornos cloud. En [7] podemos encontrar información detallada sobre cómo escalar elasticsearch.

Cuenta con la posibilidad de hacer sharding y replicación de los índices, garantizando así que se puede trabajar sobre diferentes partes del índice de forma paralela, y que en caso de caída / mantenimiento de un nodo, el índice sigue siendo accesible.

Una de las funciones más útiles de Elasticsearch es la función denominada 'ingest pipelines'[13], que permite realizar transformaciones sobre los datos al ser ingeridos.

Por ejemplo, se puede crear un ingest pipeline para el tipo de eventos descrito:

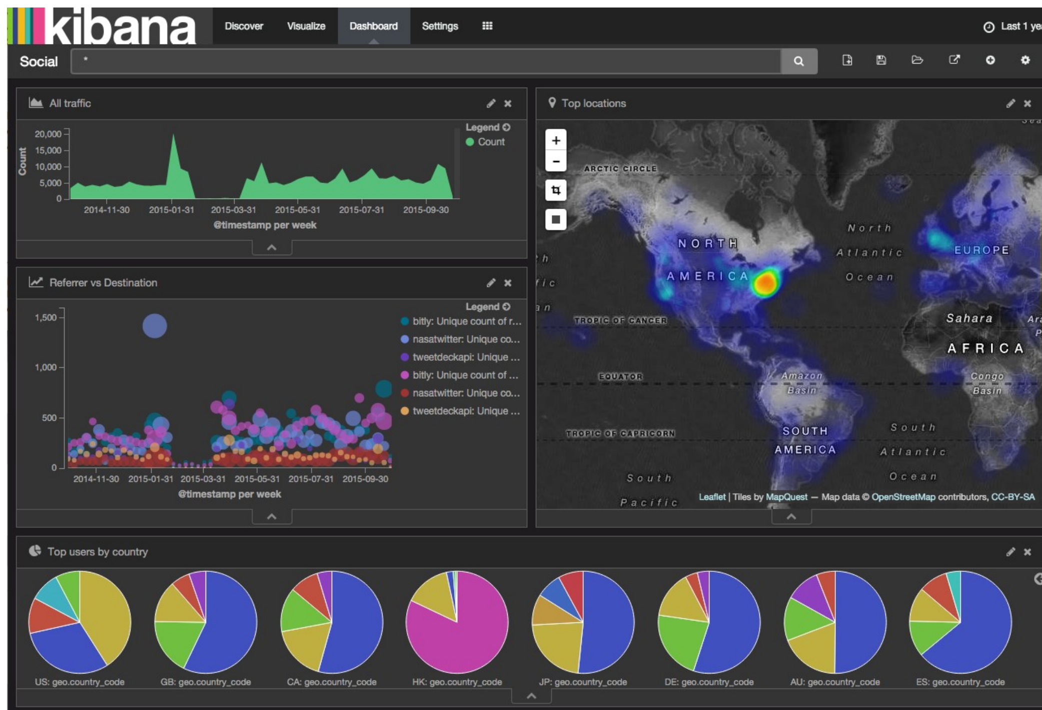
```
PUT _ingest/pipeline/dataevent
{
  "description" : "Convertir evento csv contenido en el campo message
de un documento JSON filebeat de la forma (Alerta, POP3, 110) a docume
nt json con campos separados ",
  "processors" : [
    {
      "grok": {
        "field": "message",
        "patterns": ["%{DATA:TipoDeEvento},%{DATA:Protocolo},%{DATA:Nu
mero}$"],
      }
    }
```

Podemos configurar filebeat para enrutar los mensajes al pipeline adecuado con la directiva de configuración del output elasticsearch ‘pipeline’.

Software de consulta y visualización:

- Kibana

Kibana es una muy buena opción para hacer consultas de grandes datasets por medio de elasticsearch. Permite generar tableros, y obtener una visión global de varias fuentes de datos sin mucho esfuerzo. Pero es una gran herramienta para visualizar logs.

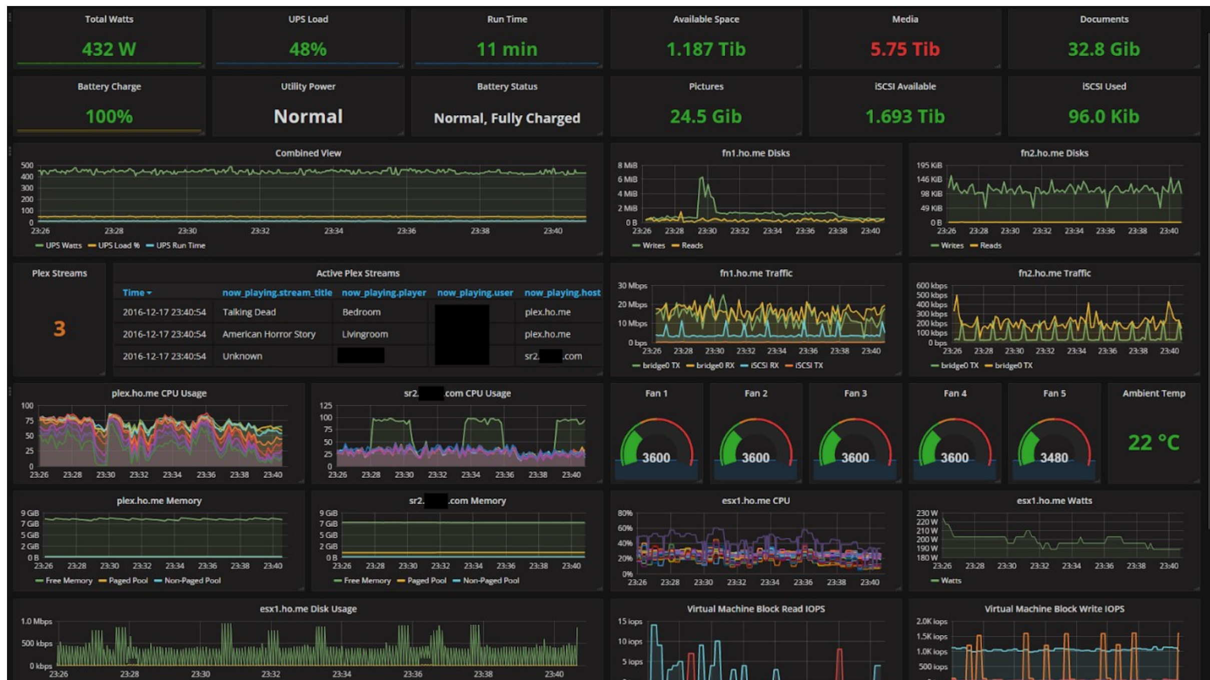


- **Banana**

Banana es un fork de Kibana diseñado para trabajar con Apache Solr.Es un proyecto muy interesante, ya que permite tener una alternativa completamente Software Libre. Su apariencia, al ser un fork es bastante similar a la de Kibana, si bien cuenta además con la posibilidad de utilizar visualizaciones basadas en la librería para visualización D3.js

