Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Normalización, almacenamiento y correlación de eventos

Autor: Jaime Márquez Fernández
Tutor: Pablo Nebrera Herrera

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Normalización, almacenamiento y correlación de eventos

Autor:
Jaime Márquez Fernández

Tutor:
Pablo Nebrera Herrera
Profesor asociado

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo Fin de Grado: Normalización, almacenamiento y correlación de eventos

Autor: Jaime Márquez Fernández

Tutor: Pablo Nebrera Herrera

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

A mis padres

A mi ahijado, Mario

Agradecimientos

A mi tutor Pablo Nebrera Herrera por la confianza depositada en mí.

A Jaime Nebrera Herrera por guiarme en este ilusionante proyecto con tanto futuro.

A mis compañeros, en el período de prácticas, en ENEO Tecnología, en especial: Andrés Gómez, Carlos Rodríguez, que han aportado su granito de arena a este proyecto, ayudándome a mejorar, y Carlos Jiménez, por el seguimiento que tuvo de mi proyecto y ayudarme a sentir como uno más a mi llegada.

También mencionar a mis compañeros que, durante esta larga etapa, han contribuido a estar donde estoy hoy. En especial, Alberto y Manuel, con los que tantos momentos he vivido, trabajado, sufrido y reído; vivencias que nos han hecho amigos, de verdad.

A los educadores que han pasado por mi vida y se han preocupado de que sus alumnos aprendieran de verdad y tuvieran curiosidad por aquello que les rodea. En especial, a mi profesora Chari, la mujer que tanta culpa tiene de que acabará estudiando una Ingeniería; por sembrar gratuitamente en mí la curiosidad por estudiar, por inculcarme valores de respeto y ayudarme a pasar momentos difíciles en mi etapa de la adolescencia.

Y por último, lo más importante de mi vida, mi familia. Dar las gracias a mi hermano, Alejandro, y a mis padres, Ana y Pepe, por lo mucho que me han dado en esta vida; confiando siempre en mí y que tanto se han desvivido para que hiciera lo que más me gustará. Por ser parte principal en el motor de mi vida.¡Os quiero! y ¡Gracias, por tanto!

Jaime Márquez Fernández

Sevilla, 2016

Resumen

En la actualidad, en el ámbito tecnológico mas concretamente, hay una clara apuesta por extraer la máxima información de los datos que se generan.

En vista de lo anterior, nos decidimos a realizar una herramienta centrada en el tratamiento de registro de eventos, logs. Esta herramienta, por tanto, toma como datos de entrada logs y es capaz de generar información la cuál puede ser utilizada para, por ejemplo, generar informes de fallos, mejorar las prestaciones de una red, depurar arquitecturas complejas(Big data, desarrollo software…). En definitiva, el objetivo principal es ayudar a extraer la máxima información posible ante una serie de logs.

Dicha herramienta está desarrollada con sistemas de código abierto ya existentes, pero personalizadas para alcanzar los objetivos propuestos. Sin entrar en mucho detalle, se muestran dichos sistemas y su funcionalidad principal:

- Rsyslog: se encarga de la ingesta de logs y su normalización.
- Apache Kafka: sistema de almacenamiento distribuido.
- Siddhi CEP: genera información en base a la correlación de los datos, logs en nuestro caso.

La herramienta intenta dar solución a una serie problemas como:

- Abstracción del usuario de la complejidad del tratamiento de un log. Ya sea desarrollador, administrador de sistemas, auditor, usuario final, etc.
- Obtención de un tiempo de respuesta menor que por los métodos habituales: visualización de ficheros con ingentes cantidades de datos, uso de la depuración en ejecución, etc.
- Diagnosticar problemas y anomalías que afecten al buen funcionamiento de la máquina, red, sensor, etc.
- Detectar patrones de fallos e intrusiones.

Abstract

In this project, I show a tool that has as input data, logs. When input data has been processed , the tool generates information that can be used to produce informs, debugging complex architectures(Big data, software development), to generate alarms, etc.

I show as I developed the tool, from design to deployment. Inside of tool, some open source systems are integrated like Rsyslog, Apache Kafka and Siddhi CEP.

In conclusion, this tool helps you to manage big amount of events and producing useful information.

Índice

Índice de Tablas

Índice de Figuras

Notación

>	Indica el uso de una terminal
. .	Directorio padre
.	Directorio actual
mvn	Comando asociado a maven
java	Comando que ejecuta clases java
service	Comando para operar sobre servicios
curl	Comando para enviar peticiones a un servidor
git	Comando para ejecutar el controlador de versiones
.java	Extensión de ficheros java
.jar	Extensión de ejecutables java empaquetados
.xml	Extensión de ficheros XML
.yml	Extensión de ficheros YAML
.conf	Extensión de ficheros de configuración
.rules	Extensión de los ficheros que contienen las reglas de normalización

1 INTRODUCCIÓN

"Getting information off the Internet is like taking a drink from a fire hydrant."

- Mitchell Kapor-

Un punto de generación de datos son los logs, registros de eventos producidos por: cualquier tipo de programa, sistema operativo, sensor, electrónica de red, etc. Se puede decir que los logs, de primeras, asustan por la cantidad de datos que lo conforman y la poca información que aportan a simple vista. Sus usos más comunes son: seguir la ejecución de un programa, detección de anomalías, fallos de configuración…

El formato de un log está estandarizado(RFC 5424**[1]** y RFC 3164**[2]**) así que, tras cierto periodo de aprendizaje, no es tan complicado entender y extraer información útil de éstos. Eso está bien si no tenemos una red de grandes dimensiones o pocos servicios que monitorizar, pero cuándo el tamaño de los servicios, red o equipos a monitorizar crecen, la inspección visual y utilización de herramientas básicas acaban por ser tediosas y difíciles de manejar. Por eso en estas líneas se presenta una solución que facilite la tarea del tratamiento de un log y obtenga el mayor rédito posible de éstos.

1.1 Alcance

Se pretende realizar un desarrollo, consistente en una herramienta que sea capaz de importar logs en diferentes formatos y los normalice a un formato común(JSON). Posteriormente, se debe incluir una inteligencia capaz de realizar cálculos complejos sobre los datos, proporcionando la mayor información posible(informes, alertas, estadísticas…). Todo ello, tanto datos como resultados, debe almacenarse en un sistema que garantice alta disponibilidad.

Los logs una vez normalizados, independientemente del servicio, si presentan campos comunes se deberán nombrar de la misma forma.

Se sugiere que los sistemas utilizados sean:

- Como colector y normalizador de logs **Rsyslog**
- El sistema de almacenamiento distribuido escogido debe ser **Apache Kafka**
- Y como motor de correlación **Siddhi CEP**

Además, deberá disponer de funciones que permitan al usuario hacer uso del motor de correlación. Se deben considerar, al menos, las siguientes funcionalidades básicas:

- Iniciar un plan de ejecución
- Reiniciar el plan de ejecución existente
- Listar las *queries* asociadas al plan de ejecución existente
- Modificar un plan de ejecución existente
    - Añadir *query*
    - Eliminar *query*

Esta extensión puede ser incorporada mediante una API REST que interactúe con el motor de correlación.

En la siguiente imagen(Figura 1-1) se muestra el comportamiento que debe tener la aplicación:

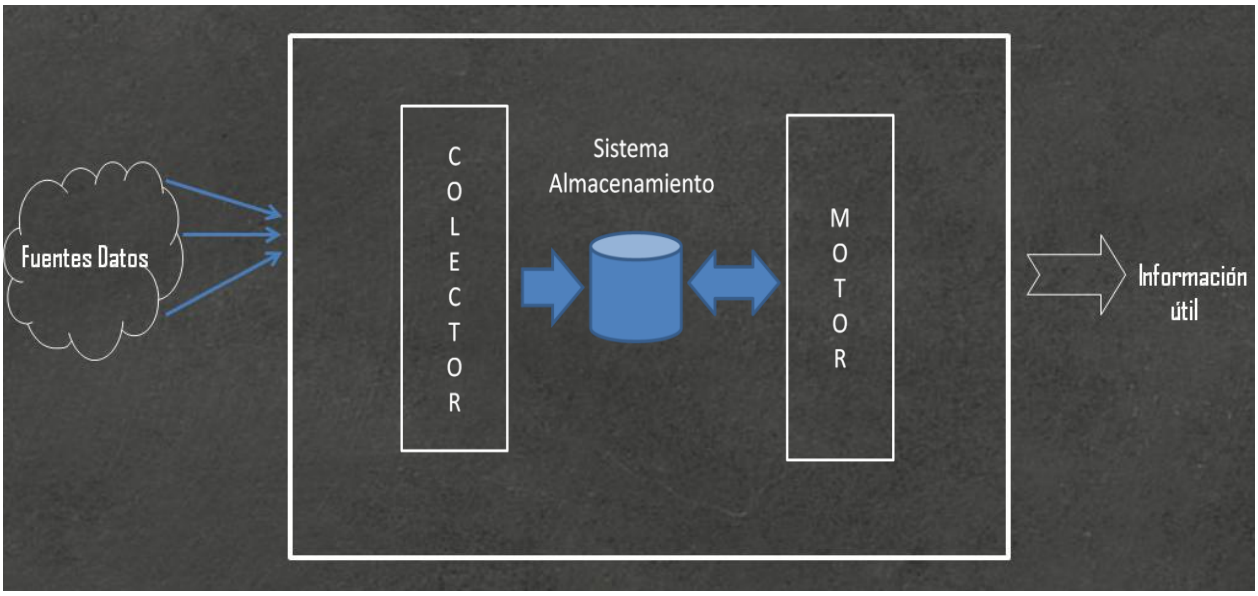

Figura 1-1. Esquema general de la herramienta

1.2 Objetivos

Una vez concluida la herramienta, ésta debe ser capaz de:

- Recolectar todo tipo de logs, independientemente del formato. Aunque en este proyecto nos centraremos en dos tipos de formato de entrada: JSON y RFC5424-3164.
- Normalizar a un formato común, JSON.
- Almacenamiento distribuido, persistente y con alta disponibilidad. Tanto de los logs en "crudo", como de los resultados producidos por el motor de correlación.
- Realizar cálculos complejos sobre los logs, mediante el motor de correlación.
- Ofrecer un método de interacción con el usuario, que permita a éste la personalización/modificación de las funciones del motor.

1.3 Conceptos teóricos implicados

1.1.1. Log

Ya que el término es anglosajón, partiremos de su definición en inglés e intentaremos darle un significado particularizado al mundo tecnológico:

Definición log[1]: *"a full written record of a journey, a period of time, or an event"*

De la que se puede deducir que, aplicando el ámbito telemático, un log es un registro de un evento. Básicamente es una huella que deja un servicio en un tiempo determinado.

Una vez definido el término, entramos a explicar el protocolo Syslog, que nos ayudará a la hora de tratar un log.

1.1.2. Syslog Protocol[2]

Este protocolo nació para dar respuesta a la problemática del formato y método de transporte, dado que cada fabricante ponía su formato y utilizaba el protocolo de transporte que, para ellos, eran los más adecuados. Esto provocó que cada log fuera totalmente diferente al siguiente, incluso dentro de un mismo servicio. Derivado de esto surgió la complejidad del tratamiento de un log, convirtiendo su recolección en tediosa al necesitar una configuración diferente para cada tipo de log recibido, viéndose dificultada así la extracción de información.

Dada la importancia que se le empezó a dar a los logs y a su tratamiento, se pensó en esta estandarización como solución que satisficiera a todas las partes.

Este protocolo ofrece una capa de abstracción que permite utilizar cualquier protocolo de transporte sin afectar al log y su formato. Ofrece un formato común para los parámetros generales presentes en todos los servicios, sin llegar a perder la personalización que cada servicio necesite.

Su aplicación es bastante sencilla, solo hay que seguir las indicaciones respecto al formato y tipos de datos permitidos, permitiéndonos usar el protocolo de transporte que más se adecue a nuestras necesidades. Separando así las capas de aplicación y transporte de forma clara y concisa.

```
UTF-8-STRING    = *OCTET ; UTF-8 string as specified
                  ; in RFC 3629

OCTET           = %d00-255
SP              = %d32
PRINTUSASCII    = %d33-126
NONZERO-DIGIT   = %d49-57
DIGIT           = %d48 / NONZERO-DIGIT
NILVALUE        = "-"
```

Tabla 1–1. Tipos de datos permitidos

[1] Definición extraída del diccionario online de Cambridge: http://dictionary.cambridge.org/es/
[2] Definido en las RFCs 5424 y 3164, esta última obsoleta por la aplicación de la primera.

```
SYSLOG-MSG      = HEADER SP STRUCTURED-DATA [SP MSG]

HEADER          = PRI VERSION SP TIMESTAMP SP HOSTNAME
                  SP APP-NAME SP PROCID SP MSGID
PRI             = "<" PRIVAL ">"
PRIVAL          = 1*3DIGIT ; range 0 .. 191
VERSION         = NONZERO-DIGIT 0*2DIGIT
HOSTNAME        = NILVALUE / 1*255PRINTUSASCII

APP-NAME        = NILVALUE / 1*48PRINTUSASCII
PROCID          = NILVALUE / 1*128PRINTUSASCII
MSGID           = NILVALUE / 1*32PRINTUSASCII

TIMESTAMP       = NILVALUE / FULL-DATE "T" FULL-TIME
FULL-DATE       = DATE-FULLYEAR "-" DATE-MONTH "-" DATE-MDAY
DATE-FULLYEAR   = 4DIGIT
DATE-MONTH      = 2DIGIT  ; 01-12
DATE-MDAY       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31 based on
                          ; month/year
FULL-TIME       = PARTIAL-TIME TIME-OFFSET
PARTIAL-TIME    = TIME-HOUR ":" TIME-MINUTE ":" TIME-SECOND
                  [TIME-SECFRAC]
TIME-HOUR       = 2DIGIT  ; 00-23
TIME-MINUTE     = 2DIGIT  ; 00-59
TIME-SECOND     = 2DIGIT  ; 00-59
TIME-SECFRAC    = "." 1*6DIGIT
TIME-OFFSET     = "Z" / TIME-NUMOFFSET
TIME-NUMOFFSET  = ("+" / "-") TIME-HOUR ":" TIME-MINUTE


STRUCTURED-DATA = NILVALUE / 1*SD-ELEMENT
SD-ELEMENT      = "[" SD-ID *(SP SD-PARAM) "]"
SD-PARAM        = PARAM-NAME "=" %d34 PARAM-VALUE %d34
SD-ID           = SD-NAME
PARAM-NAME      = SD-NAME
PARAM-VALUE     = UTF-8-STRING ; characters '"', '\' and
                               ; ']' MUST be escaped.
SD-NAME         = 1*32PRINTUSASCII
                  ; except '=', SP, ']', %d34 (")

MSG             = MSG-ANY / MSG-UTF8
MSG-ANY         = *OCTET ; not starting with BOM
MSG-UTF8        = BOM UTF-8-STRING
BOM             = %xEF.BB.BF
```

Tabla 1–2. Definición formato log

Así, si seguimos dicha especificación, cada log estará compuesto por una serie de campos generales(la fecha, la hora, el nombre del servicio…) y otros que incluyen la información específica de cada servicio(MSG, STRUCTURED DATA). Además, nos define el tipo de dato que puede ser utilizado en cada campo, estandarizando tanto el formato como el tipo de dato a utilizar.

Definamos, con más detalle, algunos campos que nos harán falta a la hora de tratar un log:

- **PRI**: indica la fuente del log, véase Tabla 1–3, y el nivel, véase Tabla 1–4, con el que fue generado. Estos campos nos ayudarán a clasificar el log según su criticidad y la fuente que lo genera.
- **HOSTNAME**: indica la ip o el nombre de la máquina donde se generó el log.
- **APP_NAME**: el nombre de la aplicación ó servicio.
- **MSG**: contiene la información específica de cada log, en lenguaje natural, que ayuda a entender la causa de éste.

Numerical Code	Facility
0	kernel messages
1	user-level messages
2	mail system
3	system daemons
4	security/authorization messages
5	messages generated internally by syslogd
6	line printer subsystem
7	network news subsystem
8	UUCP subsystem
9	clock daemon
10	security/authorization messages
11	FTP daemon
12	NTP subsystem
13	log audit
14	log alert
15	clock daemon (note 2)
16	local use 0 (local0)
17	local use 1 (local1)
18	local use 2 (local2)
19	local use 3 (local3)
20	local use 4 (local4)
21	local use 5 (local5)
22	local use 6 (local6)
23	local use 7 (local7)

Tabla 1–3. Numeración de las diferentes fuentes

Numerical Code	Severity
0	Emergency: system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

Tabla 1–4. Niveles de criticidad

1.1.3. Rest

Rest es una tecnología basada en recursos, ampliamente utilizada en el mundo del desarrollo web, que nos permitirá conectar al usuario con el motor para obtener datos o ejecutar acciones en éste. Aunque Rest presenta una aplicación más estricta y correcta centrada en el ámbito de la arquitectura software, su definición ha ido degenerando a la explicada, anteriormente, por el uso de las APIs Rest en el desarrollo web. Por tanto, nos vale dicha definición, ya que, queremos presentar una interfaz entre el usuario y el motor.

Esta tecnología se apoya, principalmente, en el protocolo HTTP. A continuación explicaremos los elementos de este protocolo que nos harán falta, sin entrar en profundidad en la definición completa de dicho protocolo.

1.3.1.1 HTTP Protocol[3]

Este protocolo, como se ha comentado anteriormente, abarca más allá de lo que nosotros necesitamos. Lo que en realidad nos atañe son los métodos con los que obtener y modificar un recurso y la respuesta que debemos devolver según los diferentes casos que se presenten.

Los métodos definidos en este protocolo, que utilizaremos, son:

- **POST**: utilizado para añadir un nuevo recurso o modificar uno existente.
- **GET**: utilizado para obtener los recursos existentes.
- **DELETE**: utilizado para eliminar un recurso existente.

Los códigos que debe proporcionar, según el estado del recurso y la petición son:

Código	Explicación
200	Petición aceptada y ejecutada correctamente
202	Petición aceptada, pero no se completo la ejecución
404	Recurso no encontrado
500	Servidor fuera de servicio

Tabla 1–5. Códigos HTTP de respuesta

Por último, debemos tener en cuenta que la URL utilizada no puede contener verbos o acciones. Esto último, no se incluye en el protocolo HTTP, sino que es una guía de buenas prácticas de la implementación de Rest. A continuación, un ejemplo de cómo incluir un verbo o acción de forma apropiada en una URL.

Ejemplo 1–1. *url/action=action_to_make/[resource_id]*

[3] Definido en la RFC 2068[3]

2 MODELADO DE LA HERRAMIENTA

Programming without an overall architecture or design in mind is like exploring a cave with only a flashlight: You don´t know where you´ve been, you don´t where you´re going and you don´t know quite where you are.

-Danny Thorpe-

En esta sección, se introducirá al lector en el funcionamiento y funcionalidades de las diferentes herramientas empleadas. Para luego completar con un diseño, incluyendo dichas funcionalidades, que nos facilite el desarrollo e integración de la herramienta.

2.1 Especificaciones

Antes de pensar en cada parte del diseño, debemos pensar en las entradas y salidas del sistema. Como entrada puede haber diferentes servicios y formatos, en este proyecto nos centraremos en los servicios:

- SSH que generará logs en formato RFC5424
- Apache2, servidor web, que generará logs en formato JSON[4].

Como salida se ha escogido el formato JSON. Por tanto, los logs deben pasar un proceso de normalización para que ambos servicios presenten el mismo formato y misma nomenclatura en campos generales.

Por último, en lo referente a codificación, será utilizado el lenguaje de programación Java, ya que, Siddhi CEP está programado en ese mismo lenguaje, Apache Kafka presenta una API para dicho lenguaje. Debiendo aplicar, por tanto, un diseño basado en orientación a objetos.

Una vez hecha estas aclaraciones, pequeñas pero importantes, procedamos al diseño de cada herramienta por separado y a su posterior integración, comprobando que todo se desarrolla con normalidad.