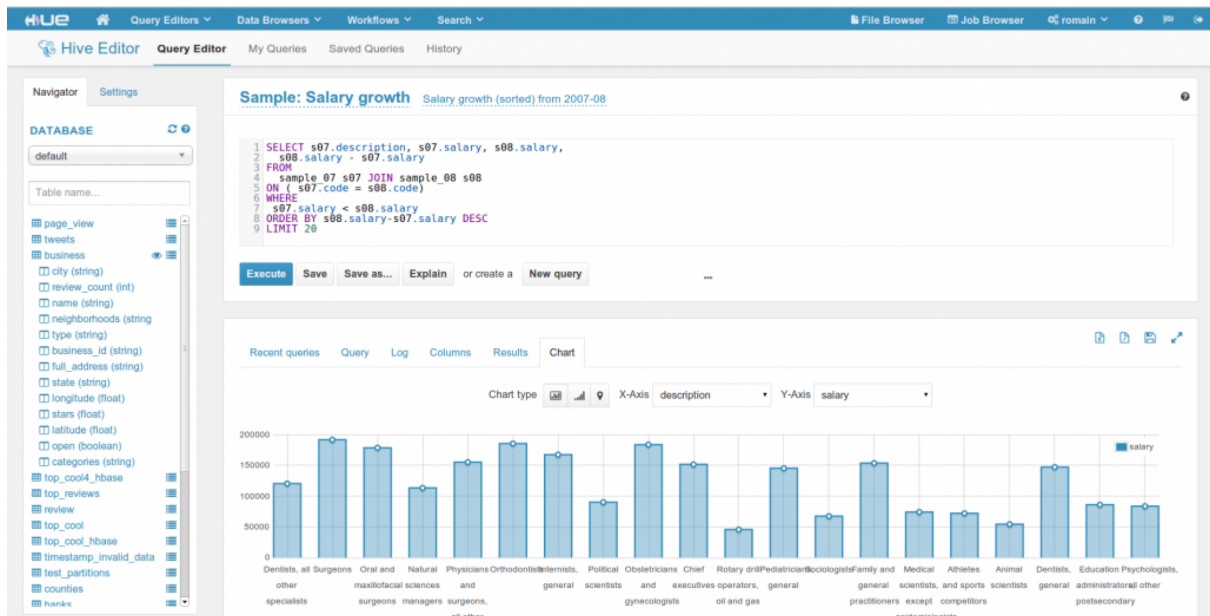
- **Grafana**

Grafana se trata de una herramienta muy potente para representar gráficamente métricas. Aunque puede parecer muy similar a Kibana, su cometido es más la representación gráfica, aunque en su última versión cuenta con funciones muy interesantes, como por ejemplo la integración con sistemas de mensajería para alertas, como por ejemplo, Telegram.



- **Hue**:

Hue no es un software diseñado para visualizar datos, sino un editor de consultas para Hadoop orientada a grandes almacenes de datos, si bien tiene un gran número de opciones dedicadas a poder visualizar datos de forma visual.



4. Generación de la información de prueba:

Si bien los contenedores son una tendencia en alza en el despliegue de arquitecturas software y en general de aplicativos y servicios, es cierto que hay un número alto de preocupaciones relativas a su seguridad.

Sin embargo, a la hora de realizar pruebas, se trata de una herramienta muy valiosa, ya que permite correr muchas instancias de una misma aplicación de forma muy cómoda y rápida.

Docker se ha utilizado en la elaboración de la prueba de concepto para poder generar fácilmente un número elevado de servidores que envíen información de registro al pipeline de datos.

Para realizar una simulación del comportamiento del sistema de forma integral, se ha utilizado la pequeña utilidad escrita en Java.

El contenedor se genera a través del siguiente Dockerfile:

```
FROM ubuntu:16.04
MAINTAINER Carlos de la Cruz

RUN apt-get update && apt-get install -y openjdk-8-jdk curl && curl -L -O
https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-6.2.4-amd64.deb && dpkg -i
filebeat-6.2.4-amd64.deb && apt-get clean && rm -rf /var/lib/apt/lists/*

COPY . /app
COPY filebeat/filebeat.yml /etc/filebeat/filebeat.yml
ENTRYPOINT ["/app/startup.sh"]
```

Si bien ejecutar dos servicios en un mismo contenedor contradice el patrón extendido de ‘un servicio por contenedor’, para propósitos de pruebas no se ha considerado que aporte valor generar varios contenedores, si bien a través de docker compose se podrían definir tres: uno para el generador de logs, otro para filebeat, y otro de almacenamiento que comparta el fichero de log entre ambos contenedores.

Para generar la imagen, se ejecuta el comando:

```
docker build -t thelogger .
```

Ahora, podemos ejecutar todas las instancias del generador de carga que queramos mediante el comando:

```
docker run --add-host kafka-01:206.189.3.210 --add-host kafka-02:206.189.11.15 -td
thelogger
```

5. Transformación de los datos ingeridos

No es común que la información esté lista para ser analizada, procesada y utilizada en el formato de origen. Siempre será necesario realizar operaciones intermedias, como eliminar campos que no sean relevantes, añadir meta información relativa al entorno en que fue adquirida (ubicación geográfica, cantidad de usuarios conectados en el momento en que se produjo un evento, etc).

Por ejemplo en el caso tomado como ejemplo, la información que llega se trata de información delimitada por comas. La API Kafka Streams permite realizar una transformación en tiempo real de la información para reescribirla en otro topic como campos separados, lo cual nos permitiría manejar la información desde Elasticsearch de una forma mucho más potente.

Además, esto nos permite realizar estas transformaciones de una forma escalable, ya que las aplicaciones Kafka streams permiten aumentar el número de nodos sin necesidad de un clúster/coordinador como en el caso de las aplicaciones Spark Streaming, lo que lo hace ideal para transformaciones de tipo sencillo que requieran alto rendimiento, dejando las aplicaciones Spark Streaming para operaciones más complejas, como ejecución de modelos / algoritmos de Machine Learning sobre los datos.

Las aplicaciones Kafka streaming pueden correr simplemente ejecutandose desde línea de comandos, si bien será necesario para su despliegue en producción la creación de units systemd[2], o utilizar un service wrapper. Hay algunas soluciones de Software Libre para esto, como por ejemplo 'Yet another Java Service Wrapper' (https://sourceforge.net/projects/yajsw/), o una versión comercial como Java Service Wrapper, de Tanuki Software.

En el supuesto que se trata en este proyecto, resultaría de gran utilidad para hacer transformaciones post-ingesta de la información:

- Limpieza de los datos
- Enmascaramiento de información sensible
- Transformación entre formatos de serialización

- Geolocalización aproximada de las direcciones IP de los logs.
- Conversión de los timestamps generados por metricbeat y filebeat para poder utilizarlos en un stream de salida con un serializador distinto.

Bootstrap de un proyecto Kafka Streams:

Comenzar un proyecto Apache Kafka Streams es de lo más sencillo. Solo es necesario contar con el IDE ‘IDEA’. el JDK de Java y Maven.

Desde línea de comandos debemos escribir:

mvn archetype:generate -DarchetypeGroupId=org.apache.kafka -DarchetypeArtifactId=streams-quickstart-java -DarchetypeVersion=1.1.0 -DgroupId=streams.examples -DartifactId=streams.examples -Dversion=0.1 -Dpackage=myapps

Esto genera un proyecto listo para compilar.

```
[carlos@pikonlabs-1 streams.examples]$ mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------< streams.examples:streams.examples >-----------------
[INFO] Building Kafka Streams Quickstart :: Java 0.1
[INFO] --------------------------------[ jar ]--------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ streams.examples ---
[INFO] Deleting /home/carlos/Streaming/streams.examples/target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ streams.examples ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ streams.examples ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 3 source files to /home/carlos/Streaming/streams.examples/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ streams.examples ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/carlos/Streaming/streams.examples/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ streams.examples ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ streams.examples ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ streams.examples ---
[INFO] Building jar: /home/carlos/Streaming/streams.examples/target/streams.examples-0.1.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.620 s
[INFO] Finished at: 2018-06-04T23:21:06+02:00
[INFO] ------------------------------------------------------------------------
[carlos@pikonlabs-1 streams.examples]$
```

Que se puede ejecutar de la siguiente forma:

```
carlos@pikonlabs-1 myapps]$ mvn exec:java -Dexec.mainClass=myapps.Pipe
```

Uno de los ejemplos que genera el arquetipo permite replicar un topic de entrada en uno de salida, y resulta de gran utilidad para comprender el funcionamiento del framework.

En el siguiente enlace puede verse un ejemplo de un esqueleto de código para un proyecto de detección de anomalías:

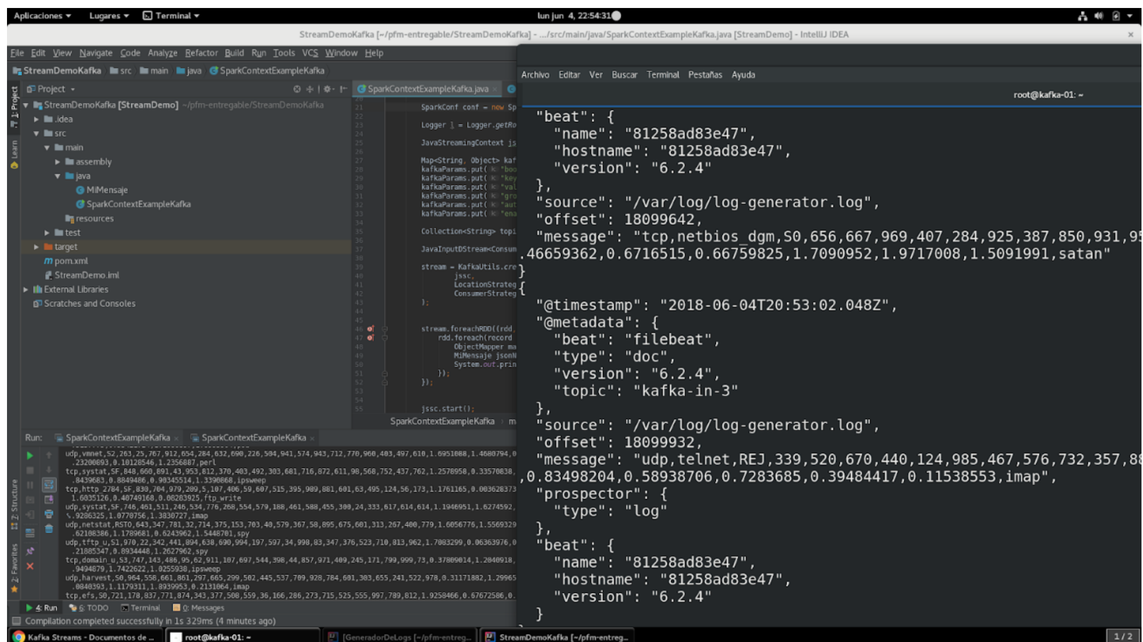
https://github.com/confluentinc/kafka-streams-examples/blob/4.1.0-post/src/main/java/io/confluent/examples/streams/AnomalyDetectionLambdaExample.java

Bootstrap de un proyecto Apache Spark:

Para iniciar un proyecto Spark basta con añadir las siguientes dependencias:

```
<dependencies>
<dependency>
   <groupId>org.apache.spark</groupId>
   <artifactId>spark-streaming_2.11</artifactId>
   <version>2.3.0</version>
</dependency>
   <dependency>
       <groupId> org.apache.spark</groupId>
       <artifactId>spark-streaming-kafka-0-10_2.11</artifactId>
       <version>2.3.0</version>
   </dependency>
</dependencies>
```

En la figura podemos ver un ejemplo de un proyecto sencillo que comienza un stream utilizando Spark streams. En el anexo 1 se muestra el ejemplo de código mostrado en la figura.



En este caso, se trata de una aplicación que realiza una transformación a partir de un topic de entrada kafka-in-3 y transforma cada mensaje a formato CSV a través de un serializador.

6. Procesamiento de los datos y frameworks para computación distribuida y manejo de Big Data

Zookeeper:

Apache Zookeeper es un componente necesario para varios componentes software como Kafka, Apache Drill.

Zookeeper se utiliza por piezas de software que operan en modo clúster, para almacenar configuraciones de forma distribuida, así como para la elección de maestros / promoción de nodos esclavos a maestros (Quórum), entre otras funciones.

Existen otras piezas software de propósito similar, como por ejemplo etcd, que es empleada en orquestadores como Kubernetes.

Se trata de un software sin demasiados requisitos de software y hardware.
Uno de sus puntos débiles era la seguridad, ya que por defecto no se usaba ningún método para autenticar los nodos,lo que facilitaba tipos de ataque. Si bien sigue siendo otro componente a asegurar especialmente mediante medidas adicionales (Firewall hardware, VLAN's dedicadas, etc), sí es cierto que ha introducido funcionalidades dedicadas a hacerlo más seguro, como autenticación entre nodos mediante SASL, Keytabs kerberos dedicados, etc.

Hadoop:

Hadoop es un framework para computación distribuida que proporciona varias facilidades a la hora de trabajar con big data.

Sus componentes más importantes son:

- HDFS (Sistema de archivos distribuido)

Hdfs crea un sistema de ficheros entre los nodos del clúster con un tamaño de bloque normalmente grande, ideal para procesos Batch. La forma de operar con el es mediante el comando DFS, al que deberemos proporcionar la dirección del clúster ZooKeeper y los comandos para interactuar con el sistema de archivos.

- YARN (Planificador de trabajos). Es el encargado de pasar los trabajos a MapReduce

- MapReduce es el ejecutor del código java de los trabajos.

Varias de las herramientas descritas posteriormente utilizan estos componentes de Hadoop, como motores de bases de datos y frameworks para trabajo con streams.

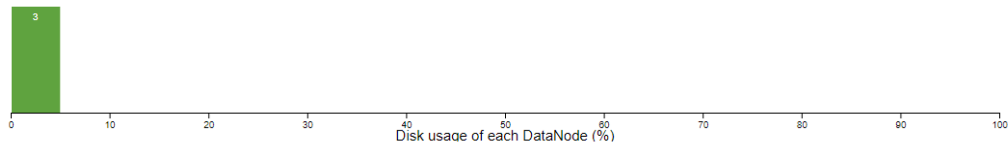
Más adelante, en el apartado 7 describimos su uso en el contexto de la definición de la arquitectura.

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Datanode Information

✔ In service ● Down ⊘ Decommissioned ⏻ Decommissioned & dead 🔧 In Maintenance & dead

Datanode usage histogram



In operation

Show 25 entries

Search:

Node	Http Address	Last contact	Last Block Report	Capacity	Blocks	Block pool used	Version
✔hadoop-0:9866 (167.99.215.158:9866)	http://hadoop-0:9864	1s	140m	24.06 GB	73501	970.82 MB (3.94%)	3.1.0
✔hadoop-1:9866 (206.189.3.98:9866)	http://hadoop-1:9864	0s	254m	24.06 GB	73501	970.82 MB (3.94%)	3.1.0
✔hadoop-2:9866 (174.138.11.120:9866)	http://hadoop-2:9864	1s	142m	24.06 GB	73501	970.82 MB (3.94%)	3.1.0

Hadoop incluye un sistema básico de gestión que permite conocer el estado de los nodos.

Apache Spark

Spark es un framework que mejora las características de Hadoop. Utilizando algunos de los componentes de este último, podemos realizar por ejemplo operaciones de enriquecimiento más complejas con los datos que hemos recolectado.

Spark cuenta con varios sub-frameworks orientados a diferentes operaciones de interacción con datos:

- Spark GraphX, para operar con grafos
- SparkSQL, que permite utilizar SQL de forma programática sobre diferentes fuentes de datos como streams.

- Spark streaming, para crear streams de datos a partir de sistemas de messaging u otras fuentes de datos. Este sub framework es interesante en nuestro caso, porque permitiría comparar por ejemplo valores que están sucediendo en tiempo real con otros almacenados a medio o largo plazo, y detectar diferencias de comportamiento.

- SparkML es una librería para machine learning. Permite utilizar el gestor de trabajos (YARN) de Hadoop, y utilizar datos almacenados en HDFS, lo cual lo hace interesante en el caso que estamos estudiando.

  SparkML incluye funciones como k-means [4] que permite categorizar grandes cantidades de 'puntos' (mediciones, entradas en log…) de forma paralela. Con ayuda de esta función, es relativamente sencillo separar los datos en conjuntos de forma que los que resulten con menor número de puntos podrían considerarse anómalos.

  Spark cuenta además con facilidades a la hora de trabajar con grandes conjuntos de datos, pues da soporte a tipos de datos como Parquet o Avro, etc.

  Se trata de un framework que requiere una configuración compleja para desplegar aplicaciones que lo utilicen, ya que requiere un cluster de procesado para funcionar, ya sea Hadoop YARN, Mesos, o su propio componente de software stand-alone para clustering. Sin embargo, esta flexibilidad de poder utilizar diferentes sistemas de cómputo también resulta una ventaja.