2.2 Colector Rsyslog

Rsyslog[5][4] es un sistema de *logging* capaz de tratar logs de diferentes sistemas(Unix, Windows, Solaris…) y ofrece una variada cantidad de formatos de salida(ElasticSearch, HDFS, mySQL…) . Está desarrollado en el lenguaje de programación C y ofrece un diseño modular que permite tener compilado y ejecutándose sólo los módulos que se precisen, con el consiguiente ahorro en memoria y mejora en tiempo de computación.

En esta sección, se introducirá al lector en los conceptos claves que, posteriormente, serán utilizados para desarrollar un colector con las características apropiadas. Recordemos que la función de este colector es obtener los logs de diferentes fuentes y normalizarlos a un formato común, JSON.

[4] Para ello, se debe modificar la configuración de dicho servicio, véase Anexo A: Configuración Apache2

[5] Rsyslog es un proyecto de código abierto desarrollado en el seno de Adiscon y liderado por Rainer Gerhards

2.2.1 Funcionamiento Rsyslog

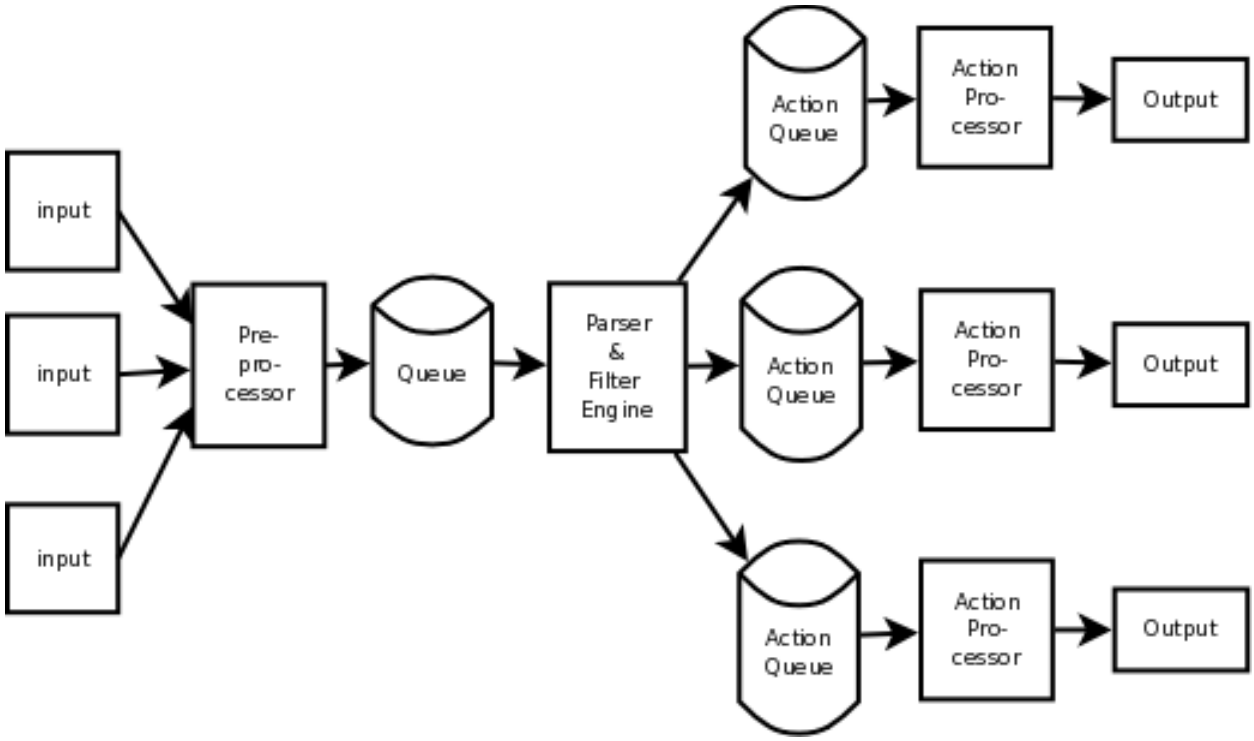

Figura 2-1. Arquitectura software Rsyslog

Definamos cada componente, esto nos ayudará para la posterior configuración del sistema.

2.2.1.1 Preprocessor

Se encarga de recibir el log en primera instancia y hacer un primer procesamiento. Es decir, obtiene el formato de entrada del log, los diferentes campos y valores que lo conforman. En esta fase se obtienen todos los campos definidos en el protocolo Syslog(si se ha hecho uso de esta especificación).

2.2.1.2 Parser & Filter Engine

En esta fase se aplican dos acciones:

- ***Parser***: se encarga de extraer más duplas, campo-valor.
  Normalmente, se emplea un *parser* para cada tipo de servicio dependiendo del formato original del log. También se puede emplear uno común, con diferentes reglas de extracción, si el formato es común para todos los servicios.
- ***Filter***: se filtra el log para que reciba un tratamiento especial, en función del valor de alguno de los campos obtenidos anteriormente.

2.2.1.3 Action Processor

Aplica la acción correspondiente para cada log, dependiendo del filtrado. Aquí se elige el sistema donde serán enviados los logs y la plantilla utilizada para cada servicio.

2.2.1.4 Queue

Tanto la *queue* inicial como las *action queues*, se encargan de ir encolando los logs recibidos, esto nos ayuda a mantener la secuencia temporal con la que fueron recibidos.

2.2.2 Módulos

En este apartado mostraremos las funcionalidades que presenta cada módulo utilizado. Para más tarde realizar un desarrollo, con dichos módulos, que nos ayude a cumplir con los requisitos especificados para esta herramienta. Como hay que cumplir dos tareas principales, dividiremos esta sección en sendos apartados, Normalización y Salida.

2.2.2.1 Normalización

En esta tarea se deben solucionar dos problemas, el paso de formato RFC5424 a JSON y el paso de formato JSON a JSON[6].

Para la normalización, RFC5424 a JSON, se puede utilizar el módulo:

- **mmnormalize**: Este módulo se basa en el previo conocimiento del log y de la librería liblognorm**[5]**. Funciona de forma equivalente a las expresiones regulares[7], pero presentando unas reglas más fáciles de interpretar; creadas en lenguaje legible, disminuyendo así el tiempo de aprendizaje. Estas reglas se aplican al contenido del campo MSG, ya que, si se cumple el formato RFC5424 el preprocesador ha obtenido los demás campos previamente.

  En la siguiente figura, Figura 2-2, se muestra como se debe crear una regla para que liblognorm obtenga los datos de interés contenidos en el campo MSG.

  Es tan sencillo como dar una palabra clave al campo que vamos a obtener (campo tag en la figura) y a continuación poner el tipo de dato qué es (campo match description en la figura), todo ello precedido de la palabra clave "rule=[8]".

```
rule=[<tag1>[,<tag2>...]]:<match description>
```

Figura 2-2. Formación reglas liblognorm

En la siguiente tabla, Tabla 2-1, mostramos los diferentes tipos de datos que soporta la librería de normalización liblognorm.

Tipo de dato	Descripción
number	Dígitos decimales
float	Dígitos coma flotante en forma no científica
hexnumber	Digito en formato hexadecimal
kernel-timestamp	Marca de tiempo en el formato kernel de Linux
whitespace	Espacios en blanco hasta la próxima palabra
string	Cadena de texto
word	Cadena de caracteres hasta el siguiente espacio en blanco o fin de línea

[6] Aunque este formateo parezca un tanto raro, esto es así porque se debe utilizar una variedad de JSON para que Rsyslog entienda que es JSON y poder aplicar el módulo pertinente

[7] Según el autor de liblognorm, el mismo que Rsyslog, su tiempo de computo es menor que una expresión regular

[8] También es soportada la palabra clave "rule=:".

string-to	Caracteres hasta la siguiente palabra indicada
aplha	Caracteres alfanuméricos hasta el siguiente espacio en blanco, puntuación, digito decimal o carácter de control
char-to	Caracteres hasta el siguiente carácter indicado
rest	Cero o mas caracteres hasta el fin de línea
quoted-string	Cadena de caracteres entre comillas dobles
op-quoted-string	Posiblemente cadena de caracteres entre comillas dobles
date-iso	Fecha en formato ISO (‘YYYY-MM-DD’)
time-24hr	Tiempo en formato ‘HH:MM:SS’
time-12hr	Tiempo en formato ‘HH:MM:SS’, llegando las HH hasta las 12
duration	Intervalo de tiempo transcurrido en formato ‘HH:MM:SS’
date-rfc3164	Fecha en formato RFC3164(M D HH:MM:SS)
date-rfc5424	Fecha en formato RFC5424(Y-M-DTHH:MM:SS)
ipv4	IP versión 4 formato decimal con punto
ipv6	IP versión 6 formato definido en RFC4291
mac48	Dirección MAC definida en el estándar IEE 802
cef	Formato CEF
checkpoint-lea	Formato LEA utilizado por checkpoint
cisco-interface-spec	Descripción interfaces cisco
iptables	Formato utilizado por Netfilter
json	Formato JSON
alternative	Te permite que un campo vaya en diferentes formatos
repeat	Te permite extraer una secuencia con el mismo patrón

Tabla 2–1. Tipos de datos soportados por liblognorm

Como se ha mostrado hay gran cantidad de datos soportados, incluyendo algunos de gran interés, como iptables, cisco o checkpoint.

En nuestro caso, solo utilizaremos alguno de ellos como: ipv4, word, string.

Por último, si el mensaje presenta un patrón que se repite en cada log se puede hacer uso de la opción prefix. Su definición es parecida a la de rule, incluyendo dentro del campo prefix match description los datos a extraer y su tipo.

```
prefix=<prefix match description>
```

Figura 2-3. Utilización de prefix

- **mmjsonparse**: Rsyslog tiene un modulo capaz de extraer los datos de un log que viene en formato JSON. Este *parser* funciona con el formato *lumberjack*[9], consistente en el formato base JSON añadiéndole la palabra clave @cee al principio del mapa clave-valor. Cumpliendo este formato el normalizador ya sabe que lo que sigue es formato JSON y es capaz de extraer los atributos con sus respectivos valores. A continuación se muestra como se debe de modificar el JSON original.

Ejemplo 2–1. *Ejemplo formato JSON formato lumberjack* .
@cee:{campo1:valor1, campo2:valor2,..., campoN:valorN}

Para más detalle de su aplicación , véase ANEXO A: Configuración Apache2, donde se han modificado los logs para cumplir dicho formato.

2.2.2.2 Salida

En nuestro caso haremos uso de dos módulos de salida. Uno al sistema Apache Kafka y otro a un fichero local, este último nos ayudará a hacer comprobaciones del formato y en la fase de depuración.

- **omkafka**: este módulo se encarga de crear y configurar un conector para utilizar el sistema Apache Kafka, dicho conector se llama productor, explicado con más detalle en 2.3Apache Kafka.
- **omfile**: este módulo se encarga de escribir los logs en el sistema de ficheros local. El único requisito es que el fichero exista y que Rsyslog tenga permiso de escritura sobre él.

2.2.3 Diseño del colector

Siguiendo la arquitectura general presentada en la sección anterior, debemos personalizar Rsyslog teniendo en cuenta que:

- Habrá 2 tipos de formato a la entrada:
  - JSON para el servicio Apache2
  - RFC5424 para el servicio ssh.

[9] Definido por el propio Rsyslog.

- El formato de salida será JSON.
- Se hará uso tanto del sistema Apache Kafka, como de un fichero local, para el almacenamiento de los logs.

Habiendo hecho estas distinciones y haciendo uso de los términos y módulos definidos en la sección anterior, quedan los siguientes diseños:

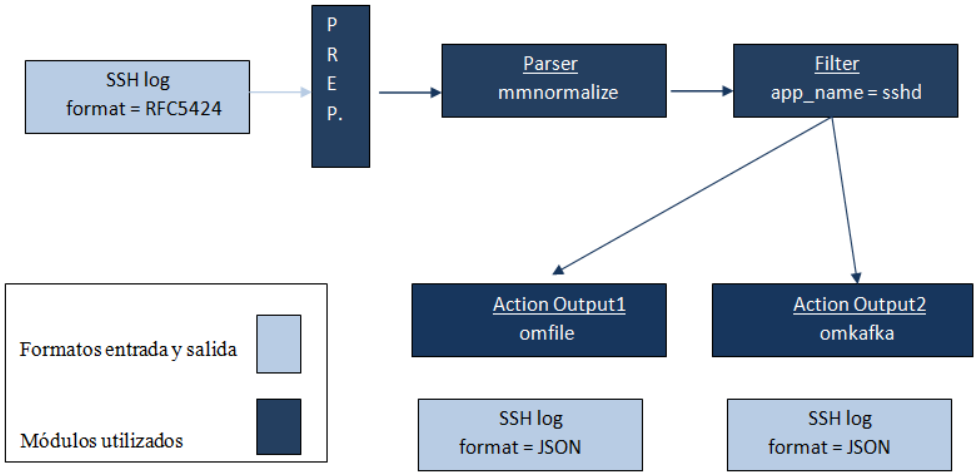

Figura 2-4. Diseño configuración Rsyslog para SSH

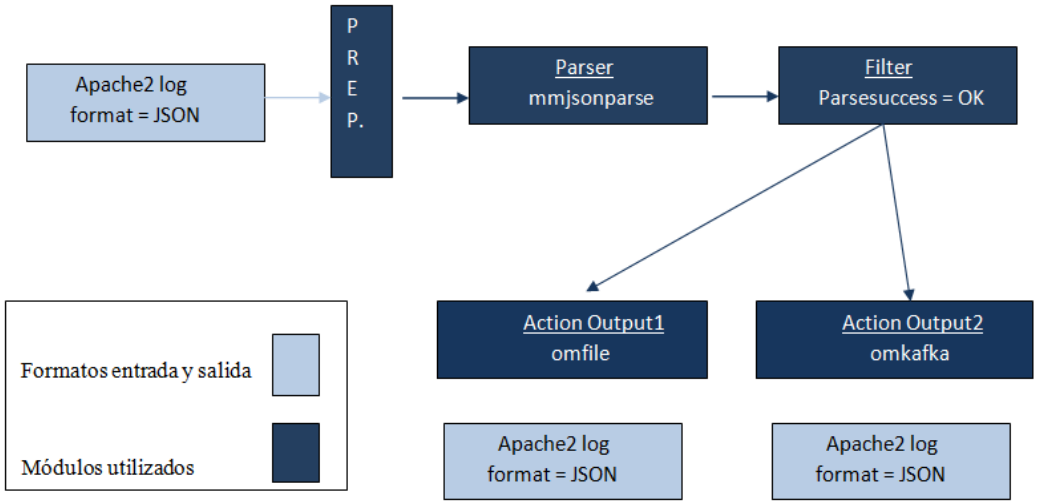

Figura 2-5. Diseño configuración Rsyslog para Apache2

2.3 Apache Kafka

Apache Kafka**[6]** es un sistema distribuido de mensajes de alto rendimiento y está basado en el diseño publicador-suscriptor. Este sistema ofrece:

- **Rapidez**: un solo servidor es capaz de manejar 100 lecturas/escrituras por segundo desde múltiples clientes.
- **Escalabilidad**: al estar basado en tecnología de clúster, es capaz de aumentar o disminuir el tamaño del clúster según la carga soportada, todo ello transparentemente para el usuario y sin caídas del sistema.
- **Persistencia**: los datos son replicados a través del clúster, lo que le capacita para prevenir pérdidas de datos. Cada servidor puede manejar terabytes de mensajes sin impacto sobre el rendimiento.
- **Distribuido desde el diseño**: ofreciendo garantía ante fallos y durabilidad.

Estas características lo hacen el apropiado para utilizarlo como sistema de almacenamiento.

No es objeto de este proyecto realizar un análisis detallado de dicho sistema, aún así, introduciremos algunos conceptos que nos ayuden a comprender como funciona y nos ayude a realizar los conectores con el motor de correlación.

2.3.1 Funcionamiento

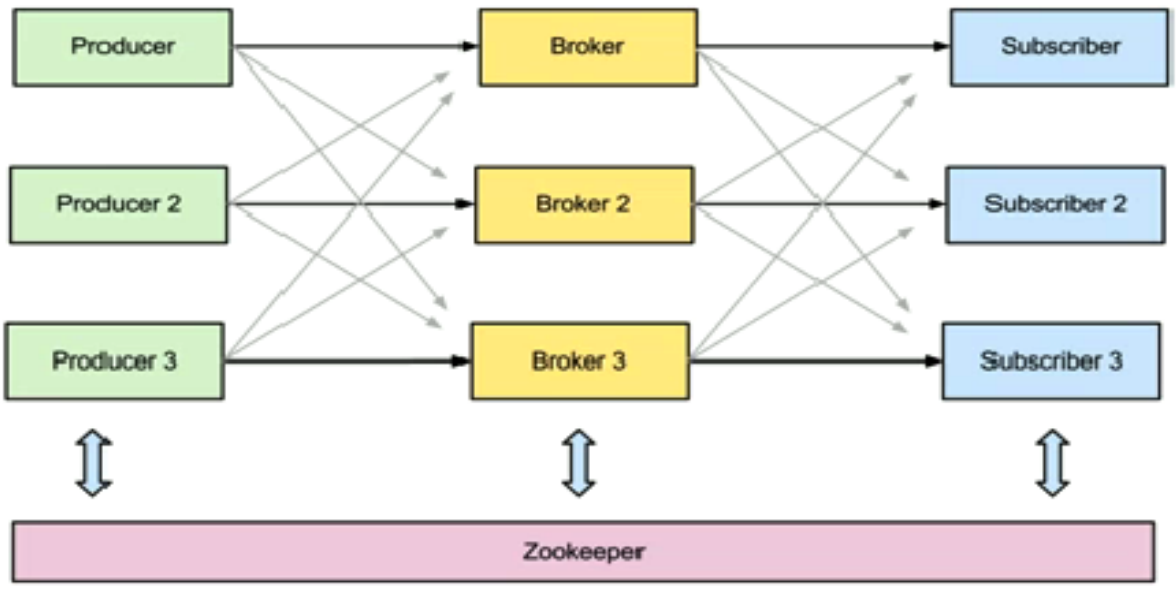

Figura 2-6. Arquitectura Apache Kafka

Como se comentó anteriormente este sistema está basado en el diseño publicador-suscriptor. Su comportamiento es muy simple: un cliente, productor, produce una serie de datos en un clasificador llamado *topic* y otro cliente, suscriptor, puede usar esos datos individualmente o mediante la pertenencia a un grupo. Para asegurar un consumo en orden a los mensajes producidos, al llegar al clúster, se les pone una marca denominada offset, facilitando así la forma de consumo(que puede ser configurada como queramos: desde el principio siempre, por dónde nos habíamos quedado…).

Por último, no podemos olvidar Zookeeper que ofrece una funcionalidad de alto interés al mantener un control de los líderes del clúster, denominado *broker*, y de los grupos que pueden consumir de dicho clúster.

Hemos introducido los conceptos básicos de este sistema que nos ayudarán a configurar el sistema Apache Kafka y a realizar los conectores con el motor de correlación.

2.3.2 Especificaciones

Realizaremos un entorno sencillo sin configuraciones complejas, es decir, un solo ordenador, que será el *broker*, sin replicas, y un *topic* para cada servicio que produzca datos.

También, usaremos la API Java que presenta dicho sistema para realizar un suscriptor que genere las entradas de Siddhi CEP y un productor que genere los mensajes con la información obtenida, tras su paso por el motor.

2.4 Siddhi CEP

Siddhi CEP**[7]** es un procesador de eventos complejos, basado en un diseño simple pero robusto. Es capaz de procesar grandes volúmenes de datos y realizar cálculos bastante costosos en otro tipo de sistemas, computacionalmente hablando, ofreciendo así, un tiempo de respuesta bastante bueno.

Este sistema será el encargado de aplicar la inteligencia sobre el conjunto de logs almacenados. Dicha inteligencia consiste en una serie de operaciones complejas que son capaces de extraer la información que el usuario desee.

2.4.1 Funcionamiento

En la siguiente figura se resume el comportamiento de dicho sistema:

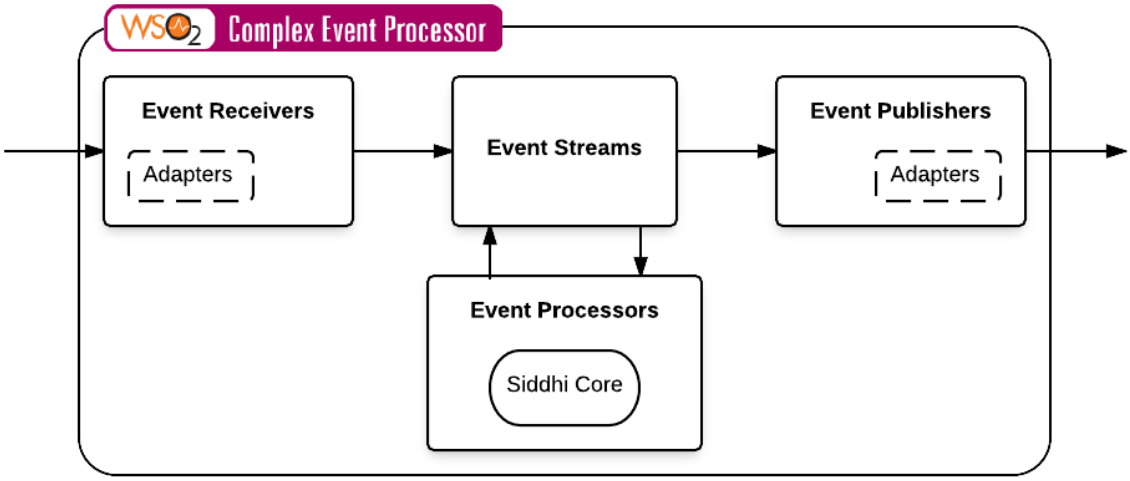

Figura 2-7. Arquitectura Software Siddhi CEP

A continuación desgranamos cada componente, para entender cómo funciona y poder hacer un uso, sencillo pero completo.

- **Event Receivers** : dicho receptor consume una serie de datos y genera el *stream*, al cual se aplican una serie de operaciones definidas. Como existen multitud de formatos a la entrada, dicho receptor se encarga de adecuar cada formato para generar un *stream* (formato con el que funciona dicho procesador de eventos).

- **Event Streams** : es una secuencia de eventos que siguen un esquema determinado. A la definición de dicho esquema se le denomina Event Stream Definition. El esquema está definido por un nombre único y lo conforman una serie de atributos que están compuestos por: el nombre(que deber ser único) y el tipo asociado a dicho atributo.

```
define stream <stream name> (<attribute name> <attribute type>,
<attribute name> <attribute type>,  … );
```

Figura 2-8. Como definir un stream en Siddhi

Para conformar un *stream*, Siddhi nos ofrece diferentes tipos de datos definidos en la siguiente tabla:

Tipo	Definición
string	Cadena de caracteres
int	Número decimal
long	Ídem a int, aumentando el rango
float	Número en coma flotante
double	Ídem a float, aumentando el rango
bool	Tipo que puede tomar verdadero o falso
object	Tipo que tiene asociado una serie de atributos

Tabla 2–2. Tipos de datos soportados por Siddhi CEP

- **Event Processors** : se encarga de definir las operaciones para un determinado *stream* y asociarle un disparador, que se ejecute al utilizar dichas operaciones; el disparador a su vez tiene asociado un Event Publisher.

  Básicamente, el procesador de eventos define unas operaciones, denominadas *queries*, sobre un *stream* de entrada y genera un *stream* de salida. Las *queries* tienen un formato similar a SQL, denominado SiddhiQL[10] **[8]**. Dichas *queries* unidas al *stream* de entrada y la definición de los manejadores asociados a cada *query*, definen un plan de ejecución. A continuación, se muestra como se define una *query*.

```
from <input stream name>
select <attribute name>, <attribute name>, ...
insert into <output stream name>
```

Figura 2-9. Como definir una query en Siddhi

- **Event Publisher** : realiza la funcionalidad inversa al *event receivers*, es decir, toma como entrada los datos que generan las operaciones, en formato *stream* y lo adecua al formato del sistema de salida.

[10] Este formato facilita el aprendizaje si se tienen los conocimientos básicos sobre SQL

En base a lo comentado, siddhi CEP ofrece una variedad de operaciones que te permite complicar el diseño. Por ejemplo, te permite unir flujos de datos, es decir, diferentes fuentes pueden presentar una *query* que derive en otro *stream* que junte las diferentes fuentes y obtener así, información compleja(p.ej: capacidad de carga que puede soportar una maquina con diferentes servicios, cuantas medidas por segundo toma determinado sensor, etc).

2.4.2 Operaciones

Como se ha comentado, sobre cada *stream* se pueden realizar una serie de operaciones, a continuación se muestran las definiciones y funcionalidades de cada una de ellas.

2.4.2.1 Proyecciones de query

Estas operaciones modifican un *stream* para generar otro.

Acción	Descripción	Uso[11]
Selección atributos	Selección de algunos atributos del stream de entrada para ser insertados en el stream de salida	from <input stream name> **select <attribute name>, <attribute name>, ...** insert into <output stream name>;
Selección de todos los atributos	Selección de todos los atributos de entrada para ser insertados en el stream de Salida	from <input stream name> **[select *]** insert into <output stream name>;
Renombrado de atributos	Cambio de nombre de un atributo en el stream de salida respecto al stream de entrada	from <input stream name> **select <attribute name> as <outAttributeName>** insert into <output stream name>;
Crear atributo	Como crear un atributo en el stream de salida con un valor por defecto	from <input stream name> select <attribute name>, **'Text' as <newAttributeName>** insert into <output stream name>;

[11] Las secuencias entre los caracteres [], indican el carácter opcional de estas opciones.

Uso de operaciones lógicas y matemáticas	Indica las operaciones aritmético-lógicas, definidas en Siddhi. Se muestran en el orden de prioridad.	• () • IS NULL • NOT • * / % • + - • < <= > >= • == != • IN • AND • OR

Tabla 2–3. Operaciones de proyección

2.4.2.2 Filtros

Estas operaciones permiten incluir una condición de control para generar el stream de salida.

Acción	Descripción	Uso[12]
Filtrado según condición	Selección de algunos atributos del stream de entrada para ser insertados en el stream de salida	from <input stream name>**[<filter condition>]** select <attribute name>, <attribute name>, ... insert into <output stream name>;
	Operaciones que se pueden realizar en la condición	• < , <=, >, >=, ==, !=, and, or, not : tienen el mismo significado que en otros lenguajes. • contains : te dice si un atributo, definido como string, contiene la secuencia por la que preguntas. • instanceof : comprueba el tipo de dato que es el atributo

Tabla 2–4. Operaciones de filtrado

[12] Las secuencias entre los caracteres [], indican el carácter opcional de estas opciones.

2.4.2.3 Windows

Estas operaciones trabajan sobre un subconjunto limitado de eventos, ya sea en tiempo o número, denominado ventana . Antes de explicar los diferentes tipos de ventanas debemos definir los tipos de eventos, según la perspectiva de una ventana.

Tipo[13]	Definición
current-events	La query solo tiene efecto a la llegada de un evento a la ventana
expired-events	La query emitirá resultados cuando los eventos expiren, es decir salgan de la ventana
all-events	Se emiten resultados tanto a la llegada de un evento como a la salida de éstos de la ventana

Tabla 2–5. Tipos de eventos

Al operar sobre conjuntos también se definen una serie de operaciones de agregación bastante útiles:

Operación	Definición
sum	Realiza una suma acumulada sobre determinado atributo
avg	Realiza la media aritmética sobre un atributo especifico
max	Obtiene el máximo valor, para el atributo, presente en los eventos tratados
min	Obtiene el máximo valor, para el atributo, presente en los eventos tratados
count	Enumera el atributo especificado, de los eventos tratados

Tabla 2–6. Operaciones de agregación

Teniendo en mente estas operaciones y definiciones de eventos, explicaremos los distintos tipos de ventanas que nos ofrece siddhi CEP:

Tipo	Descripción	Uso[14]
Length window	Ventana que mantiene los últimos N eventos producidos.	from <input stream name>[<filter condition>]**#window.length(N)** select <attribute name>, <attribute name>, ... insert into <output stream name>

[13] Si no se usa ninguna especificación, siddhi toma por defecto la opción 'current-events'
[14] Las secuencias entre los caracteres [], indican el carácter opcional de estas opciones.

Time window	Ventana que mantiene los eventos recibidos dentro del último periodo T.	from <input stream name>[<filter condition>]**#window.time(T)** select <attribute name>, <attribute name>, ... insert into <output stream name>;
Time batch window	Ventana que procesa los eventos por lotes. Recolecta los eventos llegados dentro del último periodo T y los agrupa en un lote.	from <input stream name>[<filter condition>]**#window.timeBatch(T)** select <attribute name>, <attribute name>, ... insert into <output stream name>;
Length batch window	Emite los eventos como un lote a la llegada del N-esimo evento.	from <input stream name>[<filter condition>]**#window.lengthBatch(N)** select <attribute name>, <attribute name>, ... insert into <output stream name>;
Unique window	Se queda con los últimos eventos que son únicos de acuerdo con el atributo dado.	from <input stream name>[<filter condition>]**#window.unique(Attribute)** select <attribute name>, <attribute name>, ... insert into <output stream name>;
Firs unique window	Se queda con los primeros eventos que son únicos de acuerdo al atributo dado.	from <input stream name>[<filter condition>]**#window.firstUnique(Attribute)** select <attribute name>, <attribute name>, ... insert into <output stream name>;
External time window	Ventana capaz de procesar acorde a la fecha y hora suministrada por el stream de entrada, en vez de aplicar la fecha y hora de la maquina donde se aloja siddhi. Se indica la fecha y hora que se utiliza(timestamp) y el periodo de ejecución de la ventana(T).	from <input stream name>[<filter condition>]**#window.externalTime(timestamp, T)** select <attribute name>, <attribute name>, ... insert into <output stream name>;

Tabla 2–7. Tipos de ventana

Se ha hablado de periodos temporales, así que, a continuación de muestran las unidades temporales que soporta siddhi CEP.

Unidad	Sintaxis
Year	year | years
Month	month | months
Week	week | weeks
Day	day | days
Hour	hour | hours
Minutes	minute | minutes | min
Seconds	second | seconds | sec
Milliseconds	millisecond | milliseconds

Tabla 2–8. Unidades soportadas por las ventanas temporales

2.4.2.4 Joins

Esta operación se encarga de unir *streams*.

Operación	Descripción	Uso[15]
Join	1. Toma dos streams como entrada. 2. Cada stream debe tener asociada una ventana. 3. Genera como salida un evento de cada stream 4. Con la opción on, sidhhi solo unirá los eventos que cumplan la condición. 5. Con la opción within, siddhi solo unirá los eventos que estén dentro del periodo temporal indicado en la condición. 6. Con la opción unidirectional podremos indicar que un flujo es dominante sobre el otro, es decir solo el stream que marque esta opción podrá ejecutar el proceso de join. 7. Existen diferentes tipos de join, aunque en la versión utilizada, 3.0 de SiddhiQL), solo se encuentra disponible inner join.	from <stream>#<window> **[unidirectional]** **join** <stream>#<window> **[unidirectional]** **[on <condition>] [within <time>]** select ({<attribute-name>}| '*') insert [<output-type>] into <stream-name>

Tabla 2–9. Operación Join

[15] Las secuencias entre los caracteres [], indican el carácter opcional de estas opciones.

2.4.2.5 Patterns

Esta operación es capaz de obtener patrones de uno o varios *streams*.

Operación	Descripción	Uso[16]
Pattern	1. Toma uno o mas streams como entrada. 2. Obtiene concordancias entre los eventos a través de una serie de ocurrencias pre relacionales o postrelacionales. 3. Los streams de entrada deben estar identificados unívocamente. 4. Cualquier evento a la salida es una colección de eventos que cumplen con el patrón establecido. 5. Los atributos seleccionados deben ser renombrados. 6. Con la opción every, la query se ejecuta cada vez que un evento cumpla el patrón. Si no se establece la query solo se ejecuta una vez. 7. Con la opción within<time>, indicamos que no puede superarse dicho tiempo entre la llegada del primer y último evento que conforman el patrón. 8. Se pueden establecer las ocurrencias de los eventos que conforman el patrón, añadiendo <mínimo : máximo> tras la definición de los eventos que conforman el patrón. Con esto indicamos el número de eventos para que el patrón se cumpla.	from [every] **<stream> -> [every] <stream> ... <stream> within <time>** select <attribute-name> {,<attribute-name>} insert into <stream-name> partition by <partition-id>

Tabla 2–10. Operación Patterns

2.4.2.6 Sequences

Similar a Patterns, se diferencian en que *patterns* pueden coincidir eventos que cumplen la condición del patrón pero difieren en lo demás. Con *sequences*, se asegura que la secuencia obtenida, entre primer y último evento, es idénticamente igual.

[16] Las secuencias entre los caracteres [], indican el carácter opcional de estas opciones.

Operación	Descripción	Uso[17]
Pattern	1. Opera sobre uno o mas streams. 2. Toma como entradas streams definidos en formato de expresiones regulares. 3. Los streams de entrada deben estar identificados unívocamente. 4. Cualquier evento a la salida es una colección de eventos que cumplen exactamente con el orden del patrón establecidos 5. Los atributos seleccionados deben ser renombrados. 6. Con la opción within<time>, indicamos que no puede superarse dicho tiempo entre la llegada del primer y último evento que conforman el patrón. 7. Se pueden establecer las ocurrencias de los eventos que conforman el patrón, en este caso usando expresiones regulares: a. * : Cero o más coincidencias. b. + : Una o más coincidencias. c. ? : Cero o una coincidencia.	from **<event-regular-expression-of-streams>** within <time> select <attribute-name> {, <attribute-name>} insert into <stream-name>;

Tabla 2–11. Operación Sequences

2.4.2.7 Otras operaciones

Se han definido las operaciones más importantes y utilizadas, aunque no son las únicas. A continuación, se muestran otras operaciones interesantes, que completan este apartado.

Operación	Descripción	Uso[18]
Output rate limiting	Te permite fijar la tasa de salida de la query. • El elemento output-type puede tomar los siguientes valores: ○ every ○ last ○ all	from <stream-name> select ({<attribute-name>}| '*'|) **output { (<output-type>)? every (<time-expr> | <event-interval> events)** } | { **snaphost every <time-expr>** } insert into <stream-name>;

[17] Las secuencias entre los caracteres [], indican el carácter opcional de estas opciones.
[18] Las secuencias entre los caracteres [], indican el carácter opcional de estas opciones.

	• Se puede indicar el periodo en forma de tiempo o de eventos, time-expr y event-interval respectivamente. • Con la opción snapshot podemos ofrecer una instantánea del stream. Esta opción debe usarse conjuntamente con una operación de tipo window.	
In-built functions	• Convert. Convierte un tipo de dato primitivo en otro. Tambien puede cambiar el tipo de dato. • Coalesce. Selecciona el valor del primer atributo que sea no nulo. • IsMatch. Te permite comparar un atributo con una expresión regular para comprobar su coincidencia. Devuelve true o false en función de la coincidencia. • Concat. Te permite unir dos atributos de tipo String en un único valor.	• convert(attribute, primitive data type) • convert(attribute1, primitive data type, attribute2) • coalesce(attribute1,…,attributeN) • isMatch(regular expression, attribute) • concat(String attribute1, String attribute2)
Event Tables	Te permite crear una tabla, en vez de un stream, y utilizarlo como salida.	• **Creación** define table table_name (attributename data type, attributename data type); • **Inserción de una fila** from nameStream select attribute1, attribute2 insert into event-table-name; • **Eliminación de una fila** from nameStream delete raw on (remove condition);

Tabla 2–12. Otras operaciones

2.4.3 Especificaciones

Como se ha comentado en la sección, 2.4.1. Funcionamiento, siddhi es capaz de manejar grandes volúmenes de datos y realizar operaciones bastantes complejas.

Nuestro propósito no es llevar al extremo al sistema siddhi CEP, sino que, nos centraremos en un diseño bastante simple, que sea capaz de mostrar sus armas, y como en un futuro se pueden ampliar las funcionalidades, alcanzando mayor grado de complejidad.

Así que, realizaremos un diseño basado en un *stream* de entrada, realizando la mayoría de operaciones posibles mediante *queries* y generando los *streams* asociados a éstas.

2.5 API Rest

Como se comentó la sección, 1.1. Alcance, se debe dotar al usuario de ciertas capacidades. Para que estas capacidades se puedan llevar a cabo, se hace uso de una API Rest. Esta API recibirá peticiones HTTP en formato JSON y responderá mediante HTTP, con el código apropiado, según se haya desarrollado la acción planteada por el usuario.

2.6 Diseño UML

Una vez mostrado el funcionamiento de todas las herramientas procedamos a realizar un diseño**[9]** que aúne todas las funcionalidades de las herramientas mostradas y nos ayude a unir cada sistema simple, para formar una herramienta compleja.

2.6.1 Diagrama de casos de uso

En el siguiente diagrama se muestran, los requisitos que dotan al usuario de la capacidad para modificar el comportamiento de siddhi CEP.

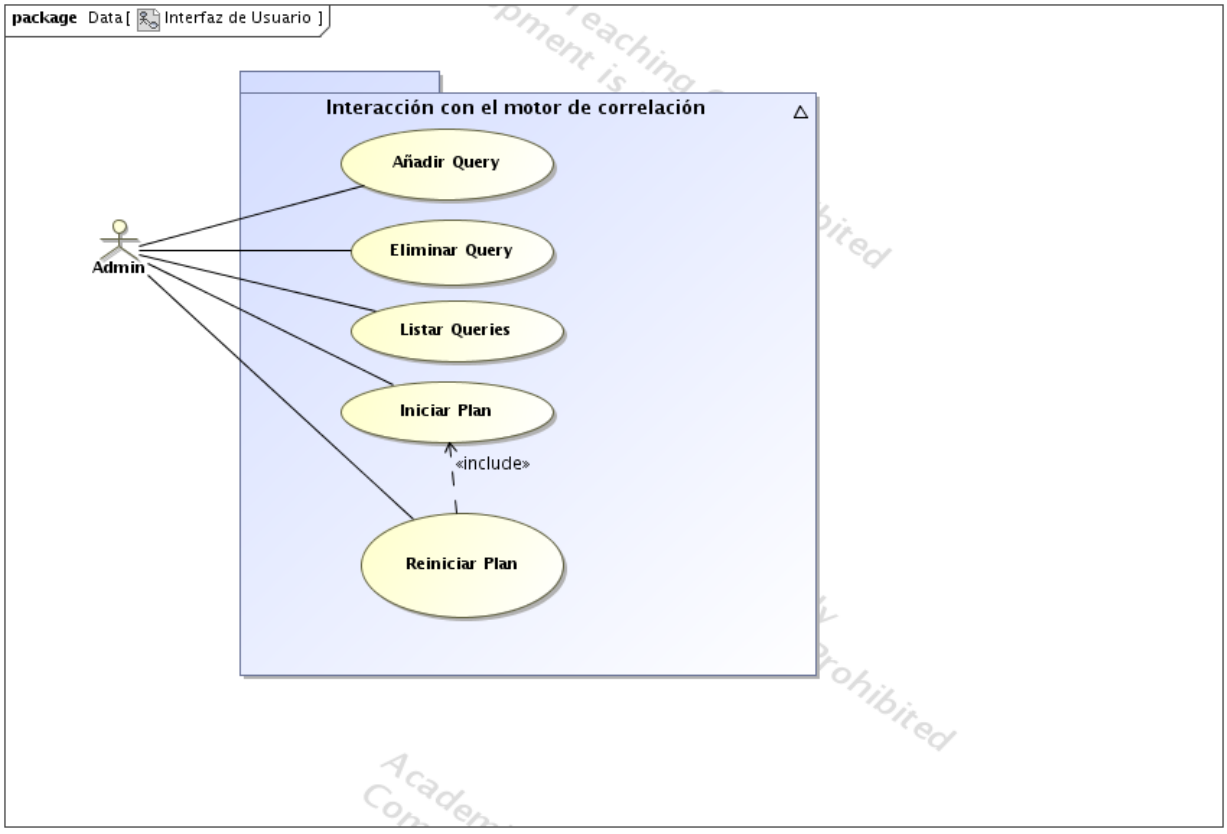

Figura 2-10. Diagrama Casos de Uso para Administrado

2.6.2 Diagrama de clases para siddhi CEP-API Rest

Para hacer uso de Siddhi, se necesita una clase intermedia capaz de gestionar la entrada, la salida, el propio Siddhi y las interacciones con el usuario. A dicha clase la denominaremos SiddhiHandler, controlará el consumo y la producción a Apache Kafka, mediante la creación y configuración de un consumidor y productor, respectivamente.