Motores de base de datos orientados a big data:

- Hive

  Hive es un software de 'Data Warehouse'…. No es una base de datos al uso, sino un software que permite realizar consultas sobre grandes conjuntos de datos en varios tipos de formato. Requiere un cluster Hadoop previamente configurado, ya que transforma un símil de SQL en trabajos MapReduce.
  Hive incorpora un entorno que permite realizar las consultas de modo interactivo.

El principal punto débil de Hive es que no está diseñado para consultas online, sino que las consultas realizadas consumen una gran cantidad de tiempo, ya que hay una transformación a código ejecutable en cada consulta.

- Drill

Apache Drill es una alternativa moderna a otros motores de base de datos para ser utilizados en entornos Hadoop.

A partir de datos sin un esquema bien definido almacenados en HDFS, Drill es capaz de comportarse como un motor SQL estándar.

Una función muy interesante en el supuesto estudiado es que permite realizar consultas sobre carpetas de datos almacenados en formato JSON.

Drill escala muy bien, desde ejecución embebida –sin utilizar ningún tipo de recurso externo- a ejecución distribuida, utilizando ZooKeeper y YARN.

Además cuenta con un interfaz web que permite gestionar las consultas en ejecución, ver detalles relativos a estas (explain), ver el estado del clúster…

localhost:8047/storage/dfs

Apache Drill Query Profiles Storage Metrics Threads Logs

Configuration

```
{
  "type": "file",
  "enabled": true,
  "connection": "hdfs://hadoop-0:9000",
  "config": null,
  "workspaces": {
    "root": {
      "location": "/topics/kafka-in-3/partition=0",
      "writable": true,
      "defaultInputFormat": null,
      "allowAccessOutsideWorkspace": false
    }
  },
  "formats": {
    "psv": {
      "type": "text",
      "extensions": [
        "tbl"
      ],
      "delimiter": "|"
    },
    "csv": {
      "type": "text",
      "extensions": [
        "csv"
```

Back Update Disable Delete

En este captura puede verse la configuración del plugin para HDFS. Esta configuración permite realizar consultas ad-hoc sobre una carpeta con un amplio número de ficheros JSON con los datos generados aleatoriamente. Pueden hacerse transformaciones sobre el esquema, agregación de datos, etc.

localhost:8047/profiles

Apache Drill Query Profiles Storage Metrics Threads Logs Options

No running queries.

Completed Queries

Loaded 100 profiles

Search Profiles

Display 10

Time	User	Query	State	Duration
06/03/2018 11:16:21	anonymous	SELECT count(*) FROM dfs.root.`kafka-in-3*.json`	Succeeded	15.878 sec
06/03/2018 10:58:32	anonymous	SELECT count(*) FROM dfs.root.`kafka-in-3*.json`	Succeeded	14.176 sec
06/03/2018 10:57:50	anonymous	SELECT count(*) FROM dfs.root.`kafka-in-3*.json`	Succeeded	15.794 sec
06/03/2018 10:57:38	anonymous	SELECT count(*) as num from FROM dfs.root.`kafka-in-3*.json`	Failed	0.019 sec
06/03/2018 10:57:31	anonymous	SELECT count(*) as num_eventos from FROM dfs.root.`kafka-in-3*.json`	Failed	0.020 sec
06/03/2018 10:55:27	anonymous	SELECT t.beat.hostname, count(t.beat.hostname) as mensajes FROM dfs.root.`kafka-in-3*.json` t group by t.beat.hostname	Succeeded	15.170 sec
06/03/2018 10:54:49	anonymous	SELECT t.beat.hostname, count(t.beat.hostname) as cuenta FROM dfs.root.`kafka-in-3*.json` t group by t.beat.hostname limit 100	Succeeded	16.630 sec
06/03/2018 10:54:36	anonymous	SELECT t.beat.hostname, count(t.beat.hostname) as cuenta FROM dfs.root.`kafka-in-3*.json` t limit 100	Failed	0.495 sec
06/03/2018 10:54:10	anonymous	SELECT `@timestamp`,t.beat.hostname FROM dfs.root.`kafka-in-3*.json` t limit 100	Succeeded	2.928 sec
06/03/2018 10:53:59	anonymous	SELECT `@timestamp`,t.beat.hostname FROM dfs.root.`kafka-in-3*.json` t limit 1	Succeeded	2.937 sec

First 1 2 3 4 5 ... 10 Last

Showing

El interfaz web nos permite ver el tiempo y el estado de la ejecución de las consultas. También incluye un profiler que permite ver de forma gráfica el plan de ejecución de las consultas SQL, de forma similar a otros motores de base de datos tradicionales como Microsoft SQL Server.

- Impala

Impala es un motor de SQL sobre hadoop al igual que Drill, pero está más fuetemente acoplado a Hadoop, y requiere además el almacén de meta información de Hive. Se creó con la idea de tener un motor de consultas sobre big data con relativamente baja latencia.

Otras distribuciones de Hadoop:

- Mapr
- Hortonworks - Versión open source disponible
- Cloudera - Solo versión enterprise

Estas distribuciones añaden valor a Hadoop para su uso en empresas / organizaciones, añadiendo estabilidad, soporte y, de alguna forma, adecuando su uso a un entorno productivo.

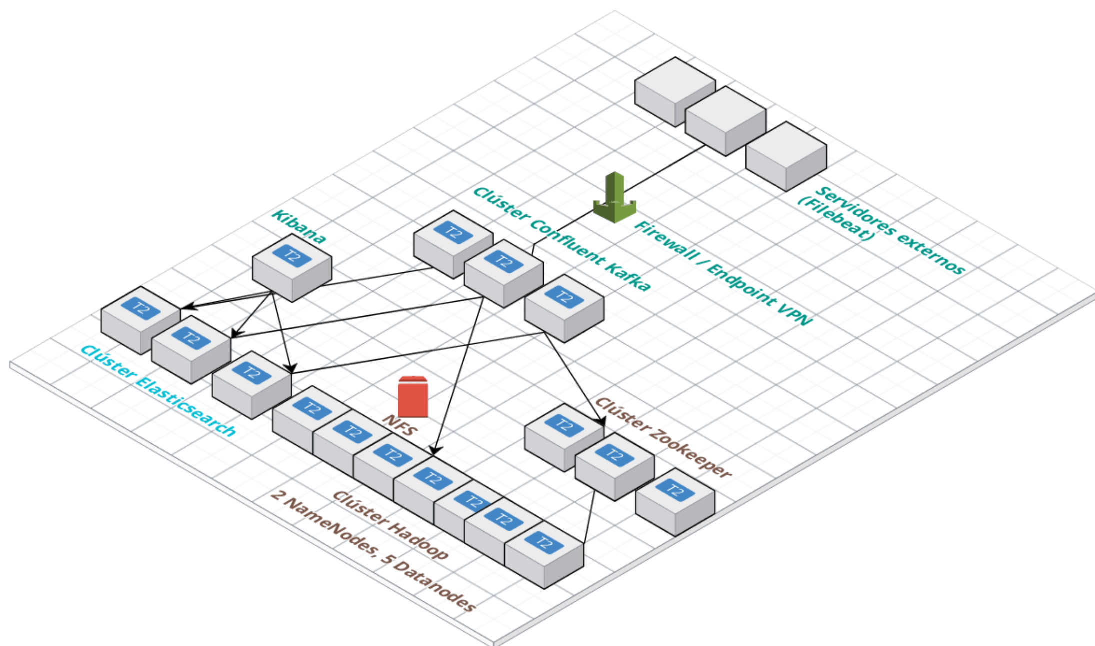
Cada una de estas tres distribuciones comerciales tiene varias características únicas. Por ejemplo MapR incluye grandes mejoras en cuanto a velocidad, empaqueta Pig y Hive con todas sus dependencias, y permtie el uso de un sistema de archivos propio.

En el caso de Hortonworks, es la única distribución de Hadoop optimizada para ejecutarse en hosts Windows y entornos Azure.