Las interacciones con Apache Kafka no se recogen en el diagrama porque ya hay una API que implementa dicho comportamiento, nosotros solo personalizaremos el comportamiento para generar los *streams* de entrada/salida del motor. No se harán, por tanto, modificaciones del comportamiento definido en la API que proporciona dicho sistema.

Por último, la clase *resource* se encargará de asociar las peticiones HTTP recibidas a un método que modifique los parámetros necesarios en la clase SiddhiHandler.

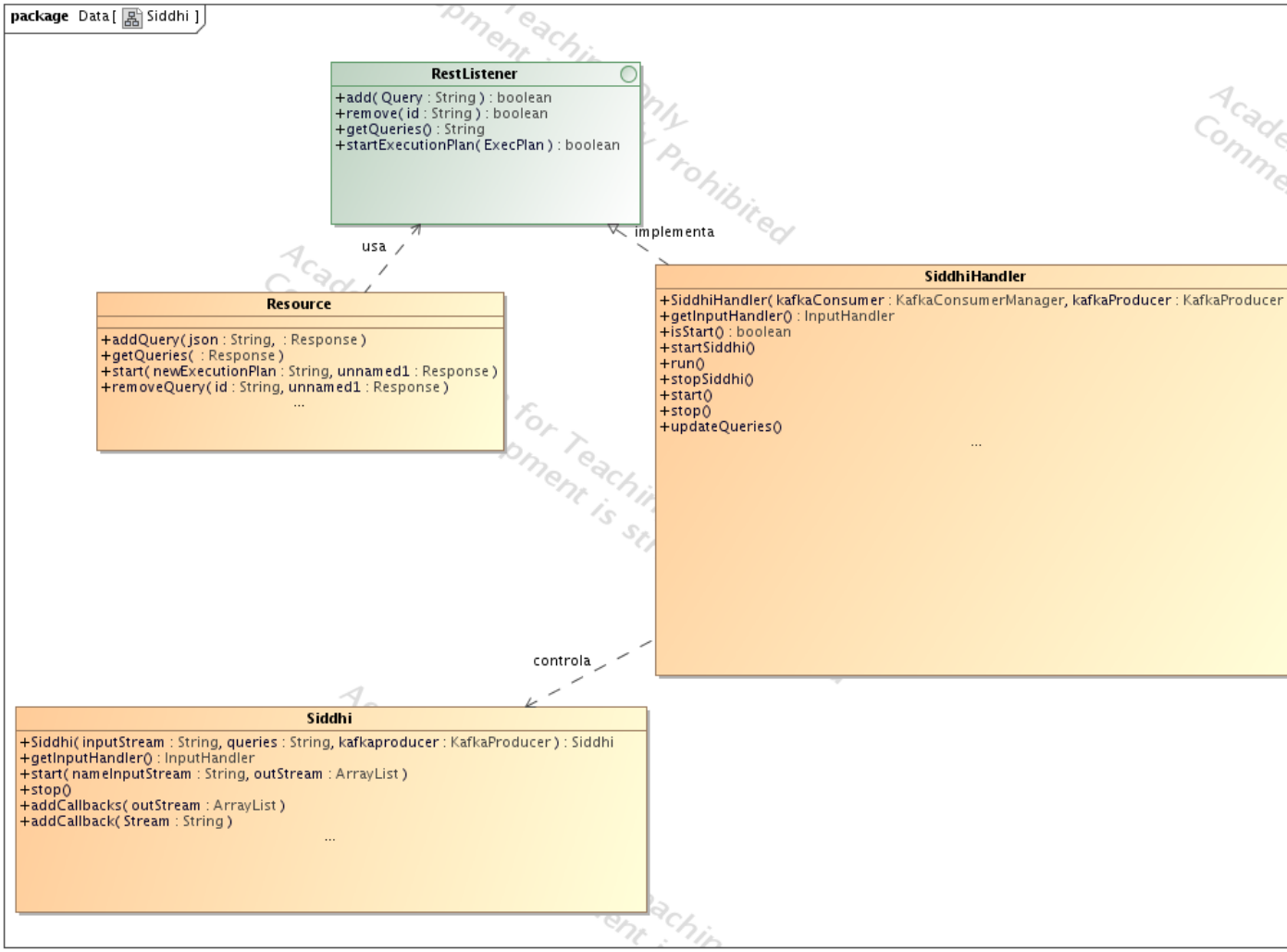

Figura 2-11. Diagrama de Clases Siddhi CEP

2.6.3 Diagramas de paso mensaje Usuario-Manejador Siddhi

A continuación, mostramos el comportamiento que deberá tener la herramienta al interactuar con el usuario.

En el siguiente diagrama se muestra, las interacciones que suceden a la hora de realizar un petición de incluir una *query*. A la hora de recibir una petición de esta características se debe comprobar que la id y el *stream* de salida asociado a la *query* no estén en uso.

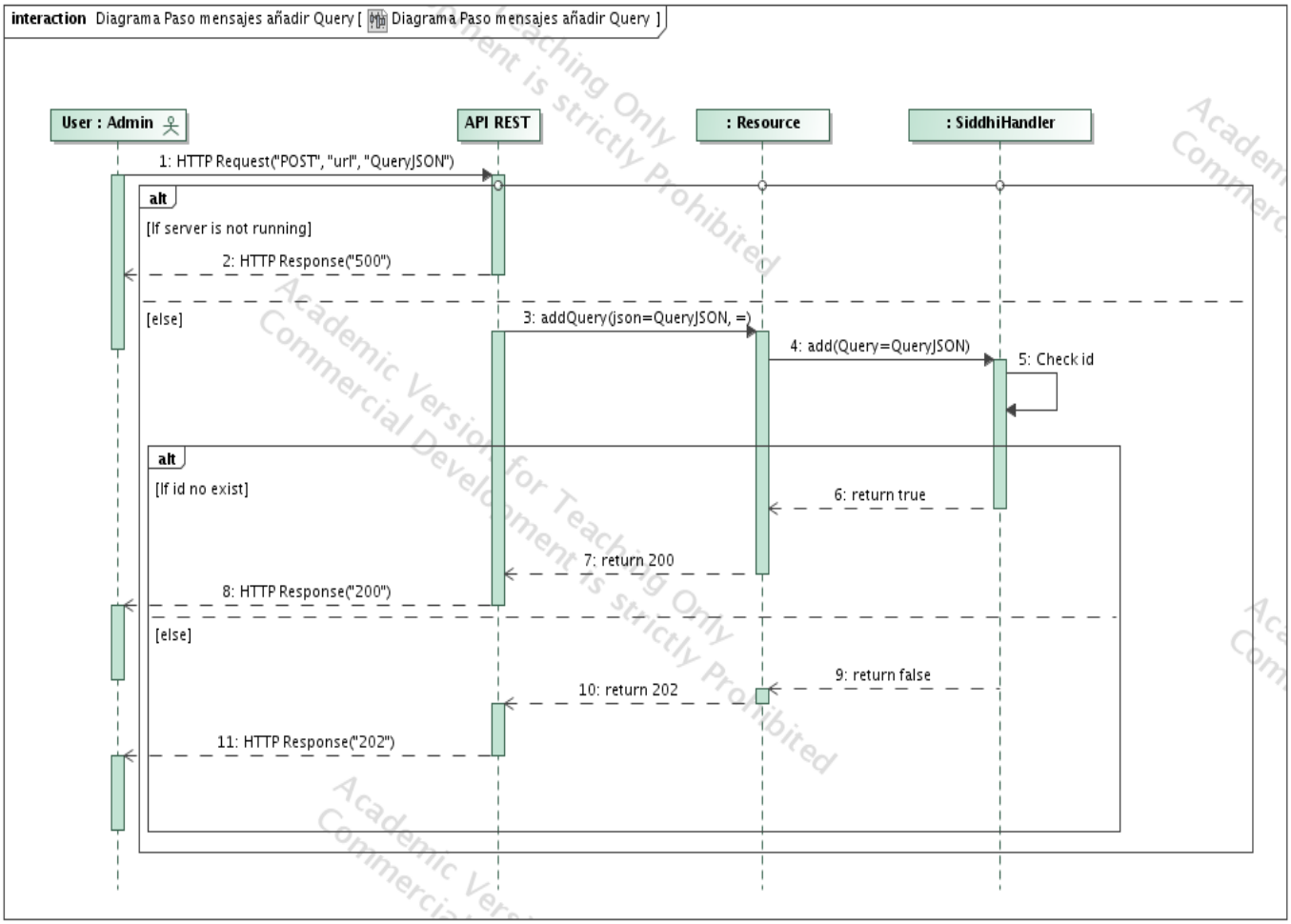

Figura 2-12. Diagrama Paso Mensajes(Inclusión Query)

En el siguiente diagrama se muestra, las interacciones que suceden a la hora de realizar un petición de eliminar una *query*. A la hora de eliminar una *query* se debe comprobar que el id de la *query* exista.

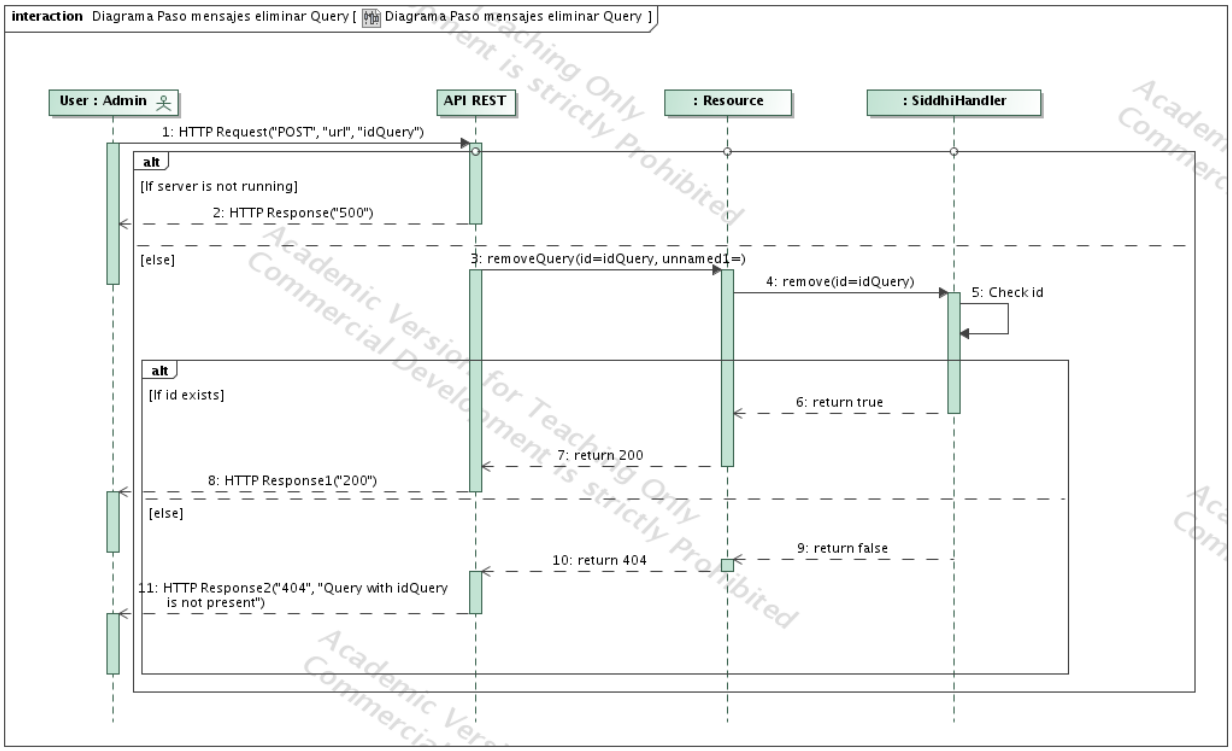

Figura 2-13. Diagrama Paso Mensajes(Eliminación Query)

Por último, en lo referente al apartado *queries*, se ofrece la posibilidad de obtener las *queries* definidas hasta el momento. Ayudando así, a llevar un control de las *queries* definidas.

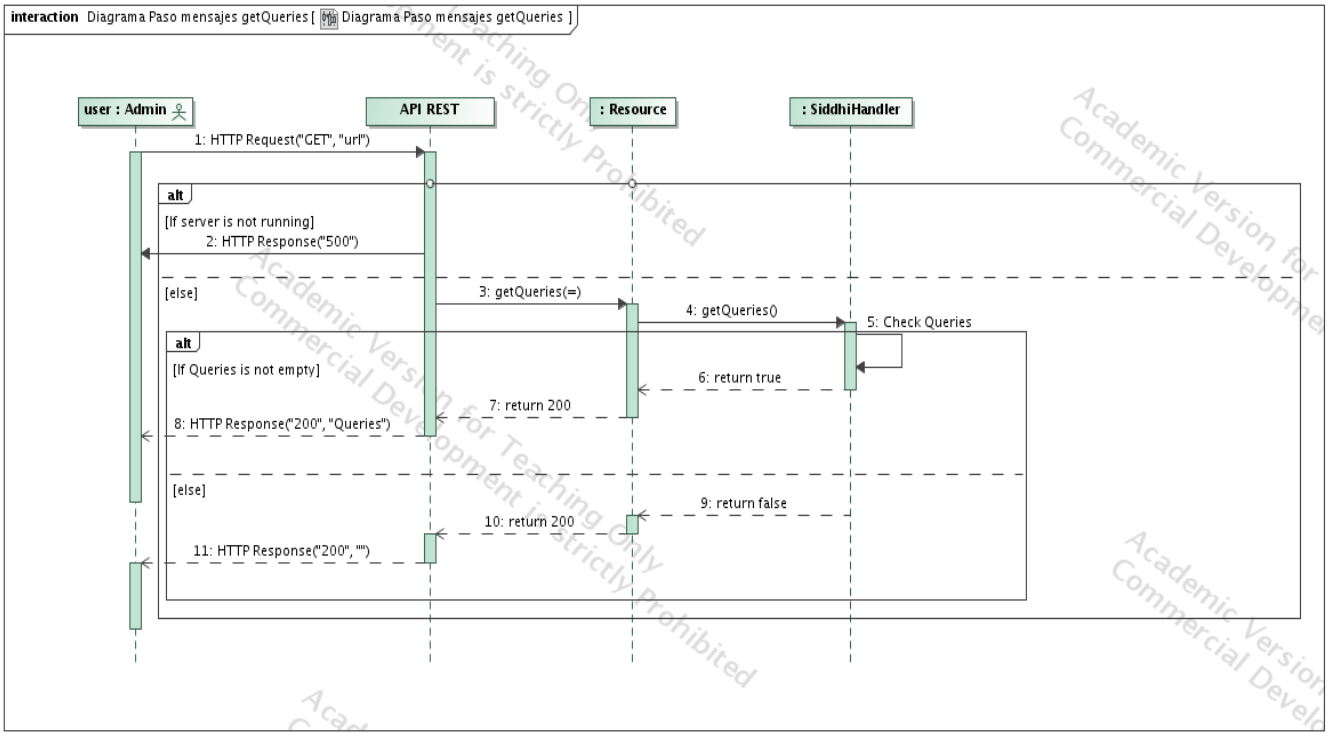

Figura 2-14. Diagrama Paso Mensajes(Obtener Queries definidas)

En el siguiente diagrama se muestra como se tratan las peticiones de inicio. En este apartado se muestra un comportamiento especial, asociado al reinicio de un plan por modificaciones en el plan de ejecución ya existente; liberando así, al usuario de tener que definir el *stream* de entrada nuevamente.

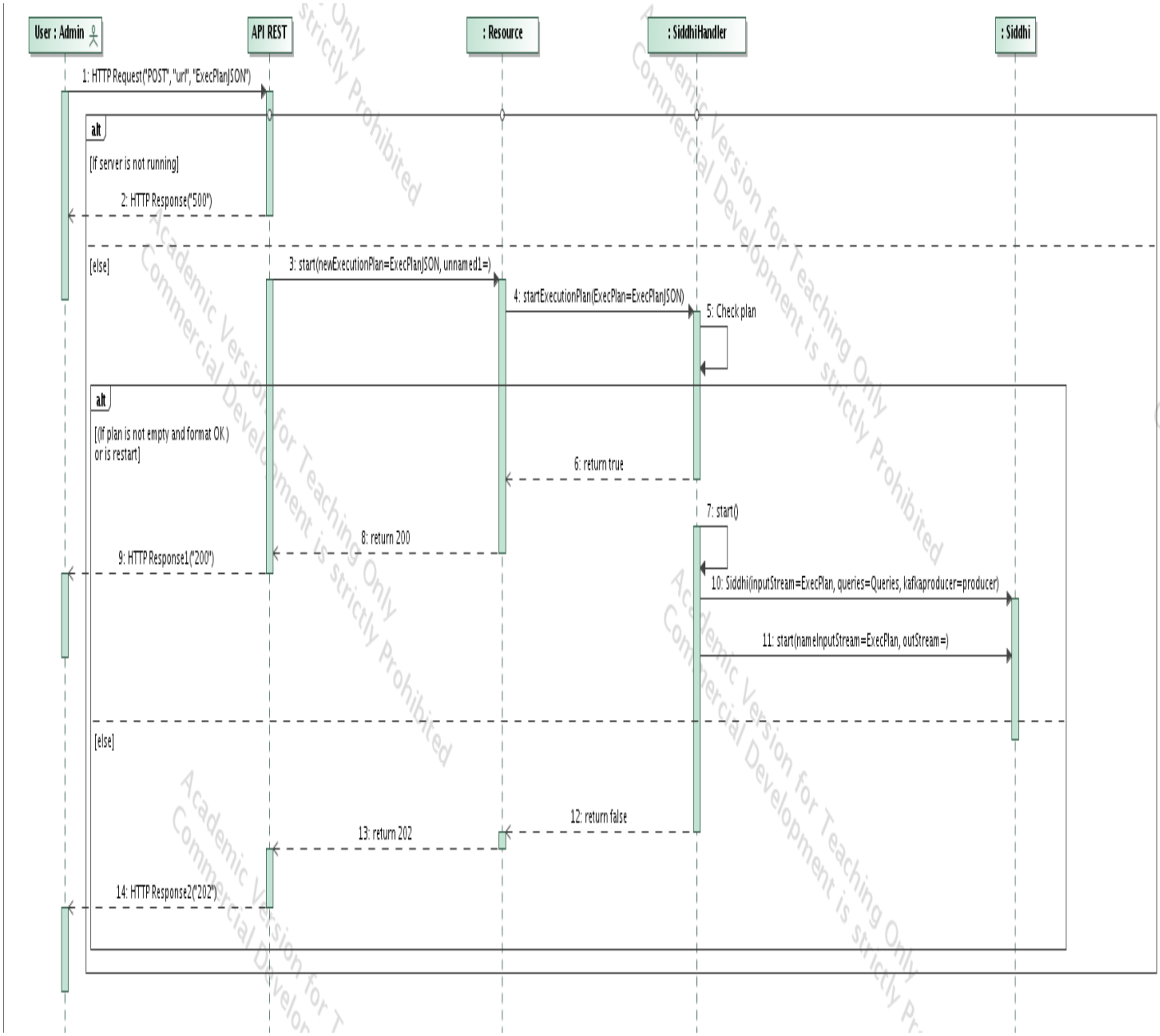

Figura 2-15. Diagrama Paso Mensajes(Inicio/Reinicio del motor)

2.7 Arquitectura detallada

En nuestro caso haremos una primera aproximación al uso de la herramienta, centrándonos en dos formatos de entrada, JSON y RFC 5424, y realizando operaciones sencillas sobre dichas entradas sin realizar cálculos que compliquen la arquitectura, es decir, sólo realizaremos operaciones sobre un *stream* y generaremos, también, un solo un *stream* de salida.