Cloudera por su parte, añade soporte para gestión multi-cluster.

Tras un estudio de las características de estos, se ha llegado a la conclusión de que no se hace necesario el uso de ninguna de estas para el caso que nos ocupa, y se optará por la distribución estándar de Hadoop 3.1

7. Elaboración de la arquitectura para extracción, transformación y carga de la información



Como se ha detallado anteriormente, la arquitectura contará con un componente de extracción (filebeat), streaming/buffering (Kafka + Kafka Connect + Apps Spark + Apps Kafka Streams), otro de análisis ad-hoc/corto plazo (Elasticsearch, Kibana, Apache Drill) y con otro relacionado con un almacenamiento a más largo plazo. (Almacén de datos Hadoop HDFS)

Estos sub-componentes comparten algunos elementos en nuestra arquitectura que requieren una configuración específica para interactuar entre ellos.

A continuación se describen los pasos seguidos para desplegar la arquitectura en un entorno de pruebas (conjunto de servidores virtuales en Digital Ocean, entorno cloud comercial).

Infraestructura utilizada en la prueba de concepto:

Para la prueba de concepto se han desplegado los siguientes servidores:

- Kafka01, kafka02
  - Clúster Confluent Kafka
  - Clúster Kafka-Connect
  - Apache Zookeeper
  - Apache Drill

- Hadoop0 (Máster Hadoop),Hadoop1, Hadoop2 (Data Nodes)
  - Clúster HDFS
  - Clúster YARN
- Elk-01
  - Elasticsearch
  - Kibana
  - Logstash (descartado)

- Docker01
  - Aplicación generadora de mensajes LOG alatorios/de pruebas
  - Filebeat

Aprovisionamiento de servidores:

Para el despliegue de cada uno de los componentes de la arquitectura en el entorno de pruebas se ha utilizado Ansible, y Terraform para aprovisionar la infraestructura virtual. En este caso hemos utilizado el proveedor Digital Ocean para Terraform, pero en el caso de estudio, se trata de una infraestructura on-premises que utiliza la plataforma VMWare, así que para este caso bastaría con crear un fichero terraform para dicho proveedor. Si bien la sintaxis que utiliza este proveedor es notablemente distinta, las funcionalidades que ofrece son bastante similares (reserva de recursos, elección de VLAN's, dispositivos de almacenamiento, etc).

Aprovisionamiento de la infraestructura virtual (infraestructura como código) con Terraform y Ansible:

Terraform permite crear infraestructura de forma automatizada a partir de definiciones de código. Esto es muy útil porque permite tener control de versiones en la infraestructura y también redimensionar las piezas ya existentes en cualquier momento de forma sencilla.

También queremos conseguir despliegues de software totalmente repetibles y que puedan ser ejecutados de forma idempotente, así que es una buena práctica utilizar gestores de configuración. En este caso, se ha utilizado Ansible, el gestor de configuración adquirido recientemente por Red Hat.

Lamentablemente Terraform no incluye un proveedor para este, pero sí tiene uno que permite ejecutar comandos remotos y locales, los cuales utilizaremos para ejecutar los playbooks Ansible. Los playbooks son ficheros de instrucciones que modelan configuraciones que queremos aplicar en una o varias máquinas.

Para poder interactuar con proveedores cloud, Terraform necesita conectarse a las API's que estos proveedores de servicios exponen, por lo que necesitaremos un token generado en la interfaz de gestión, en este caso, de Digital Ocean. [20]

Extracto del fichero Terraform utilizado:

```
resource "digitalocean_droplet" "terraform" {
	image = "ubuntu-16-04-x64"
	count = "3"
	name = "confluent-${count.index}"
	region = "ams3"
	size = "s-2vcpu-2gb"
	ssh_keys = [ "${var.digitalocean_ssh_fingerprint}"]
	provisioner "remote-exec" {
		inline = [ "apt-get update"]
	}
```

```
	provisioner "local-exec" {
		command = "sleep 120; ANSIBLE_HOST_KEY_CHECKING=False ansible-playbook -u ubuntu --private-key ./deployer.pem -i '${self.ipv4_address},' master.yml"
	}
}
```

Así, tras la ejecución del comando 'terraform apply' se generarán tres servidores virtuales, instalando lo básico para ejecutar playbooks ansible y añadir las claves ssh necesarias. [3]. En [12] puede encontrarse más inforamción acerca del proveedor Terraform para digital ocean.

master.yml es un playbook ansible general, que ejecutará en los servidores nuevamente generados los roles correspondientes (hadoop-master, nodo zookeper…). En el anexo 2 se reproduce el playbook para configuración e instalación de Hadoop.

Estos roles nos permitirán configurar servidores para uno o varios cometidos de una forma sencilla, de forma similar al sistema de roles ofrecidos por Microsoft Windows Server.

Configuración del clúster Hadoop

Para realizar pruebas con los conectores, se ha configurado un clúster Hadoop de tres nodos.

Dado que en principio la idea no será programar aplicaciones MapReduce, sino utilizar Hadoop como una herramienta más en nuestra plataforma de logging, solo utilizamos alguno de los servicios que ofrece.

El clúster de pruebas tendrá tres nodos, uno maestro y dos DataNodes.

Esta configuración tiene la desventaja de que el NameNode es un punto único de fallo a pesar de que añadamos más DataNodes y aumentemos la replicación en los ficheros. No obstante, de cara a una posible configuración de producción, es posible realizar una configuración en HA utilizando NameNodes secundarios en standby, y realizar una

conmutación en caso necesario. Esta configuración requiere de almacenamiento compartido en red (utilizando NFS).

HDFS proporciona mecanismos primitivos de fencing (aislamiento de nodos en fallo) para evitar problemas en los que los NameNodes no puedan determinar quién es el maestro.

Toda la información relativa a HA con Hadoop puede encontrarse en esta página de la documentación [1]

Enviado datos a HDFS para su consumo desde Hadoop / Spark

En el marco de este proyecto se ha decidido utilizar el framework Hadoop para una crear un data lake.

Dado que en este caso Hadoop es una pieza más en una estrategia para el análisis masivo de eventos de seguridad, se ha optado por no utilizar algunos de sus subsistemas, sino que solo los componentes que resulta de nuestro interés. En este caso son HDFS y Yarn. El uso que se pretende hacer de Yarn es simplemente utilizarlo para ejecutar trabajos de Spark o de Apache Drill. HDFS almacenará nuestros eventos a largo plazo.

HDFS es un sistema de archivos distribuido que nos facilitará procesar los amplios volúmenes de datos generados por la amplia cantidad de servidores que forman la red de notarías.

Para alimentar HDFS, se ha decidido utilizar la plataforma Kafka Connect de Confluent. Confluent Kafka es una distribución de Apache Kafka que además proporciona librerías de streaming y conectores apropiados para alimentar diferentes destinos (sinks) a partir de Apache Kafka.

Dado que suponemos un número elevado de servidores, lo apropiado será instalar varios nodos para la extracción de datos desde Kafka a HDFS, y configurarlos en cluster. Confluent permite transferir datos desde varias fuentes (Sources) y destinos (Sinks).

Lamentablemente no permite realizar demasiadas transformaciones en los datos procesados como otras herramientas como logstash, pero esto puede suplirse modificando los conectores existentes (es posible ya que todos son Open Source), o bien escribir una aplicación Spark para realizar estas transformaciones. Se ha optado por no utilizar logstash debido a que se desea una entrega a elasticsearch a muy corto plazo, y para el número de eventos que se tiene se requeriría un número alto de instalaciones de logstash para poder hacer un enriquecimiento y entrega en los tiempos deseados.

Instalación de la plataforma confluent:

Una configuración para producción de la plataforma confluent es relativamente sencilla ya que proporciona un tipo de configuración llamado ‘distribuido’. Así, utilizando este tipo de configuración, es sencillo escalar la plataforma para un elevado número de particiones y topics de Kafka.

Kafka connect crea una serie de topics especiales en Kafka para realizar un seguimiento del estado del consumo de los topics.