Poniendo en común todo lo definido en este capítulo, podemos realizar una primera aproximación al diseño final.

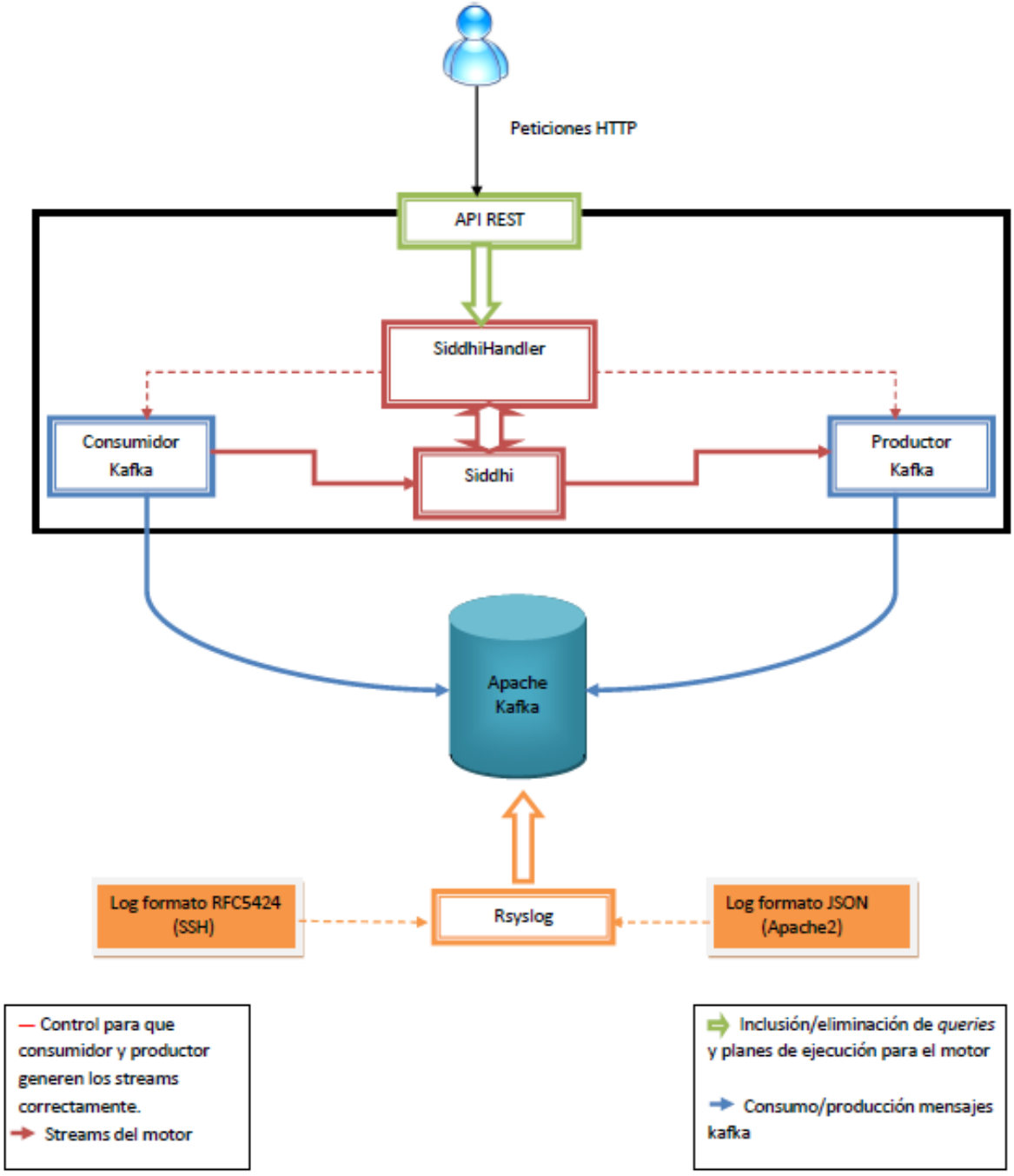

Figura 2-16. Diseño Final

3 PREPARACIÓN DEL ENTORNO

Pessimists, we're told, look at a glass containing 50% air and 50% water and see it as half empty. Optimists, in contrast, see it as half full. Engineers, of course, understand the glass is twice as big as it needs to be

- Bob Lewis -

En este capítulo mostraremos como iniciar el entorno, instalar y configurar los diferentes sistemas que conforman el proyecto y las herramientas utilizadas para su desarrollo.

3.1 Condiciones iniciales

En primer lugar definiremos algunas condiciones previas:

- Este proyecto ha sido desarrollado, en su totalidad, en:
  - Un ordenador con procesador de la familia i7, 8GB RAM y 1 TB almacenamiento en disco.
  - Sistema operativo Ubuntu 14.04 LTS.
- Algunas pruebas se realizaron mediante el conexionado remoto a una máquina virtual, en la cual corría un sistema operativo CentOS.
- Indispensable el manejo de la Shell de Linux(conexiones remotas, uso de comandos básicos, editado de configuraciones con editores base, conocimiento de scripts de servicio…).
- Operar con una JVM versión 1.6(o superior).

3.2 Rsyslog

Si se opera en un sistema UNIX es probable que Rsyslog sea el sistema de logging configurado por defecto en el sistema operativo. Para que Rsyslog ofrezca las funcionalidades necesarias, definidas en el 2.2.2 , para cumplir los objetivos propuestos, se necesita la versión 8.7.0 o superior. Aun teniendo una versión válida 8.20.0, se muestra, a continuación, una guía de instalación y otra de configuración.

3.2.1 Requisitos del sistema

Como mínimo el sistema debe presentar las siguientes herramientas de desarrollo:

- C compiler (normalmente gcc)
- make
- libtool
- rst2man (parte de herramientas de documentación de Python, si quieres generar los ficheros man)
- Bison and Flex (preferiblemente, en otro caso yacc y lex)
- zlib development package (normalmente *libz-dev*)
- json-c (normalmente llamado *libjson0-dev* ó similar)
- libuuid (normalmente *uuid-dev*, si no se encuentra usar –disable-uuid)
- libgcrypt (normalmente *libgcrypt-dev*)
- liblogging (sólo el componente stdlog es un requisito primordial)
- libestr

3.2.2 Instalación

1. Descargamos la versión 8.7.0[19]
2. Nos situamos en el directorio donde hayamos descargado el archivo y descomprimimos

   ```
   >tar xzf rsyslog-8.7.0
   ```

3. Todos los módulos necesarios excepto, mmjsonparse, mmnormalize y omkafka, vienen incluidos en la configuración por defecto. Para instalar los módulos hay dos formas de proceder:

   1. Manualmente módulo por módulo

      ```
      >cd rsyslog-8.7.0/
      >cd plugins/mmjsonparse/
      >make
      >make install
      ```

   2. Al configurar el servicio

      ```
      >./configure  - -enable-omkafka
      ```

4. Una vez compilados los módulos, instalamos el servicio.

   ```
   >./configure [Obligatorio si en el punto anterior hemos optado por la primera opción]
   >make
   >make install
   ```

[19] http://www.rsyslog.com/downloads/download-v8-stable/

3.2.3 Configuración

Como en el sistema ya teníamos instalado el servicio Rsyslog previamente, ya existe un script de servicio para manejarlo, pero que ejecuta la versión antigua. Para cambiar dicho comportamiento nos dirigiremos al script de inicio y cambiaremos la ruta del ejecutable a la nueva versión instalada.

```
jmf@jmf:~$ cat /etc/init.d/rsyslog
#! /bin/sh
### BEGIN INIT INFO
# Provides:          rsyslog
# Required-Start:    $remote_fs $time
# Required-Stop:     umountnfs $time
# X-Stop-After:      sendsigs
# Default-Start:     2
# Default-Stop:      0 1 6
# Short-Description: enhanced syslogd
# Description:       Rsyslog is an enhanced multi-threaded syslogd.
#                    It is quite compatible to stock sysklogd and can be
#                    used as a drop-in replacement.
### END INIT INFO

#
# Author: Michael Biebl <biebl@debian.org>
#

# PATH should only include /usr/* if it runs after the mountnfs.sh script
PATH=/sbin:/usr/sbin:/bin:/usr/bin
DESC="enhanced syslogd"
NAME=rsyslog

RSYSLOGD=rsyslogd
RSYSLOGD_BIN=/usr/local/sbin/rsyslogd
RSYSLOGD_OPTIONS="-c5"
RSYSLOGD_PIDFILE=/var/run/rsyslogd.pid

SCRIPTNAME=/etc/init.d/$NAME

# Exit if the package is not installed
[ -x "$RSYSLOGD_BIN" ] || exit 0

# Read configuration variable file if it is present
[ -r /etc/default/$NAME ] && . /etc/default/$NAME
```

Figura 3-1. Modificación script de servicio Rsyslog

Una vez realizado el cambio debemos reiniciar el servicio[20]:

```
>sudo service rsyslog restart
```

3.2.3.1 Estructura de ficheros

- **/etc/rsyslog.conf** : fichero de configuración general. En él, se recoge la configuración de parámetros generales y comunes a todas las configuraciones existentes.
- **/etc/rsyslog.d/** : directorio donde se almacenarán las configuraciones, normalmente un fichero de configuración por servicio.
- **/usr/local/sbin/rsyslogd** : ejecutable asociado al servicio.
- **/etc/init.d/rsyslog** : script de servicio de Rsyslog. Llamando[21] a este script con los parámetros correspondientes podemos iniciar, parar, reiniciar, comprobar estado del servicio…

[20] Cada vez que apliquemos un cambio en la configuración, deberemos ejecutar dicho comando
[21] Tambien se podría hacer uso del comando service rsyslog [action], evitando asi tener que recordar la ruta al script

3.2.3.2 Estructura de un fichero de configuración

Una vez instalado y puesto en marcha el servicio, describiremos el esqueleto que todo fichero de configuración debe cumplir.

```
#Cargar módulos

#Definición de plantillas

#Obtención de campos mediante los módulos "parser"

#Filtrado

        #Ejecutamos acción de salida según filtrado
```

Figura 3-2. Esqueleto fichero configuración Rsyslog

3.3 Apache Kafka

3.3.1 Instalación

En principio no importa la versión utilizada, eso sí, se recomienda utilizar una versión estable. En mi caso he utilizado la versión 0.8.2.0.

Como esta herramienta contiene los ejecutables empaquetados en .jar, no hace falta realizar ninguna compilación, basta con tener la versión pertinente de la máquina virtual de Java. Así, para su instalación basta con descargar el fichero comprimido y descomprimir:

1. Descargamos la versión 0.8.2.0[22].
2. Descomprimimos

   ```
   >tar -xvzf kafka_2.10-0.8.2.0.tgz
   ```

3.3.2 Estructura de ficheros

Una vez descomprimido ya tenemos los ejecutables listos para usarse. Este proyecto presenta una serie de scripts que nos facilitan las tareas, abstrayéndonos de la complejidad de dicho sistema. A continuación, mostramos los scripts que nos ayudarán a la hora de realizar nuestro trabajo.

- **kafka_2.10-0.8.2/bin/** : directorio donde se encuentran los scripts.
  - **zookeeper-server-start.sh** : inicia zookeeper que es el encargado de la organización del sistema.
  - **kafka-server-start.sh** : inicia kafka que es el encargado de ejecutar las tareas.
  - **kafka-console-consumer.sh** : nos proporciona un consumidor de mensajes kafka, utilizando la línea de comandos. Ideal para realizar pruebas.
  - **kafka-console-producer.sh** : nos proporciona un productor de mensajes kafka, mediante el uso de la línea de comandos. Ideal para realizar pruebas.

[22] https://www.apache.org/dyn/closer.cgi?path=/kafka/0.8.2.0/kafka-0.8.2.0-src.tgz

- **kafka_2.10-0.8.2/config/** : directorio donde se almacenan los ficheros de configuración.
    - **zookeeper.properties** : fichero que contiene la configuración de zookeper.
    - **server.properties** : fichero que contiene la configuración de kafka.

3.3.3 Configuración

En este caso la configuración utilizada será la que incluye el sistema por defecto, es decir:

- **Un único *broker* escuchando en localhost:9092**
- **Zookeeper escuchando en localhost:2181**
- **Una partición**
- **Una réplica**
- **Un *topic* para cada servicio(apache2_log, ssh_log) y un *topic* para cada prueba realizada**

Aunque los *topics* se crean automáticamente al recibir un mensaje de un usuario, perteneciente a un grupo válido, crearemos los *topics* manualmente para evitar posibles problemas.

Situados dentro de la carpeta kafka_2.10-0.8.2.0 debemos ejecutar lo siguiente:

```
>bin/zookeeper-server-start.sh config/zookeeper.properties

 >bin/kafka-server-start.sh config/server.properties
```

Figura 3-3. Puesta en marcha Apache Kafka

```
>bin/Kafka-topics.sh --create --zookeeper localhost:2181 /

--replication-factor 1 --partitions 1 --topic topic_name
```

Figura 3-4. Creación de topics en Apache Kafka

Comprobemos que hemos creado los *topics* correctamente:

```
jmf@jmf:~/pfg/kafka_2.10-0.8.2.0$ bin/kafka-topics.sh --list --zookeeper localhost:2181
apache2_log
correlation_service
ssh_log
jmf@jmf:~/pfg/kafka_2.10-0.8.2.0$
```

Figura 3-5. Comprobación creación de topics

3.3.4 Pruebas

Por último realicemos algunas comprobaciones para demostrar que la instalación y configuración han sido correctas mediante el envío de mensajes a los diferentes *topic* creados.

```
jmf@jmf:~/pfg/kafka_2.10-0.8.2.0$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic correlation_service
[2016-07-28 11:54:15,527] WARN Property topic is not valid (kafka.utils.VerifiableProperties)
Esto es una prueba, para el topic correlation_service
^Cjmf@jmf:~/pfg/kafka_2.10-0.8.2.0$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic apache2_log
[2016-07-28 11:55:34,618] WARN Property topic is not valid (kafka.utils.VerifiableProperties)
Esto es una prueba, para el topic apache2_log
^Cjmf@jmf:~/pfg/kafka_2.10-0.8.2.0$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic ssh_log
[2016-07-28 11:56:25,756] WARN Property topic is not valid (kafka.utils.VerifiableProperties)
Esto es una prueba, para el topic ssh_log
^Cjmf@jmf:~/pfg/kafka_2.10-0.8.2.0$
```

Figura 3-6. Producción de mensajes a los diferentes topics

```
jmf@jmf:~/pfg/kafka_2.10-0.8.2.0$ bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic correlation_service --from-begi
nning
Esto es una prueba, para el topic correlation_service
^CConsumed 1 messages
jmf@jmf:~/pfg/kafka_2.10-0.8.2.0$ bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic apache2_log --from-beginning
Esto es una prueba, para el topic apache2_log
^CConsumed 1 messages
jmf@jmf:~/pfg/kafka_2.10-0.8.2.0$ bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic ssh_log --from-beginning
Esto es una prueba, para el topic ssh_log
^CConsumed 1 messages
jmf@jmf:~/pfg/kafka_2.10-0.8.2.0$
```

Figura 3-7. Mensajes consumidos de los diferentes topics

3.4 Siddhi CEP

Para el desarrollo de esta herramienta, hemos utilizado las siguientes versiones:

- Siddhi CEP 3.1.0
- SiddhiQL 2.0.0

3.4.1 Requisitos del sistema

Requisito	Descripción
RAM	2 GB mínimo
Memoria	512 MB tamaño de la pila de datos. Esto es, generalmente, suficiente para procesar un mensaje tipo de SOAP pero los requisitos varían con el tamaño de los mensajes y el número de mensajes procesados simultáneamente.
Disco	1 GB mínimo, excluyendo el espacio ocupado por los ficheros de logs y las bases de datos.
JVM	1,6,* o 1,7,*

Tabla 3–1. Requisitos Siddhi CEP

3.4.2 Descarga de ficheros

En este caso no hace falta realizar una instalación y configuración como tal. Descargaremos los ficheros para comprobar cómo se desarrolla un proyecto para, más tarde, apoyarnos en las funcionalidades que ofrece siddhi CEP. Este trabajo de investigación nos llevará a crearnos nuestro fichero POM, el cual nos ayudará a realizar nuestra implementación.

Todo lo desarrollado en este apartado será cosecha propia, excepto la utilización de las clases que ofrece siddhi CEP para realizar las operaciones.

3.5 Herramientas utilizadas

3.5.1 Magic Draw

Es una herramienta de modelado software, su dinamismo y versatilidad nos ayudarán en el análisis y diseño de todo el desarrollo relacionado con el motor de correlación y la API Rest.

Soporta UML, el lenguaje de modelado software que utilizaremos, ya que, nos ofrece las capacidades para desarrollo de orientación a objetos; concepto que define el lenguaje de programación que emplearemos, Java.

3.5.1.1 Requisitos de la herramienta

		Mínimo	Recomendado	Alto rendimiento
CPU		Pentium™ 4, 1.6 GHz or higher	Core™ 2 Duo, 2.2 GHz	Core™ i5, 4.0 GHz
RAM		1 GB	3 GB	8 GB
Espacio en disco		500MB ó más dependiendo de los plugins utilizados		
Modo video		800*600 @ 64k colores	1280*1024 @ 64k colores	1280*1024 @ 64k colores
SO		Windows Vista, Mac OS X Lion ó Linux que ejecute el JVM recomendado	Windows 7/8, Mac OS X Mountain Lion ó Linux que ejecute el JVM recomendado	Windows 7/8, Mac OS X Mountain Lion ó Linux que ejecute el JVM recomendado
Java		32 bit	64 bit	64 bit
JAVA_ARGS parámetros	**Si 32 bits Java usado**	JAVA_ARGS=-Xmx800M	JAVA_ARGS=-Xmx1400M	JAVA_ARGS=-Xmx1400M
	Si 64 bits Java usado	JAVA_ARGS=-Xmx800M		

Tabla 3–2. Requisitos MagicDraw

3.5.2 IntelliJ Idea

Es un entorno de desarrollo, ampliamente utilizado en el entorno Big Data debido a que integra herramientas como git, ant, maven, etc.; especializado en el lenguaje de programación Java, aunque ampliable a otros lenguajes mediante módulos.

Lo utilizaremos para desarrollar el código asociado al motor de correlación(siddhi CEP y la API Rest, ambos desarrollados en el lenguaje de programación Java) .

3.5.3 Maven

Es una herramienta de gestión inteligente de proyectos software. Basado en el concepto de modelado de proyectos basado en objetos(POM). Es capaz de construir, generar informes y documentación de un proyecto a partir de una pieza central de información.

Esta pieza de información es un fichero[23] que recoge el esqueleto, las dependencias y las pruebas del proyecto.

Las opciones que proporciona esta herramienta que nos serán útiles son:

- **dependencies** : resolverá las dependencias especificadas en el fichero POM.
- **package** : se encarga de construir el proyecto, compilar y empaquetar. Este comando generará un directorio donde se guardarán los ejecutables generados en formato .jar y las clases asociadas al proyecto.
- **clean package** : elimina el, anteriormente citado, directorio y todo lo relacionado con la construcción del proyecto realizada anteriormente.

3.5.4 Curl

Es una herramienta para transferir datos a/desde un servidor mediante algunos protocolos de aplicación, HTTP en nuestro caso. Usada para interaccionar con el motor de correlación. Si, en un futuro, se amplían las funcionalidades mediante una aplicación web para ofrecer una interfaz más amigable y fácil de usar, el uso de este comando será transparente al usuario, valiendo por tanto el desarrollo realizado.

Algunas opciones de interés que presenta son:

- **-d** : para indicar los datos que queremos enviar.
- **-H** : para incluir en la cabecera de la petición HTTP algunas opciones, como el formato de los datos que se envían.
- **-v** : usado para depuración, muestra el estado de las peticiones y respuestas asociadas a éstas.
- **-X** : para especificar el método HTTP empleado en la petición, ver Tabla 1–5. Códigos HTTP de respuesta.

[23] Normalmente nombrado pom y de formato XML

3.5.5 Git

Git**[10]** es un controlador de versiones, rápido, escalable y distribuido. En nuestro caso lo emplearemos para ir controlando el desarrollo de la herramienta e ir realizando avances progresivos sin modificar el comportamiento previo de la herramienta, ya probado y funcionando correctamente.

Algunas opciones que emplearemos serán:

- **init** : para crear un repositorio
- **add** : para añadir ficheros al seguimiento
- **status** : para comprobar el estado de los ficheros que componen el repositorio
- **commit** : hace efectivos los cambios en el repositorio
- **branch** : para listar, crear y eliminar ramas
- **checkout** : para moverte por las diferentes ramas
- **pull** : para actualizar tu repositorio local al commit mas reciente
- **merge** : para fusionar ramas
- **reset** : para volver a un estado determinado
- **clone** : para copiar un repositorio remoto
- **push** : para enviar los cambios realizados a un repositorio remoto

3.5.6 Nano

Es un editor en línea de comandos, de interfaz amigable y fácil manipulación. Lo utilizaremos para editar los ficheros de configuración.

3.5.7 Logger

Es una interfaz con syslog a través de la línea de comandos. Nos ofrece la posibilidad de redirigir los logs de Apache2 a Rsyslog.

```
CustomLog "|/usr/bin/logger -p local6.info -t access" apache_json
```

Figura 3-8. Uso de logger en el fichero de configuración de Apache2

Explicamos las opciones utilizadas en el comando:

- **-p** : indicamos a donde mandaremos el log y con qué nivel separados por un punto. En nuestro caso usamos una de las fuentes reservadas para uso local(local6) y el nivel de información(*info*).
- **-t** : nos sirve para ponerle una etiqueta a los logs generados. Pondremos la etiqueta *access* porque registraremos los accesos que se producen al servidor web Apache2.

4 DESARROLLO

Commenting your code is like cleaning your bathroom — you never want to do it, but it really does create a more pleasant experience for you and your guests.

- Ryan Campbell-

En este capítulo se muestra el desarrollo del proyecto, es decir pondremos en práctica lo especificado en el diseño, enunciado en el capítulo 2, Modelado de la herramienta. No desarrollaremos la totalidad de código y configuraciones, en el presente capítulo, pero si marcaremos las líneas seguidas para el desarrollo, enunciado completamente en los Anexos.

4.1 Rsyslog

En esta sección, explicaremos las pautas para realizar una configuración que nos ayude a poner en práctica, lo esperado de esta herramienta. Como se comento en el capitulo 2.2.1 Funcionamiento Rsyslog, este sistema funciona en base a un fichero de configuración, en los siguientes apartados se muestran los elementos relevantes y su forma de utilización. El desarrollo completo de la configuración de este sistema se encuentra en el ANEXO B: Configuración Rsyslog.

4.1.1 Templates

Los templates, plantillas a partir de ahora, marcan el esquema que seguirán los datos salientes del sistema. Permitiendo al usuario configurar el formato que más se adecua a sus necesidades.

4.1.1.1 Tipos

Dada la variedad en lo que a formato se refiere, Rsyslog nos ofrece varias posibilades con las que cubrir dicha alternancia.

Tipo	Descripción
list	Se crea una lista de valores. Con esta opción podemos separar los valores con los caracteres que deseemos: crear un mapa clave-valor, formato CSV. . .
subtree	Basado en el concepto CEE, genera una plantilla en forma de árbol, útil para datos con jerarquía o en formato JSON.
string	Similar a list, solo difiere en su forma de uso.
plugin	Permite al usuario realizar un código que se encargue de generar su propio estilo de plantilla.

Tabla 4–1. Tipos de plantilla Rsyslog

4.1.1.2 Uso

Antes de decidir el tipo a utilizar, pensemos cuál de ellas se adecuan más a nuestras necesidades. Apache2 generará logs en formato JSON por lo que parece bastante factible el uso del tipo *subtree*.

El uso de *subtree* es bastante simple, quedando la plantilla para el servicio Apache2 como sigue:

```
template(name="norm_apache" type="list"){
    property(name="$!all-json")
    constant(value="\n")
}
```

Figura 4-1. Plantilla para Apache2

Por otro lado, SSH generará logs en formato RFC5424, por lo que, deberemos construir el JSON manualmente tras la extracción de datos, pareciendo la opción más viable *list*.

El uso de *list* también es bastante simple, pero necesita un trabajo más laborioso y la definición de dos nuevos conceptos:

- **constant** : con esta directiva introduciremos el nombre que identificará al valor dentro del JSON.

  Presenta las siguientes opciones:

  - **value** : donde colocaremos el valor que le queremos dar a la constante.
  - **outname** : nombre del contenido de salida, usado para formatos estructurados. No aplicable en nuestro caso.

- **property** : con esta directiva introduciremos el dato, dentro del JSON. Esta directiva presenta bastante opciones, de las cuales nos interesan:

  - **name** : para indicar el dato que queremos incluir. Rsyslog presenta una serie de datos que obtiene del preprocesamiento y están disponibles para su inclusión. Tras realizar la extracción de campos específicos estos estarán disponibles, mediante el uso de la secuencia $!nombre_atributo.
  - **dateFormat** : con esta opción podremos escoger el formato de fecha. En nuestro caso escogemos el formato rfc3339.
  - **format** : te permite especificar el formato del campo, al estar creando un objeto JSON elegiremos la opción json.

Explicado esto, se muestra la plantilla utilizada para el servicio SSH:

```
template(name="norm_ssh"
         type="list"){
             constant(value="{")
             constant(value="\"message\":\"")                  property(name="msg" format="json")
             constant(value="\",\"raw_message\":\"")           property(name="rawmsg")
             constant(value="\",\"host\":\"")                  property(name="hostname" format="json")
             constant(value="\",\"fromhost\":\"")              property(name="fromhost" format="json")
             constant(value="\",\"fromhost-ip\":\"")           property(name="fromhost-ip" format="json")
             constant(value="\",\"syslogtag\":\"")             property(name="syslogtag" format="json")
             constant(value="\",\"program_name\":\"")          property(name="programname" format="json")
             constant(value="\",\"pri\":\"")                   property(name="pri" format="json")
             constant(value="\",\"pri-text\":\"")              property(name="pri-text" format="json")
             constant(value="\",\"iut\":\"")                   property(name="iut" format="json")

             constant(value="\",\"syslogfacility\":\"")        property(name="syslogfacility" format="json")
             constant(value="\",\"syslogfacility_text\":\"")   property(name="syslogfacility-text" format="json")
             constant(value="\",\"syslogseverity\":\"")        property(name="syslogseverity" format="json")
             constant(value="\",\"syslogseverity_text\":\"")   property(name="syslogseverity-text" format="json")
             constant(value="\",\"syslogpriority\":\"")        property(name="syslogpriority" format="json")
             constant(value="\",\"syslogpriority_text\":\"")   property(name="syslogpriority-text" format="json")

             constant(value="\",\"@timegenerated\":\"")        property(name="timegenerated" dateFormat="rfc3339")
             constant(value="\",\"protocol-version\":\"")      property(name="protocol-version" format="json")
             constant(value="\",\"structured-data\":\"")       property(name="structured-data" format="json")
             constant(value="\",\"app-name\":\"")              property(name="app-name" format="json")
             constant(value="\",\"procid\":\"")                property(name="procid" format="json")
             constant(value="\",\"msgid\":\"")                 property(name="msgid" format="json")
             constant(value="\",\"inputname\":\"")             property(name="inputname" format="json")

             constant(value="\",\"auth\":\"")                  property(name="$!auth" format="json")
             constant(value="\",\"method\":\"")                property(name="$!auth-method" format="json")
             constant(value="\",\"user\":\"")                  property(name="$!username" format="json")
             constant(value="\",\"ip\":\"")                    property(name="$!src-ip" format="json")
             constant(value="\",\"port\":\"")                  property(name="$!src-port" format="json")
             constant(value="\",\"protocol\":\"")              property(name="$!protocol" format="json")
             constant(value="\"}\\n")
         }
```

Figura 4-2. Plantilla para SSH

4.1.2 Módulos

A continuación explicaremos el modo de utilización de los diferentes módulos empleados.

4.1.2.1 mmnormalize

Para usar este módulo, que se encarga de obtener los datos del campo MSG mediante reglas, ejecutaremos la directiva *action* con los siguientes parámetros:

- **type=”mmnormalize”.** Con esto indicamos el módulo que llevará acabo la acción.
- **ruleBase=”/etc/rsyslog.d/ssh.rules”**. Deberemos indicar la ruta[24] hacia el fichero que contiene las reglas de normalización que queremos aplicar.

```
*.* action (type = “mmnormalize” ruleBase= “/etc/rsyslog.d/ssh.rules”)
```

Figura 4-3. Modo utilización del módulo mmnormalize

[24] Se aconseja que sea la ruta absoluta, aunque no se prohíbe el uso de rutas relativas.

4.1.2.2 mmjsonparse

Para obtener el objeto JSON contenido en el log, deberemos ejecutar dicho modulo con los siguientes parámetros en la directiva. También especificaremos la fuente y el nivel de log(local6.info)

- **type="mmjsonparse"**. Indicamos el módulo encargado de ejecutar la acción.

```
local6.info action (type = "mmjsonparse")
```

Figura 4-4. Modo utilización del módulo mmjsonparse

4.1.2.3 omfile

Para ir guardando los logs en un fichero del sistema local, deberemos ejecutar la directiva *action* con los siguientes parámetros:

- **type="omfile"**. Con esto indicamos el modulo que llevará acabo la acción.
- **file="/var/log/norm_ssh.log"**[25]. Indicamos la ruta hacia el fichero, donde se guardarán los logs.
- **template="norm_ssh"**[26]. La plantilla que debemos aplicar para conseguir el formato deseado, ver Uso.

```
action (type = "omfile" file="/var/log/norm_ssh.log"
template="norm_ssh")
```

Figura 4-5. Modo utilización del módulo omfile

4.1.2.4 omkafka

Para enviar los logs al sistema Apache Kafka, deberemos ejecutar la directiva *action* con los siguientes parámetros:

- **type="omkafka"** . Con esto indicamos el modulo que llevará acabo la acción.
- **topic="ssh_log"**[27] . Con esto indicamos el *topic* donde queremos que se publiquen los logs.
- **broker="localhost"** . La dirección del *broker* que manejar el cluster de Kafka.
- **confParam=["compression.codec=snappy","socket.timeout.ms=10", "socket.keepalive.enable=true"]**. Esto es un array que contiene algunos parámetros para el productor encargado de enviar los logs a Apache Kafka, en nuestro caso indicamos el tipo de compresión y la configuración del socket.
- **template="norm_ssh"**[28] . La plantilla que debemos aplicar para conseguir el formato deseado, ver Uso.

[25] "/var/log/norm_apache.log" Para el caso de Apache2.
[26] "norm_apache". Para el caso de Apache2.
[27] "apache2_log".Para el caso de Apache2.
[28] "norm_apache". Para el caso de Apache2.

```
action (type = "omkafka" topic="ssh_log" broker="localhost"
confParam=["compression.codec=snappy","socket.timeout.ms=10",
"socket.keepalive.enable=true"] template="norm_ssh"  )
```

Figura 4-6. Modo utilización del módulo omkafka

4.2 Motor de correlación

En esta sección mostraremos los aspectos más destacados del motor de correlación, que incluye a Siddhi CEP y la API Rest, su desarrollo completo puede encontrarse en el ANEXO C: Motor Correlación. Dado que lo que acontece a esta sección es desarrollo de código, haremos hincapié en los aspectos fundamentales sin entrar en más detalle.

4.2.1 Estructura del motor

Tres secciones importantes conforman el motor: siddhi CEP, Apache Kafka y la API Rest. El proyecto se divide en los siguientes directorios y ficheros:

- **PFG-master/pom.xml** : este fichero contiene las dependencias y plugins necesarios para que todos los componentes funcionen correctamente. Incluye, también, una serie de opciones para ayudar a la herramienta maven a compilar y empaquetar los ejecutables obtenidos.
- **PFG-master/config/configFileCS.yml** : si no se indica un fichero de configuración al ejecutar el servicio, este fichero ofrece una configuración por defecto.
- **PFG-master/target/** : en este directorio se guardarán los resultados de la compilación y empaquetamiento.
    - **classes/** : directorio donde se encuentran las clases generadas.
    - **maven-archiver/** : directorio donde se encuentran las opciones para compilación registradas en el fichero pom.xml
    - **maven-status/** : en este directorio podremos encontrar lo relativo a la compilación.
    - **PFG_CorrelationService-1.0-SNAPSHOT.jar** : ejecutable que contiene el proyecto, resuelto sin dependencias.
    - **PFG_CorrelationService-1.0-SNAPSHOT-selfcontained.jar** : ejecutable que contiene el proyecto completo, solo haciendo falta para ejecutar una JVM válida y la herramienta java.
- **PFG-master/src/main/** : directorio donde se encuentra el código desarrollado.
    - **resources/log4j.xml** : este fichero contiene la configuración para la generación de logs del proyecto.
    - **java/pfg/Util/ConfigData.java** : se encarga de leer la configuración del servicio a partir de un fichero .yml.
    - **java/pfg/CorrelationService.java** : se encarga de iniciar los diferentes sistemas que conforman la herramienta con la configuración leída.
    - **java/pfg/Kafka** : directorio donde se encuentra lo relativo al sistema Apache Kafka

- **KafkaConsumerManager.java** : se encarga de configurar y poner en marcha el consumidor.
- **KafkaConsumer.java** : se encarga de generar la entrada a siddhi CEP
- **KafkaProducer.java** : se encarga de producir la salida de siddhi CEP

o **java/pfg/Rest** : directorio donde se encuentra lo relativo a la API Rest

- **RestManager.java** : se encarga de configurar y poner en marcha la API Rest
- **Resource.java** : se encarga de manejar las peticiones HTTP entrantes
- **RestListener.java** : interfaz con siddhi CEP, para comunicar los cambios recibidos mediante la API Rest

o **java/pfg/Siddhi** : directorio donde se encuentra lo relativo al sistema siddhi CEP

- **SiddhiHandler.java** : se encarga de manejar siddhi CEP y las interacciones de la API Rest con éste.
- **Siddhi.java** : realiza las operaciones propuestas por el usuario.
- **StartChecking.java** : se encarga de ir comprobando las peticiones recibidas para iniciar/reiniciar el motor.

4.2.2 Curl: Guía de uso

Esta herramienta nos ayudará a interactuar con el motor de correlación mediante el uso del protocolo HTTP. Es por ello que se presenta una guía que nos ayude a desarrollar las interacciones presentes en el diseño del Diagrama de casos de uso.

- Para incluir un plan de ejecución . Deberemos añadir en la petición además del plan, el nombre que recibe el *stream*, que deber ser único, todo ello en formato JSON.

```
curl -H "Content-Type: application/json" -X POST -d
'{"name_stream":"stream_name","stream":" definición del stream en
format SiddhiQL"}' http://server_URL/action=start -v
```

Figura 4-7. Petición de inclusión plan de ejecución en el motor

- Para reiniciar un plan. Basta con ejecutar el comando anterior sin incluir ningún plan. Interpretando así el motor, que no se quiere cambiar el plan definido con anterioridad.

```
curl -X POST  http://server_URL/action=start -v
```

Figura 4-8. Petición del reinicio del plan de ejecución existente

- Para añadir una *query*. Deberemos incluir en la petición además de la definición de la *query* un identificador y el nombre del *stream* de salida asociado, ambos deben ser únicos.

```
curl -H "Content-Type: application/json" -X POST -d '{"id":unique_id,
"query":"Definición de la query en format SiddhiSQL",
"OutStream":"out_stream_name"}' http://server_URL/action=add -v
```

Figura 4-9. Petición de inclusión de una nueva query

- Para eliminar una *query*. Como al crear las *queries* indicamos un identificador único, podremos utilizar dicho parámetro para la eliminación de una *query*.

```
curl -X DELETE  http://server_URL/action=delete/query/query_id -v
```

Figura 4-10. Petición de eliminación de una query

- Para listar las *queries* presentes en el motor. Por último, una opción para comprobar las *queries* definidas hasta el momento.

```
curl -X GET http://server_URL/CS/res/queries -v
```

Figura 4-11. Petición del listado de queries presentes en el motor

5 RESULTADOS

A person who never made a mistake never tried anything new.

- Albert Einstein -

En este capítulo se muestra el desarrollo completo mediante una serie de pruebas, que proyectan el potencial que proporciona siddhi CEP, la abstracción al usuario que ofrece la API Rest, la robustez a la hora de tratar logs de Rsyslog y el hecho de por qué Apache Kafka, es un sistema más que contrastado en el ámbito del Big Data.

5.1 Pruebas con el servicio SSH

El servicio SSH nos proporciona un método de conexión a máquinas remotas, poniéndolo en el punto de mira al empezar a buscar vulnerabilidades en un sistema. Si se consigue vulnerar dicho servicio y se consigue entrar, el intruso tendría total libertad, además sin levantar sospecha, al ser un usuario válido el que realiza los cambios en la máquina. Veamos a continuación una serie de operaciones que muestran cómo nos puede ayudar el sistema desarrollado en esta memoria.

5.1.1 Prueba1

Definamos, primeramente, el entorno:

- Rsyslog generará los logs para SSH en el *topic* ssh_log
- El motor a su vez escribirá los resultados en el *topic* prueba1
- El *stream* asociado a SSH, en Siddhi CEP, lo llamaremos sshStream, que tendrá la siguiente definición:

```
@config(async = 'true')define stream sshStream (
message string, raw_message string, host string,
fromhost string, fromhost_ip string, syslogtag string,
program_name string, pri string, pri_text string,
iut string, syslogfacility string,
syslogfacility_text string, syslogseverity string,
syslogseverity_text string, syslogpriority string,
syslogpriority_text string, timegenerated string,
protocol_version string, structured_data string, app_name string,
procid string, msgid string, inputname string, status string,
method string, auth_method string, user string, ip_src string,
port string, protocol string);
```

Figura 5-1. Definición sshStream

- Definiremos una *query* que obtenga las sesiones SSH para el usuario root, denominaremos a ésta query1, que a su vez tendrá asociado un stream de salida llamado query1Stream. En esta *query* aplicaremos un filtrado por usuario y seleccionaremos como atributos de salida: la ip que intenta acceder, a qué usuario, a qué maquina se quiere conectar y el método de autenticación.

  Su definición es la siguiente:

```
@info(name = 'query1') from sshStream[user == 'root'] select  ip_src,
user, host, method, auth_method insert into query1Stream;
```

Figura 5-2. Definición query1 para SSH

- Definiremos una *query* que obtenga las sesiones SSH establecidas, denominaremos a ésta query2, que a su vez tendrá asociado un *stream* de salida llamado query2Stream. En esta ocasión aplicaremos una ventana temporal de dos minutos, que nos ofrezca las sesiones SSH establecidas con éxito, siendo los atributos de salida: el número de conexiones hasta el momento, la ip de la máquina que ha iniciado sesión y en qué máquina lo hizo.