Es importante tener en cuenta una apropiada redundancia de estos topics con el setting ‘replication’ para cada uno de estos topics especiales.

Instalación del paquete de software y configuración del cluster ZooKeeper

En primer lugar, se ha hecho la instalación base del software. Basta con agregar el repositorio de confluent (Para el entorno de pruebas se utiliza Ubuntu 16.04, pero también existen paquetes disponibles para distribuciones basadas en Red Hat).

Previo a todo el proceso, se ha instalado Java en todos los nodos.

Utilizaremos la versión Open Source en lugar de la enterprise.

Una vez instalados los paquetes en todos los servidores que van a componer nuestro cluster confluent, debemos configurar ZooKeeper. Utilizamos la configuración de Zookeeper recomendada por confluent para configurar el cluster. En nuestro escenario de pruebas, mantendremos ZooKeeper en las mismas máquinas que ejecutarán Kafka

y el conector. En un entorno final de producción lo ideal será instalar el paquete de confluent en todos los nodos y activar ZooKeeper, Kafka o kafka connect según el nodo vaya a tener una u otra función.

Para el entorno de pruebas, se ha utilizado la configuración recomendada por confluent para ZooKeeper, si bien por motivos económicos no se ha desplegado un tercer servidor que permitiría tener tolerancia real a fallos.

```
tickTime=2000
dataDir=/var/lib/zookeeper/
clientPort=2181
initLimit=5
syncLimit=2
server.1=kafka-01:2888:3888
server.2=kafka-02:2888:3888
autopurge.snapRetainCount=3
autopurge.purgeInterval=24
```

Para que el clúster ZooKeeper pueda operar, debemos asignar un entero único a cada servidor en el fichero /var/lib/zookeeper/myid.

Una vez configurado zookeeper, se ha configurado una unit systemd para iniciarlo automáticamente. El siguiente fichero en los tres nodos (/etc/systemd/kafka-zookeeper) permite esto.

```
[Unit]
Description=Apache Zookeeper
After=network.target

[Service]
Type=simple
PIDFile=/var/run/zookeeper.pid
User=cp-kafka
Group=confluent
SyslogIdentifier=zookeeper
Restart=always
RestartSec=0s
ExecStart=/usr/bin/zookeeper-server-start /etc/kafka/zookeeper.properties
ExecStop=/usr/bin/zookeeper-server-stop
#ExecReload=/usr/bin/zookeeper-server restart

[Install]
WantedBy=multi-user.target
```

Esto permitirá iniciar el cluster ZooKeeper utilizando systemctl start kafka-zookeeper. También permitirá que el clúster se inicie al arranque de los nodos.

Configurando el clúster Kafka:

Kafka utilizará una configuración similar para el servicio systemd, cambiando 'zookeeper-server-start' y 'zookeeper-server-stop' por 'kafka-server-start' y 'kafka-server'stop' respectivamente, y utilizando el fichero /etc/kafka/server.properties.

Para configurar kafka correctamente en cluster, debemos modificar la configuración por defecto proporcionada por Confluent de la siguiente forma:

- broker.id = n, donde n es un entero distinto para cada servidor
- listeners=PLAINTEXT://confluent-1:9092, donde confluent-1 es el nombre dns público del servidor
- zookeeper.connect="128.199.55.193:2181,128.199.37.110:2181 "

En zookeeper.connect, indicamos las direcciones de todos los nodos del cluster zookeeper. En [8] se detalla el proceso completo de configuración de Kafka en modo distribuido. En el Anexo 3 puede verse el playbook utilizado para desplegar el clúster Kafka en el entorno de pruebas.

Confluent distributed connector:

Se deberá configurar un servicio systemd para la versión distribuida del conector Kafka -> HDFS, Elasticsearch.

La configuración del fichero de systemd es análoga a los dos servicios anteriores, especificando el path a la configuración correcta, que por defecto se encuentra en /etc/kafka/connect-distributed.properties

```
# A list of host/port pairs to use for establishing the initial connection to the Kafka cluster.
bootstrap.servers=10.133.55.54:9092,10.133.55.60:9092”
```

```
# unique name for the cluster, used in forming the Connect cluster group. Note that this must not conflict with consumer group IDs
group.id=connect-cluster
```

Podemos configurar sources y sinks en modo distribuido utilizando la API REST que proporciona el servidor.

Por ejemplo, de acuerdo a la documentación, podemos configurar un Sink de tipo fichero de la siguiente forma:

```
curl -X POST -H "Content-Type: application/json" --data '{"name": "local-file-sink", "config": {"connector.class":"FileStreamSinkConnector", "tasks.max":"1", "file":"test.sink.txt", "topics":"connect-test" }}' http://localhost:8083/connectors
```

En esta prueba de concepto se han configurados dos sinks, uno para Elasticsearch y otro para HDFS.

Configuración del conector HDFS:

La configuración que permitirá que nuestro cluster Hadoop reciba los eventos es la siguiente.

La URL especificada deberá ser uno de los master-nodes de un cluster HDFS que hemos configurado previamente.

Es importante configurar correctamente el tipo de mensajes que vamos a enviar, ya que si no lo hacemos, el conector no entregará mensajes. Por defecto, la configuración es para mensajes de tipo JSON. hay otros conversores de Llaves/Valores de utilidad en big data, por ejemplo existe un serializador para Apache Parquet. Como los eventos que estamos enviando son de tipo Syslog, utilizaremos el convertidor “StringConverter”

```
name=hdfs-sink
connector.class=io.confluent.connect.hdfs.HdfsSinkConnector
```

```
tasks.max=1
topics=uoc-logs
hdfs.url=hdfs://hadoop-0:9000
flush.size=100

key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter
key.converter.schemas.enable=false
value.converter.schemas.enable=false
```

Conector ElasticSearch:

El conector elasticsearch nos permitirá cargar los datos también en elasticsearch para realizar análisis ad-hoc de los datos ingeridos en nuestro pipeline, en combinación con Kibana. Kibana permite trabajar con varios índices siempre que el nombre siga un patrón, así que configuraremos el conector para crearlos para varios días.

```
name=elasticsearch-sink
connector.class=io.confluent.connect.elasticsearch.ElasticSearchSinkConnector
tasks.max=1
topics=uoc-logs
topic.index.map=uoc-logs:<logs-{now/d}> # Esta directiva permite separar los eventos por fecha en diferentes índices.
key.ignore=true
connection.url=http://192.168.99.50:9200
type.name=kafka-connect
bootstrap.servers=localhost:9092
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter
key.converter.schemas.enable=false
value.converter.schemas.enable=false
offset.flush.interval.ms=10000
group.id=connect-cluster
offset.storage.topic=connect-offsets
config.storage.topic=connect-configs
status.storage.topic=connect-status
schema.ignore=true
```

La opción topic.index.map permitirá mapear un topic a un nombre de índice, y generar estos índices de forma dinámica para distribuir los eventos por fecha / origen. También se pueden separar por host+fecha, etc.

Dado que podría consumirse gran cantidad de almacenamiento, se hace necesario utilizar una herramienta adicional que permita purgar los índices cada cierto tiempo. Esto es posible mediante la herramienta 'curator' proporcionada por Elastic. Bastará establecer un trabajo 'cron' que ejecute esta herramienta de línea de comandos.

7. Conclusiones:

La cantidad de tecnologías disponibles a la hora de trabajar con big data es enorme. Ha surgido toda una economía y un sector completo solo alrededor de este tipo de productos.

Es importante tener en cuenta que a pesar de la gran cantidad de tecnologías que surgen casi cada mes, siempre es necesario detenerse a evaluar y estudiar qué tecnologías son las más adecuadas antes de simplemente decidirse a utilizar la tecnología más extendida, o la más reciente.

Se trata de una materia de una gran complejidad, y cuya investigación requiere de una gran cantidad de recursos de cómputo, conocimientos en diversas ramas, y un amplio conocimiento del dominio del problema que se pretende resolver.