  Su definición es la siguiente:

```
@info(name = 'query2') from sshStream[status
=='Accepted']#window.time(2 min) select count(ip_src) as
sessions_number, ip_src, host  insert into query2Stream;
```

Figura 5-3. Definición query2 para SSH

- Definiremos una *query* que sea capaz de detectar un posible caso de intrusión para el usuario root, denominaremos a ésta query3, que a su vez tendrá asociado un *stream* de salida llamado query3Stream. En este caso aplicaremos un patrón que compruebe las sesiones fallidas para el usuario root, para ello aplicaremos un filtrado de usuario y estado de la petición de inicio de sesión, también tomaremos que esto será una intrusión si se perpetra más de 6 veces; los atributos de salida serán las ips de las máquinas que han intentado acceder y un mensaje de aviso.

  Su definición es la siguiente:

```
@info(name = 'query3') from a1=sshStream[user == 'root' and status ==
'Failed' ]<6:> select a1[0].ip_src as ip_1, a1[1].ip_src as ip_2,
user,'Intento de conexion remota sospechoso' as description  insert
into query3Stream;
```

Figura 5-4. Definición query3 para SSH

5.1.1.1 Desarrollo prueba1

Además de mostrar el resultado de las operaciones mostradas anteriormente, indicaremos como comunicarnos con siddhi CEP mediante la API Rest, para lo que haremos uso del comando curl, ver Curl: Guía de uso.

En primer lugar debemos iniciar Apache Kafka, ver Configuración. Tras esto pondremos en marcha el servicio ejecutando:

```
>java –jar PFG_CorrelationService-1.0-SNAPSHOT-selfcontained.jar
config/configFileCS.yml
```

Figura 5-5. Ejecución del servicio desarrollado

Siendo el contenido del fichero de configuración para esta prueba:

```
#Fichero de configuración del servicio de correlación

#Zookeeper

zookeeperURL: localhost:2181

zookeeperGroup: log_management

zookeeperSessionTimeout: 400

zookeeperSyncTime: 200

zookeeperCommitInterval: 1000

zookeeperOffsetReset:smallest


#Kafka

kafkaBrokers: localhost:9092

kafkaSerializer: kafka.serializer.StringEncoder

kafkaACK: 1


#Kafka Consumer

consumerTopic: ssh_log


#Kafka Producer

producerTopic: prueba1

#REST

RestURI: http://localhost:8888/CS/
```

Figura 5-6. Contenido configFileCS.yml para SSH

Una vez puesto en marcha todo lo necesario, para mostrar todas las funcionalidades seguiremos la siguiente cronología:

1. Añadiremos la query1 y query2

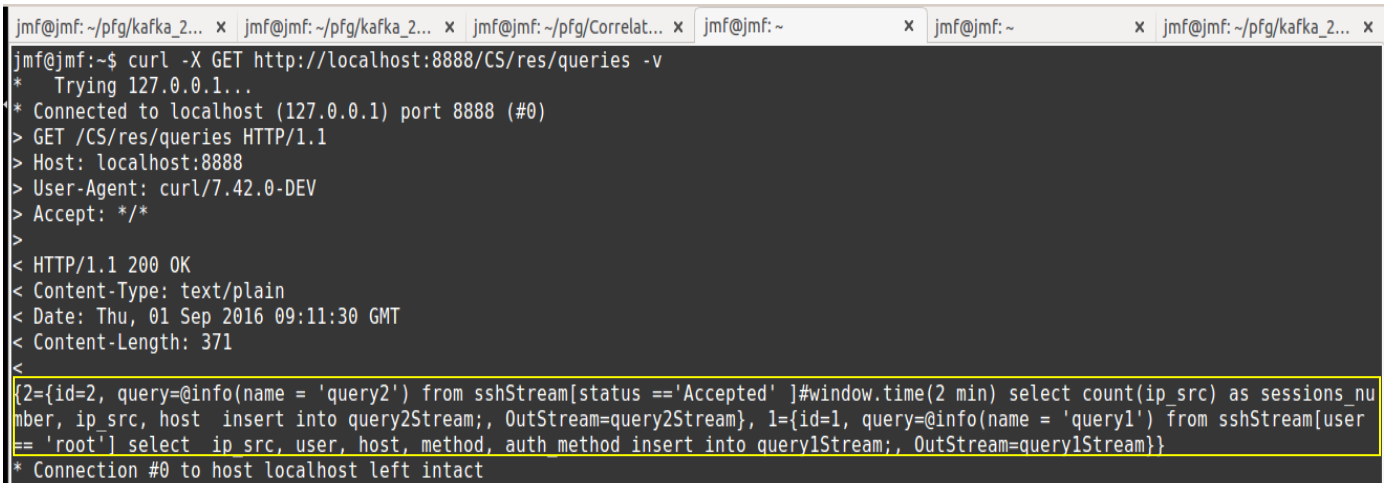

Figura 5-7. Comprobación inclusión queries 1 y 2

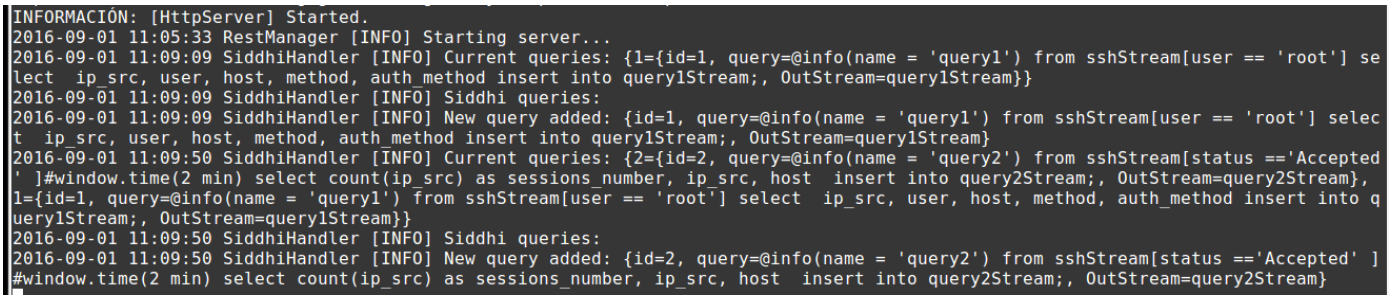

Figura 5-8. Registro de las inserciones de queries en el sistema

2. Iniciamos el plan de ejecución

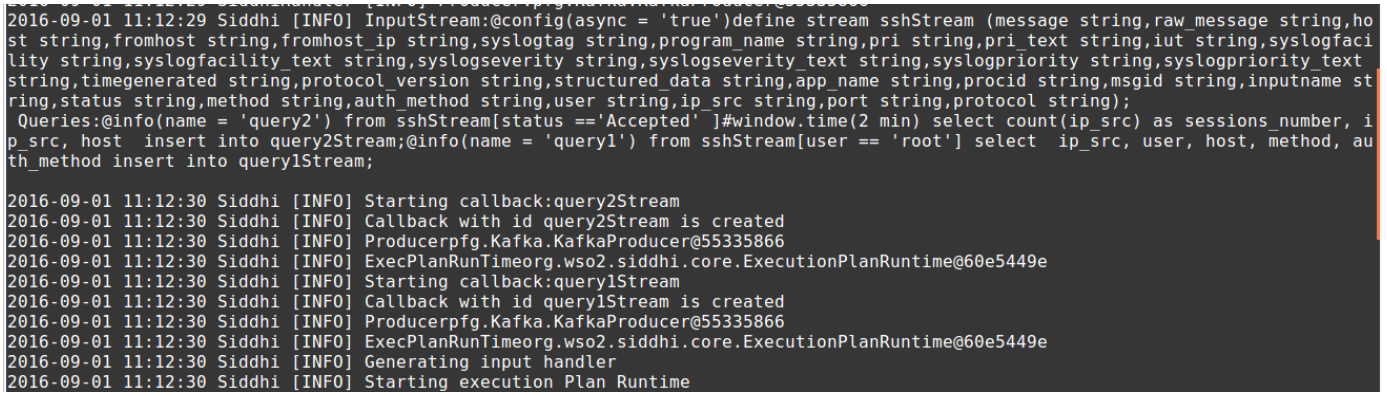

Figura 5-9. Registro del inicio del plan de ejecución en el sistema

3. Recopilamos resultados de query1 y query2.

   Se comprueba que la maquina que recibe la petición es jmf para el usuario root, que se ha utilizado contraseña como método de autenticación y que la ip del cliente es la 127.0.0.1.

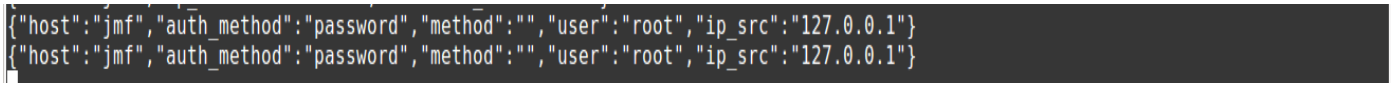

Figura 5-10. Resultados query1 de SSH

En este caso vemos como se incrementan el numero de sesiones, en la máquina donde se inician dichas sesiones y la ip que está conectada.

Figura 5-11. Resultados query2 de SSH

4. Incluimos query3 y eliminamos query1 y query2

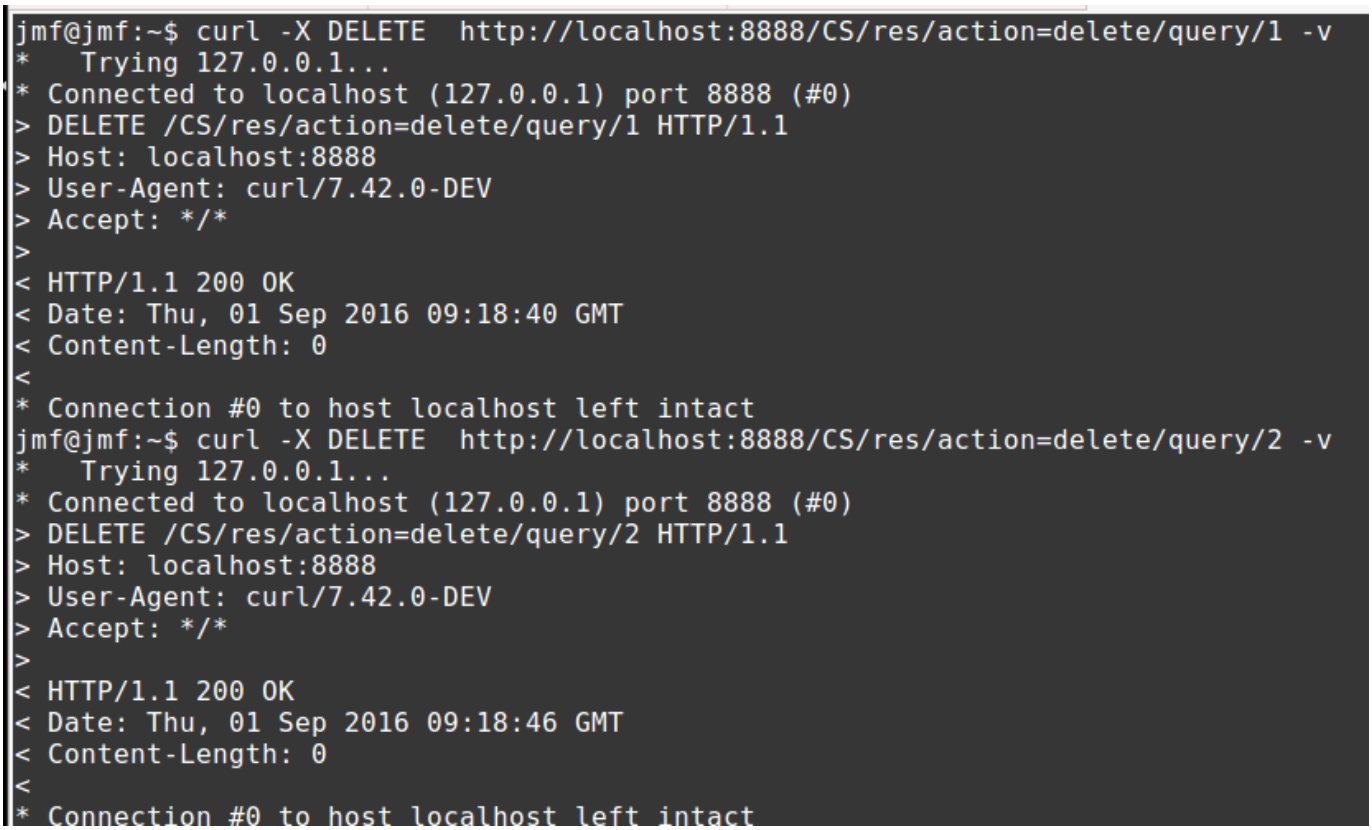

Figura 5-12. Eliminación query1 y query2

5. Listamos las queries para comprobar que, efectivamente, sólo queda query3

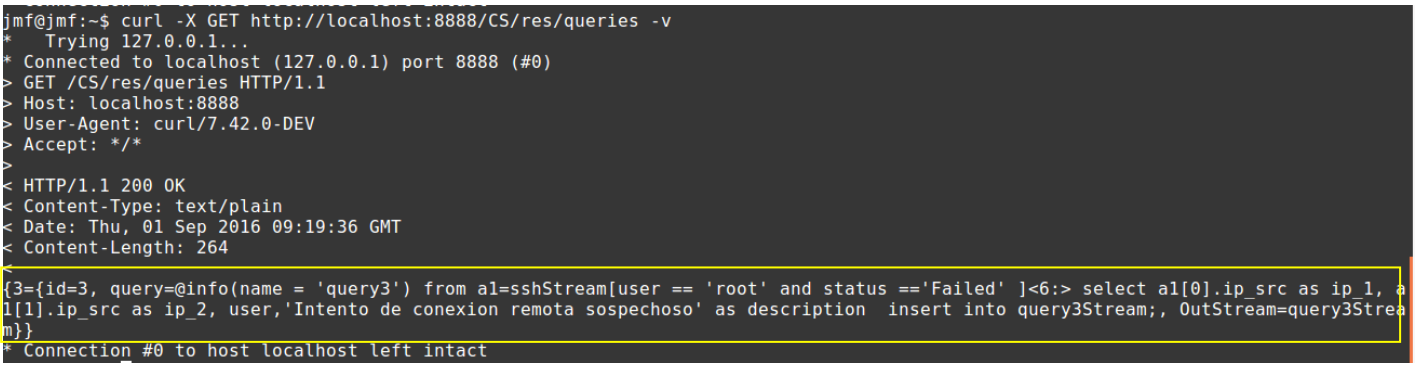

Figura 5-13. Listado queries tras eliminación

6. Reiniciamos el plan para que query3 tenga efecto. Como es un reinicio no hace falta enviar de nuevo la definición de sshStream:

```
jmf@jmf:~$ curl -X POST  http://localhost:8888/CS/res/action=start -v
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 8888 (#0)
> POST /CS/res/action=start HTTP/1.1
> Host: localhost:8888
> User-Agent: curl/7.42.0-DEV
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Date: Thu, 01 Sep 2016 09:20:31 GMT
< Content-Length: 0
<
* Connection #0 to host localhost left intact
```

Figura 5-14. Reinicio usando API Rest

```
2016-09-01 11:18:40 SiddhiHandler [INFO] Query with the id 1 has been removed
2016-09-01 11:18:40 SiddhiHandler [INFO] Current queries: {3={id=3, query=@info(name = 'query3') from a1=sshStream[user == 'root'
and status =='Failed' ]<6:> select a1[0].ip_src as ip_1, a1[1].ip_src as ip_2, user,'Intento de conexion remota sospechoso' as des
cription  insert into query3Stream;, OutStream=query3Stream}, 2={id=2, query=@info(name = 'query2') from sshStream[status =='Accep
ted' ]#window.time(2 min) select count(ip_src) as sessions_number, ip_src, host  insert into query2Stream;, OutStream=query2Stream
}}
2016-09-01 11:18:46 SiddhiHandler [INFO] Query with the id 2 has been removed
2016-09-01 11:18:46 SiddhiHandler [INFO] Current queries: {3={id=3, query=@info(name = 'query3') from a1=sshStream[user == 'root'
and status =='Failed' ]<6:> select a1[0].ip_src as ip_1, a1[1].ip_src as ip_2, user,'Intento de conexion remota sospechoso' as des
cription  insert into query3Stream;, OutStream=query3Stream}}
2016-09-01 11:20:31 SiddhiHandler [INFO] Restart is upload
2016-09-01 11:20:31 StartChecking [INFO] Restart request was received
2016-09-01 11:20:31 SiddhiHandler [INFO] Received Restart request
2016-09-01 11:20:31 SiddhiHandler [INFO] Upgrade is starting...
2016-09-01 11:20:31 SiddhiHandler [INFO] Updating queries. . .
2016-09-01 11:20:31 SiddhiHandler [INFO] Consumer:pfg.Kafka.KafkaConsumerManager@61b1a2bd
2016-09-01 11:20:31 SiddhiHandler [INFO] Producer:pfg.Kafka.KafkaProducer@55335866
2016-09-01 11:20:31 Siddhi [INFO] InputStream:@config(async = 'true')define stream sshStream (message string,raw_message string,ho
st string,fromhost string,fromhost_ip string,syslogtag string,program_name string,pri string,pri_text string,iut string,syslogfaci
lity string,syslogfacility_text string,syslogseverity string,syslogseverity_text string,syslogpriority string,syslogpriority_text
string,timegenerated string,protocol_version string,structured_data string,app_name string,procid string,msgid string,inputname st
ring,status string,method string,auth_method string,user string,ip_src string,port string,protocol string);
 Queries:@info(name = 'query3') from a1=sshStream[user == 'root' and status =='Failed' ]<6:> select a1[0].ip_src as ip_1, a1[1].ip
_src as ip_2, user,'Intento de conexion remota sospechoso' as description  insert into query3Stream;

2016-09-01 11:20:31 Siddhi [INFO] Starting callback:query3Stream
2016-09-01 11:20:31 Siddhi [INFO] Callback with id query3Stream is created
2016-09-01 11:20:31 Siddhi [INFO] Producerpfg.Kafka.KafkaProducer@55335866
2016-09-01 11:20:31 Siddhi [INFO] ExecPlanRunTimeorg.wso2.siddhi.core.ExecutionPlanRuntime@58f0d593
2016-09-01 11:20:31 Siddhi [INFO] Generating input handler
2016-09-01 11:20:31 Siddhi [INFO] Starting execution Plan Runtime
2016-09-01 11:20:31 SiddhiHandler [INFO] Consumer:pfg.Kafka.KafkaConsumerManager@61b1a2bd
2016-09-01 11:20:31 SiddhiHandler [INFO] Producer:pfg.Kafka.KafkaProducer@55335866
2016-09-01 11:20:31 SiddhiHandler [INFO] Siddhi:pfg.Siddhi.Siddhi@42ed0e35
```

Figura 5-15. Registro del reinicio en el sistema

7. Recopilamos resultados de query3

   Al producirse más de 6 intentos de conexión fallida para el usuario root, se indica un mensaje de aviso y las ips que han intentado dichas conexiones(por simplicidad solo se añaden las 2 primeras, las restantes, en este caso, serían la misma ip)

```
{"description":"Intento de conexion remota sospechoso","ip_1":"127.0.0.1","user":"root","ip_2":"127.0.0.1"}
```

Figura 5-16. Resultados query3 de SSH

5.2 Pruebas con el servicio Apache2

Apache2 es un servidor web ampliamente utilizado debido a su naturaleza de proyecto de código abierto, a que su línea de aprendizaje es bastante liviana y ofrece unas prestaciones muy buenas gracias a su diseño.

Es importante que una vez puesto en marcha sepamos ir adecuando su comportamiento en base al servicio que ofrezcamos y al tipo de clientes que lo usan. Haremos un diseño de varias operaciones que nos muestren dicho comportamiento.