La correcta comprensión de cada una de las tecnologías que conforman el pipeline por separado requieren de varios meses de investigación y pruebas para validar la viabilidad y la escalabilidad futura de cada pieza que se añade al puzzle. La falta de una sola pequeña funcionalidad en una pieza de software que pudiera parecer la panacea puede hacerla totalmente inválida para funcionar en el contexto de una empresa.

La prueba de concepto presentada para la arquitectura de captura, almacenamiento y análisis de datos se ha elaborado pensando en un posible posterior despliegue en producción, si bien hay algunos puntos que sería importante terminar de resolver:

- Lamentablemente, Kafka connect no permite hacer uso de las 'ingest pipelines' de Elasticsearch. La funcionalidad podría ser introducida en el futuro, ya que existe código para esto en forma de pull requests aportados por la comunidad, así que en posteriores versiones, podría mejorarse la forma en que se ingieren los datos en elasticseach sin tener que recurrir a escribir una aplicación Kafka Streams para conseguir transformar la información ingerida en formato csv a campos independientes en Elasticsearch.

- Para alcanzar una mayor eficiencia y un mejor aprovechamiento de los datos , hay una pieza de software adicional que sería necesario añadir: Schema Registry, incluida en la distribución de software de Confluent.
  Esto permitiría utilizar el formato Avro o Parquet en toda la parte del pipeline que interactúa con Hadoop, lo cual permite un menor consumo de espacio en disco y unos tiempos de consulta / procesamiento notablemente menores.
  Se podría configurar el conector HDFS para transformar la información Json a formato Avro. Entonces también podría configurarse una aplicación Kafka Streams que lea del topic en el que se reciben los eventos desde Filebeat y escriba a otro en formato Avro, con lo que las aplicaciones Spark se beneficiarían de unos datos en un formato mucho más eficiente de leer y escribir, se consumiría menos espacio en el almacén HDFS y la velocidad de las consultas ad-hoc desde Apache Drill sería notablemente mayor.

- En la prueba de concepto se emplea Ansible para el despliegue de las distintas partes de la infraestructura. Para el caso de una gran flota de servidores, utilizar Chef podría suponer una ventaja frente al uso de Ansible, ya que permite una gestión más centralizada de todas las configuraciones. Por ejemplo, a la hora de configurar elasticsearch + kibana de forma securizada, un gestor de configuraciones como Chef puede facilitar mucho la distribución de claves y certificados. Además, sería muy recomendable monitorizar de forma centralizada que Filebeat se está ejecutando en cada uno de los nodos y configurar alertas relativas a esto.
- El stack de Elastic (Elasticsearch + Kibana) no permite autorización / autenticación en su versión gratuita. Si esto es una necesidad, podría considerarse utilizar la alternativa comercial, o bien sustituir estas piezas en la arquitectura por sus equivalentes gratuitos / libres: Solr y Banana, combinado

con Apache Manifold (framework que añade la posibilidad de autorización en aplicaciones Java EE).

- Para un uso de una instalación de Kafka de las dimensiones sugeridas, es necesario hacer una planificación minuciosa de posible crecimiento futuro de cara a establecer qué cantidad de topics, particiones y factores de replicación son necesarios, atendiendo también a la importancia de la información almacenada.

8. Glosario

Orquestador: Sistema utilizado para automatizar a alto nivel el despliegue de servicios.

Kubernetes: Orquestador de contenedores Docker creado por Google.

Gestor de configuración: Sistema utilizado para automatizar cambios en la configuración de un gran conjunto de hosts de forma centralizada.

Chef: Término que se refiere generalmente al software 'Chef automation', gestor de configuración creado por la empresa Chef Software.

Ansible: Gestor de configuración agentless (sin agente software específico que se ejecute en las máquinas gestionadas). Actualmente es propiedad de RedHat Software.

Data warehouse: Se trata de un sistema utilizado para reporting y análisis de datos. Permite consultar cantidades masivas de información obtenidas desde las operaciones de una organización o empresa y a menudo se utiliza para estudiar cómo mejorar estas.

Data lake: Se trata de un sistema que almacena grandes cantidades de información tal y cómo fue obtenida. Pretende ser un repositorio único para toda

la información tratada dentro de una organización, o para un sistema en concreto.

Big data: Término general que se refiere al empleo de cantidades masivas de datos, usualmente para realizar predicciones, elaborar estadísticas o detectar tendencias.

Stream: En el contexto de este proyecto, se refiere a un paradigma en programación o de acceso a al información que utiliza un sistema de paso de mensajes en lugar de acceder a la información conectando a una base de datos o leyendo directamente un fichero. Normalmente la información proviene de varias fuentes (streams) y se genera a una gran velocidad.

Topic: Es una categoría o nombre de feed que agrupa un conjunto de mensajes en un sistema publicador / subscriptor, o en el caso que nos ocupa, de streaming.

Shard: Generalmente hablando, unidad de particionamiento de la información en sistemas de tratamiento de información distribuida, por ejemplo en bases de datos o en motores de búsqueda.

Partición: En Kafka, es una secuencia ordenada de registros. Varias de estas secuencias conforman los topics en Kafka.

Avro y Parquet: Formatos binarios para serialización de datos utilizados en Hadoop.

Hadoop: Framework para computación distribuida.

Yarn: Planificador de trabajos para Hadoop

HDFS: Sistema de archivos distribuido de Hadoop

MapReduce: Motor de ejecución de trabajos de Hadoop

9. Bibliografía

- [1] HDFS High Availability - https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithNFS.html – Apache Foundation
- [2] Running Spring Boot apps with SystemD - Patrick Grimard https://patrickgrimard.io/2017/10/29/running-spring-boot-apps-with-systemd/
- [3]How to spin up a Hadoop Cluster with Digital Ocean Droplets - Jeremy Morris https://www.digitalocean.com/community/tutorials/how-to-spin-up-a-hadoop-cluster-with-digitalocean-droplets
- [4] Anomaly Detection: A Survery - Varun Chandola et al – University of Minesotta http://cucis.ece.northwestern.edu/projects/DMS/publications/AnomalyDetection.pdf
- [5] Hadoop MultiNode cluster - Tutorialspoint https://www.tutorialspoint.com/hadoop/hadoop_multi_node_cluster.htm
- [6] Monitoring Apache Kafka Performance Metrics – Datadog Inc. https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/
- [7] Elasticsearch Capacity Planning – Elastic co. https://www.elastic.co/guide/en/elasticsearch/guide/current/capacity-planning.html
- [8] Setting up Apache Kafka distributed cluster – Amit Kumar http://www.allprogrammingtutorials.com/tutorials/setting-up-distributed-apache-kafka-cluster.php
- [9] Spark Streaming and Kafka integration – Apache Foundation http://spark.apache.org/docs/latest/streaming-kafka-0-10-integration.html
- [10] KSQL – Confluent Inc https://docs.confluent.io/current/ksql/docs/tutorials/examples.html#ksql-examples
- [11] Introduction to Hadoop Distributed File System - Manpreet Singh and Arshad Ali http://www.informit.com/articles/article.aspx?p=2460260&seqNum=4
- [12] How To Use Terraform with DigitalOcean - Mitchell Anicas https://www.digitalocean.com/community/tutorials/how-to-use-terraform-with-digitalocean
- [13] A new way to ingest – Elastic co. - https://www.elastic.co/blog/new-way-to-ingest-part-1

10. Anexos

Anexo 1: Ejemplo de aplicación simple utilizando Kafka Streams. Leer un topic Kafka con un mensaje JSON, de-serializarlo y extraer los campos deseados. En este caso, payload del mensaje. Se trata de una modificación de un ejemplo básico de la documentación de Spark Streams

```
public class SparkContextExampleKafka  {

   public static void main(String[] args) throws InterruptedException {
       Logger.getLogger("org").setLevel(Level.ERROR);

       SparkConf conf = new
SparkConf().setMaster("local[2]").setAppName("NetworkWordCount");

       Logger l = Logger.getRootLogger();

       JavaStreamingContext jssc = new JavaStreamingContext(conf,
Durations.seconds(1));

       Map<String, Object> kafkaParams = new HashMap<>();
       kafkaParams.put("bootstrap.servers", "kafka-01:9092");
       kafkaParams.put("key.deserializer", StringDeserializer.class);
       kafkaParams.put("value.deserializer", StringDeserializer.class);
       kafkaParams.put("group.id", "test");
       kafkaParams.put("auto.offset.reset", "latest");
       kafkaParams.put("enable.auto.commit", false);

       Collection<String> topics = Arrays.asList("kafka-in-3");

       JavaInputDStream<ConsumerRecord<String, String>> stream;

       stream = KafkaUtils.createDirectStream(
               jssc,
               LocationStrategies.PreferConsistent(),
               ConsumerStrategies.<String, String>Subscribe(topics,
kafkaParams)
       );

       stream.foreachRDD((rdd, time) -> {
           rdd.foreach(record -> {
               ObjectMapper mapper = new ObjectMapper();
               MiMensaje jsonNode =
mapper.readValue(record.value().toString().replace("@",""), MiMensaje.class);
               System.out.println(jsonNode.message);
           });
       });

       jssc.start();
       jssc.awaitTermination();

   }
```

Anexo 2: Playbook para la configuración del clúster HDFS

```
- hosts: all
  become: true
  environment:
          JAVA_HOME: /usr/lib/jvm/java-8-openjdk-amd64
  tasks:
        - name: install java
          apt:
            name: openjdk-8-jdk-headless
            state: latest
        - name: create user
          user:
            name: hadoop
            home: /opt/hadoop
            shell: /bin/bash
            groups: sudo
            uid: 1040

        - name: create directory
          file:
            state: directory
            path: /opt/hadoop
            owner: hadoop
            mode: 0744
        - name: Download file with check (md5)
          get_url:
            url:
http://mirror.cc.columbia.edu/pub/software/apache/hadoop/common/hadoop-
3.1.0/hadoop-3.1.0.tar.gz
            dest: /tmp/hadoop.tar.bz2


        - name: Untar file
          unarchive:
            src: /tmp/hadoop.tar.bz2
            dest: /opt/hadoop
            creates: /opt/hadoop/hadoop-3.1.0/bin
            remote_src: yes
            owner: hadoop
            group: hadoop

        - name: Set environment options
          template:
            src: hadoop-env.sh.j2
            dest: "{{ item }}"
            owner: hadoop
            group: hadoop
            mode: 0755
          loop:
                  - /etc/profile.d/hadoop-env.sh
                  - /opt/hadoop/hadoop-3.1.0/etc/hadoop/hadoop-env.sh


        - name: Create data directory
          file:
            state: directory
            path: /opt/hadoop/data
            owner: hadoop
            group: hadoop
            mode: 0744

        - name: Configure basic hadoop files.
          template:
            owner: hadoop
            group: hadoop
            src: "{{ item }}.j2"
```

```
            dest: "/opt/hadoop/hadoop-3.1.0/etc/hadoop/{{ item }}"
          with_items:
                  - core-site.xml
                  - masters
                  - workers
                  - yarn-site.xml
                  - hdfs-site.xml
                  - mapred-site.xml

        - name: Create ssh directory
          file:
            state: directory
            path: /opt/hadoop/.ssh
            owner: hadoop
            group: hadoop
            mode: 0700

        - name: Copy keys
          template:
                  src: "{{ item }}.j2"
                  dest: "/opt/hadoop/.ssh/{{ item }}"
          with_items:
                  - id_rsa.pub
                  - id_rsa
                  - config
        - name: Create authorized keys
          template:
                  src: "id_rsa.pub.j2"
                  dest: "/opt/hadoop/.ssh/authorized_keys"

        - name: Config
          template:
                  src: "config.j2"
                  dest: "/opt/hadoop/.ssh/config"
        - name: Generate hosts file
          template:
                  src: "hosts.j2"
                  dest: "/etc/hosts"
        - name: Add IP address of all hosts to all hosts
          lineinfile:
            path: /etc/hosts
            line: "{{ hostvars[item]['ansible_eth0']['ipv4']['address'] }} {{
hostvars[item].inventory_hostname }}"
            state: present
          with_items: "{{ groups.hadoop }}"

        - name: Check for HDFS formatted
          stat: path=/var/lib/hdfs_formatted_dont_delete
          register: hdfs_formatted

        - name: Run HDFS format if not formatted
          hosts: hadoop-0
          sudo_user: hadoop
          command: /opt/hadoop/hadoop-3.1.0/bin/hdfs namenode -format
          when: hdfs_formatted.stat.exists == False
        - name: Create sentinel file for hdfs formatted
          copy:
                  content: ""
                  dest: "/var/lib/hdfs_formatted_dont_delete"
        - name: Set env for hadoop user
          template:
                  src: hadoop-profile.j2
                  dest: /opt/hadoop/.profile
                  owner: hadoop
                  group: hadoop
        - name: Setup systemd unit for hdfs
          hosts: hadoop-0
          template:
```

```
                src: hdfs.service.j2
                dest: /etc/systemd/system/hdfs.service
        - name: Setup systemd unit for hdfs (environment)
          hosts: hadoop-0
          template:
                src: hadoop.conf
                dest: /etc/systemd/system/hadoop.conf

        - name: Setup systemd unit for yarn
          hosts: hadoop-0
          template:
                src: yarn.service.j2
                dest: /etc/systemd/system/yarn.service
```

Anexo 3: Playbook para configurar el clúster Kafka

```
#!/usr/bin/ansible-playbook
- hosts: kafka
  become: true
  tasks:
        - name: Install confluent key
          apt_key:
            url: https://packages.confluent.io/deb/4.1/archive.key
            state: present

        - apt_repository:
            repo: deb [arch=amd64] https://packages.confluent.io/deb/4.1
stable main
            state: present
        - name: Install confluent connectors
          apt:
            name: "{{ item }}"
            state: present
          loop:
                - confluent-platform-oss-2.11
                - confluent-kafka-connect-elasticsearch
                - confluent-kafka-connect-hdfs

        - name: Configure elasticsearch connector
          template:
                src: elasticsearch-connector.properties.j2
                dest: /etc/kafka-connect-elasticsearch/kafka-connect-
elasticsearch.properties

        - name: Configure elasticsearch connector
          template:
                src: elasticsearch-connector.properties.j2
                dest: /etc/kafka-connect-hdfs/kafka-connect-hdfs.properties

        - name: Configure HDFS connector
          template:
                src: connector/hdfs-connector.properties.j2
                dest: /etc/kafka-connect-hdfs.properties

        - name: Configure Kafka
          template:
                src: "{{ inventory_hostname }}/server.properties.j2"
                dest: /etc/kafka/server.properties
        - name: Configure connect in distributed mode
          template:
                src: connector/connect-distributed.properties.j2
                dest: /etc/kafka/connect-distributed.properties
        - name: Configure zookeeper
```

```
          template:
                  src: connector/zookeeper.properties.j2
                  dest: /etc/kafka/zookeeper.properties
        - name: Configure zookeeper systemd unit
          template:
                  src: zookeeper.service.j2
                  dest: /etc/systemd/system/zookeeper.service
        - name: Configure kafka systemd unit
          template:
                  src: kafka.service.j2
                  dest: /etc/systemd/system/kafka.service
        - name: Configure connect systemd unit
          template:
                  src: connect.service.j2
                  dest: /etc/systemd/system/kafka-connect.service

        - name: Data dir
          file: path=/var/lib/zookeeper state=directory
        - name: Data dir
          file: path=/var/lib/kafka state=directory

        - name: Fix permissions for zookeeper
          file: dest=/var/lib/zookeeper owner=cp-kafka group=confluent
mode=u=rwX,g=rX,o=rX recurse=yes
        - name: Fix permissions for kafka
          file: dest=/var/lib/kafka owner=cp-kafka group=confluent
mode=u=rwX,g=rX,o=rX recurse=yes

        - name: Generate unique id
          template:
                  src: "{{ inventory_hostname }}/myid.j2"
                  dest: /var/lib/zookeeper/myid
```