5.2.1 Prueba2

Procederemos de forma equivalente que en la prueba desarrollada en la sección anterior, Prueba1. Definamos el entorno:

- Rsyslog generará los logs para Apache2 en el *topic* apache2_log
- El motor a su vez escribirá los resultados en el *topic* prueba2
- El *stream* asociado a Apache2, en Siddhi CEP, lo llamaremos apacheStream, que tendrá la siguiente definición:

```
@config(async = 'true')define stream apacheStream ( time string,
message string, ip_src string, duration int, status int, request
string, urlpath string, urlquery string, bytes int, method string,
referer string, useragent string);
```

Figura 5-17. Definición apacheStream

- Definiremos una query que obtenga el número de bytes que solicita cada usuario, denominaremos a ésta query1, que a su vez tendrá asociado un stream de salida llamado query1Stream. Para ello emplearemos una ventana de tipo *timeBatch*, que se encargará de recolectar los eventos cada 2 minutos y obtendrá como salida el consumo de bytes por cada ip, ordenados por la ip del cliente.

  Su definición es la siguiente:

```
@info(name = 'query1') from apacheStream#window.timeBatch(2 min)
select ip_src, sum(bytes) as consumedBytes group by ip_src insert into
query1Stream;
```

Figura 5-18. Definición query1 para Apache2

- Definiremos una query que obtenga la información de cada usuario del sistema(contenida en el campo useragent utilizada en el protocolo HTTP), denominaremos a ésta query2, que a su vez tendrá asociado un stream de salida llamado query2Stream. En este caso realizaremos una proyección sobre los atributos ip_src y useragent.

  Su definición es la siguiente:

```
@info(name = \u0027query2\u0027) from apacheStream select ip_src,
useragent insert into query2Stream;
```

Figura 5-19. Definición query2 para Apache2

- Definiremos una query que obtenga las peticiones que no han sido resueltas de manera satisfactoria, denominaremos a ésta query3, que a su vez tendrá asociado un stream de salida llamado query3Stream. En esta ocasión aplicaremos un filtrado para obtener las peticiones que no son correctamente tratadas(código de respuesta distinto de 200), aplicando una ventana de 10 eventos y seleccionando como atributos de salida: el código que se ha generado, el método empleado en la petición, a qué recurso se intenta acceder y la ip del cliente, todo ello ordenado por los códigos de error.

  Su definición es la siguiente:

```
@info(name = 'query3') from apacheStream[status != 
200]#window.length(10) select status, method, urlpath, ip_src group by 
status insert into query3Stream;
```

Figura 5-20. Definición query3 para Apache2

5.2.1.1 Desarrollo prueba2

Suponiendo que el sistema está en marcha, no habría que realizar nada; sino lo estuviera operaríamos como en Desarrollo prueba1, sólo deberemos de cambiar el fichero de configuración. Quedando para esta prueba de la siguiente manera:

```
#Fichero de configuración del servicio de correlación

#Zookeeper

zookeeperURL: localhost:2181
zookeeperGroup: log_management
zookeeperSessionTimeout: 400
zookeeperSyncTime: 200
zookeeperCommitInterval: 1000
zookeeperOffsetReset:smallest


#Kafka

kafkaBrokers: localhost:9092
kafkaSerializer: kafka.serializer.StringEncoder
kafkaACK: 1


#Kafka Consumer

consumerTopic: apache2_log



#Kafka Producer
producerTopic: prueba2


#REST

RestURI: http://localhost:8888/CS/
```

Figura 5-21. Contenido configFileCS.yml para Apache2

Para no repetir lo mencionado en la sección anterior Prueba1, mostraremos únicamente los resultados producidos, una vez introducido el comportamiento del sistema y su correcto funcionamiento.

1. Tras incluir las queries e iniciar el plan de ejecución se obtienen los siguientes resultados:

Para la primera query debemos observar cómo se van acumulando el número de bytes solicitados por cada cliente.


```
{"consumedBytes":"350","ip_src":"192.168.1.33"}
{"consumedBytes":"3611","ip_src":"192.168.1.33"}
{"consumedBytes":"3611","ip_src":"192.168.1.33"}
{"consumedBytes":"3961","ip_src":"192.168.1.33"}
{"consumedBytes":"3961","ip_src":"192.168.1.33"}
{"consumedBytes":"3961","ip_src":"192.168.1.33"}
{"consumedBytes":"4311","ip_src":"192.168.1.33"}
{"consumedBytes":"4661","ip_src":"192.168.1.33"}
{"consumedBytes":"5011","ip_src":"192.168.1.33"}
{"consumedBytes":"5361","ip_src":"192.168.1.33"}
{"consumedBytes":"350","ip_src":"127.0.0.1"}
{"consumedBytes":"350","ip_src":"127.0.0.1"}
{"consumedBytes":"562","ip_src":"127.0.0.1"}
```


Figura 5-22. Resultados query1 de Apache2

En la segunda query debemos apreciar la información que cada cliente envía en el campo useragent, pudiendo apreciar el sistema operativo y el navegador utilizados, entre otras cosas.


```
{"useragent":"Mozilla/5.0 (Windows NT 6.1; rv:35.0) Gecko/20100101 Firefox/35.0","ip_src":"192.168.1.34"}
{"useragent":"Mozilla/5.0 (Windows NT 6.1; rv:35.0) Gecko/20100101 Firefox/35.0","ip_src":"192.168.1.34"}
{"useragent":"Mozilla/5.0 (Windows NT 6.1; rv:35.0) Gecko/20100101 Firefox/35.0","ip_src":"192.168.1.34"}

{"useragent":"Mozilla/5.0 (Windows NT 6.1; rv:35.0) Gecko/20100101 Firefox/35.0","ip_src":"192.168.1.34"}

{"useragent":"Mozilla/5.0 (Windows NT 6.1; rv:35.0) Gecko/20100101 Firefox/35.0","ip_src":"192.168.1.34"}
{"useragent":"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36","ip_src":"19
2.168.1.34"}

{"useragent":"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36","ip_src":"19
2.168.1.34"}
{"useragent":"Mozilla/5.0 (Linux; Android 6.0; ALE-L21 Build/HuaweiALE-L21) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.274
3.98 Mobile Safari/537.36","ip_src":"192.168.1.33"}
```


Figura 5-23. Resultados query2 de Apache2

Por último, en los resultados asociados a la query3 obtendremos las peticiones que no han sido tratadas con éxito para esclarecer el motivo de dicho hecho. En el caso que se presenta, es el acceso a una URL que no está contemplada para servirse en el servidor.


```
jmf@jmf:~/pfg/kafka_2.10-0.8.2.0$ bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic prueba2 --from-beginning
{"urlpath":"/html/estepathgenera404","status":"404","method":"GET","ip_src":"127.0.0.1"}
{"urlpath":"/html/estepathesinexistente","status":"404","method":"GET","ip_src":"127.0.0.1"}
{"urlpath":"/html/codigoerror","status":"404","method":"GET","ip_src":"127.0.0.1"}
{"urlpath":"/favicon.ico","status":"404","method":"GET","ip_src":"192.168.1.33"}
{"urlpath":"/html/patherroneo","status":"404","method":"GET","ip_src":"192.168.1.33"}
```


Figura 5-24. Resultados query3 de Apache2

6 CONCLUSIONES

Tell me and I forget, teach me and I may remember, involve me and I learn.

- Benjamin Franklin -

Para concluir esta memoria se proporciona como se ha distribuido el tiempo, una aproximación al coste que podría tener su realización en el mundo real, una serie de conclusiones que aportan una línea a seguir y una opinión personal sobre lo que me ha aportado el desarrollo de este proyecto.

6.1 Distribución Temporal

Esta sección muestra como se ha distribuido el tiempo en la realización del proyecto y los posteriores arreglos introducidos.

6.1.1 Estimada

A continuación se muestra una tabla, que recoge una estimación del tiempo que se preveía emplear en el desarrollo de este proyecto.

Tarea[29]	Tiempo estimado(semanas)
Elección de herramientas	1
Aprendizaje Rsyslog	1
Aprendizaje Apache Kafka	1
Aprendizaje Siddhi CEP	2
Desarrollo Rsyslog	2
Desarrollo Apache Kafka	1
Desarrollo Siddhi CEP	2
Pruebas	2
Realización memoria	2
	Total: 14

Tabla 6–1. Distribución temporal estimada

[29] Cada tarea lleva asociada una parte de documentación que será recopilada en la realización de la memoria.

Como se puede observar si tenemos en cuenta la duración del proyecto, un cuatrimestre(16 semanas si contamos semana santa y feria como lectivos), se han dejado 2 semanas en previsión por posibles contingencias o ampliaciones de funcionalidades.

6.1.2 Real

Durante la realización del proyecto surgen algunos inconvenientes que retrasan algunas tareas o se añaden algunas funcionalidades que no se habían recogido al inicio. Quedando por tanto, alterada la distribución de la siguiente manera:

Tarea[30]	Tiempo empleado(semanas)[31]
Elección de herramientas	1
Aprendizaje Rsyslog	1
Aprendizaje Apache Kafka	1
Aprendizaje Siddhi CEP	2
Desarrollo Rsyslog	3(*)
Desarrollo Apache Kafka	0.5(*)
Desarrollo Siddhi CEP	2.5(*)
Desarrollo API REST	1(*)
Pruebas	2
Realización memoria	2
	Total: 16

Tabla 6–2. Distribución temporal real

Finalmente las dos semanas previstas se usaron por diversos motivos:

- Ampliación del proyecto con una nueva tarea, desarrollo API REST
- Aumento del tiempo dedicado, respecto al estimado, a varias de las tareas(Desarrollo Rsyslog y desarrollo Siddhi CEP).
- También se redujo el tiempo empleado en la tarea desarrollo Apache Kafka.

Completando así, la totalidad del tiempo dispuesto para el desarrollo del proyecto.

[30] Cada tarea lleva asociada una parte de documentación que será recopilada en la realización de la memoria.
[31] Se marcan con asteriscos las tareas que han sido modificadas respecto a la previsión original.

6.2 Presupuesto

En esta sección traduciremos la distribución temporal anterior al gasto que hubiera supuesto, de haber sido realizado en un entorno real, teniendo en cuenta gastos personales y en recursos.

PRESUPUESTO

RECURSOS HUMANOS

Tarea	Tiempo empleado(horas)	Precio(€/hora)[32]	Desglose por tarea(€)
Decisión herramientas	10	10,5	105
Aprendizaje Rsyslog	20	10,5	210
Desarrollo Rsyslog	30	10,5	315
Aprendizaje Apache Kafka	10	10,5	105
Desarrollo Apache Kafka	5	10,5	52,5
Aprendizaje Siddhi CEP	30	10,5	315
Desarrollo Siddhi CEP	40	10,5	420
Desarrollo API REST	15	10,5	157,5
Pruebas	25	10,5	262,5
Memoria	35	10,5	367,5
		Total(€)	**2310**

RECURSOS MATERIALES

Concepto	Precio(€)
Portátil ASUS	600
Cisco UCS C260 M2	6000
Total(€)	**6600**

Total presupuesto(€)	**8910**

Tabla 6–3. Presupuesto

6.3 Conclusiones

Para concluir esta memoria de lo que ha sido mi trabajo fin de grado, proyecto una línea futura de mejora de la herramienta y concluyo con una reflexión personal de lo que ha supuesto el desarrollo de este proyecto.

6.3.1 Propuestas de mejora

Dada la envergadura temporal del proyecto y los medios al alcance de un estudiante se han quedado cosas en el tintero, como es normal. Por lo cual, presento una línea de mejora con la que continuar el desarrollo de la herramienta. Se ha desarrollado un *back-end*, del cual se presentan a continuación las posibles mejoras, quedando abierta por tanto la realización de un *front-end* que muestre los resultados de forma amigable para el usuario(dashboards, gráficos, tablas…).

[32] Aplicando salario base+plus de un titulado superior del convenio TIC[11]

6.3.1.1 Rsyslog

En este apartado solo hemos desarrollado el colector, por tanto en un futuro:

- Configurar los sensores.
- Conseguir que las transacciones entre sensor-colector sean seguras.
- Aumentar el número de servicios soportados.

6.3.1.2 Kafka

También por sencillez, en este apartado solo hemos consumido de un *topic* y utilizando solo una máquina. Para mejorar el rendimiento, se presentan las siguientes opciones:

- Aumentar el número de máquinas que conforman el clúster.
- Configurar adecuadamente ese aumento.
- Realizar una configuración en anillo de los consumidores que generan los *stream* de entrada del motor, posibilitando consumir de varios *topics* a la vez.
- Que cada consumidor sea capaz de obtener el esquema del JSON de forma automática, evitando así que el usuario tenga que conocer el esquema y que tenga que realizar a "mano" la definición del *stream*.

6.3.1.3 Siddhi Cep

Del mismo modo, se ha realizado un diseño y desarrollo simple de lo que puede aportar este motor de correlación. Quedando por tanto, para una línea futura las siguientes tareas:

- Hacerlo distribuido, intentar integrarlo en algún sistema Big Data, YARN por ejemplo, o usarlo en conjunto con otro sistema, STORM por ejemplo.
- Solucionar el reinicio en caliente de un plan de ejecución, evitar tener que reiniciar el motor cada vez que se incluya una *query*.
- Exprimir las operaciones complejas sobre los *stream* que ofrece.
- Herramienta web que se integre con curl, evitando que el usuario tenga que hacer uso de éste, para la interacción con el motor.
- Complementarlo con algún sistema o base de datos inteligente, ElasticSearch[33] por ejemplo, que consiga aportar valor a las soluciones ofrecidas por Siddhi CEP.

[33] ElasticSearch es un motor de búsqueda, catalogado dentro de las denominadas base de datos NoSQL, que utiliza los principios definidos en Apache Lucene.

6.3.2 Conclusión personal

En la realización de este proyecto, aparte de poner en práctica lo aprendido durante el Grado: diseño, desarrollo, documentación, solución de errores, cumplimiento de plazos, etc; también me ha abierto la mente a un mundo que desconocía, como es el Big Data. Aunque en este proyecto no se trate esa tecnología en concreto, sólo se menciona y usa Apache Kafka, ampliamente utilizado en dicho ecosistema. Durante la realización de este proyecto también realice, paralelamente, prácticas de empresa que me enseñaron lo que ese mundo puede aportar en un futuro y está aportando en el presente.

A la vista del resultado obtenido y de lo aprendido, valoro muy positivamente esta experiencia de la que mucho he aprendido y mucho me ha aportado, que es lo que al final te hace sentirte realizado.

REFERENCIAS

[1] R. Gerhards. The Syslog Protocol. Marzo 2009. Disponible en: https://tools.ietf.org/html/rfc5424.txt

[2] C. Lonvick. The BSD Syslog Protocol. Agosto 2001. Disponible en: https://www.ietf.org/rfc/rfc3164.txt

[3] R. Fielding; UC Irvine; J. Gettys; J. Mogul; H. Frystyk; T. Berners-Lee. Hypertext Transfer Protocol -- HTTP/1.1. Enero 1997. Disponible en : https://www.ietf.org/rfc/rfc2068.txt

[4] Adiscon. Welcome to Rsyslog. 2004.Disponible en: http://www.rsyslog.com/doc/v8-stable/

[5] Rainer Gerhards. Liblognorm 1.1.2 documentation. Disponible en: http://www.liblognorm.com/files/manual/index.html

[6] Apache. Kafka 0.8.2 Documentation. Disponible en: http://kafka.apache.org/082/documentation.html

[7] WSO2. About CEP. Disponible en: https://docs.wso2.com/display/CEP310/About+CEP

[8] WSO2. Siddhi Language Specification. Disponible en: https://docs.wso2.com/display/CEP310/Siddhi+Language+Specification

[9] Isabel Román Martínez . Apuntes Ingeniería de Software. Apuntes de asignatura.

[10] Scott Chacon;Ben Straub. *Pro Git* [en linea]. Updated: 2014. Formato en pdf con imágenes. Disponible en: https://progit2.s3.amazonaws.com/en/2016-03-22-f3531/progit-en.1084.pdf

[11] CCOO. NÓMINA CONVENIO TIC: Tablas salariales 2016. 2016. Disponible en : http://bankintercomite.es/pag/bk/general/nomina/tic/index.php?conc=tablas

[12] Jaime Márquez Fernández. Código y configuraciones desarrolladas. Septiembre 2016. Disponible en : https://github.com/jmf92/PFG

Índice de Conceptos

Glosario

API : Application Programming Interface 2

CEP : Complex Event Processing 1

CSV : Comma-Separated Values 39

HDFS : Hadoop Distributed File System 7

JSON : JavaScript Object Notation 1

JVM : Java Virtual Machine 30

LTS : Long Term Support 30

NoSQL: Not only SQL 58

REST : Representational State Transfer 2

SQL : Structured Query Language 7

SSH : Secure SHell 7

TIC : Tecnologías de la Información y la Comunicación 57

UML : Unified Modeling Language 24

URL : Uniform Resource Locator 6

XML: eXtensible Markup Language xx

YAML : Yet Ain´t Markup Language xx

YARN : Yet Another Resource Negotiator 58

ANEXO A: CONFIGURACIÓN APACHE2

A.1.Configuración servidor web Apache2

apache2.conf

```
##Archivo configuración servidor web Apache2#########################################################
#Esta configuración es la configuración por defecto, sólo se ha cambiado el formato de salida   #
#de los logs de acceso al servidor.                                                              #
###################################################################################################
#
# Do NOT add a slash at the end of the directory path.
#
#ServerRoot "/etc/apache2"

#
# The accept serialization lock file MUST BE STORED ON A LOCAL DISK.
#
Mutex file:${APACHE_LOCK_DIR} default

#
# PidFile: The file in which the server should record its process
# identification number when it starts.
# This needs to be set in /etc/apache2/envvars
#
PidFile ${APACHE_PID_FILE}

#
# Timeout: The number of seconds before receives and sends time out.
#
Timeout 300

#
# KeepAlive: Whether or not to allow persistent connections (more than
# one request per connection). Set to "Off" to deactivate.
#
KeepAlive On

#
# MaxKeepAliveRequests: The maximum number of requests to allow
# during a persistent connection. Set to 0 to allow an unlimited amount.
# We recommend you leave this number high, for maximum performance.
#
MaxKeepAliveRequests 100

#
# KeepAliveTimeout: Number of seconds to wait for the next request from the
# same client on the same connection.
#
KeepAliveTimeout 5


# These need to be set in /etc/apache2/envvars
User ${APACHE_RUN_USER}
Group ${APACHE_RUN_GROUP}

#
# HostnameLookups: Log the names of clients or just their IP addresses
# e.g., www.apache.org (on) or 204.62.129.132 (off).
# The default is off because it'd be overall better for the net if people
# had to knowingly turn this feature on, since enabling it means that
# each client request will result in AT LEAST one lookup request to the
# nameserver.
#
HostnameLookups Off
```

```
# ErrorLog: The location of the error log file.
# If you do not specify an ErrorLog directive within a <VirtualHost>
# container, error messages relating to that virtual host will be
# logged here.  If you *do* define an error logfile for a <VirtualHost>
# container, that host's errors will be logged there and not here.
#
#ErrorLog ${APACHE_LOG_DIR}/error.log
ErrorLog syslog:local6

#Redirigimos los logs de acceso, con nivel información, hacia rsyslog
CustomLog "|/usr/bin/logger -p local6.info -t access" apache_json

#Redirigimos, también hacia un fichero, los logs para depuración
CustomLog /var/log/apache2/access.log apache_json

#Aplicamos formato cee para que rsyslog pueda
#interpretar correctamente el JSON
LogFormat "@cee:{\
            \"time\":\"%t\",\
            \"message\":\"%h %l %u %t \\\"%r\\\" %>s %b\",\
            \"ip-source\": \"%a\",\
            \"duration\": %D,\
            \"status\": %>s,\
            \"request\": \"%U%q\",\
            \"urlpath\": \"%U\",\
            \"urlquery\": \"%q\",\
            \"bytes\": %B,\
            \"method\": \"%m\",\
            \"referer\": \"%{Referer}i\",\
            \"useragent\": \"%{User-agent}i\"\
           }" apache_json

# LogLevel: Control the severity of messages logged to the error_log.
# Available values: trace8, ..., trace1, debug, info, notice, warn, error, crit, alert, emerg.
# It is also possible to configure the log level for particular modules, e.g."LogLevel info ssl:warn
LogLevel warn

# Include module configuration:
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf

# Include list of ports to listen on
Include ports.conf

# Sets the default security model of the Apache2 HTTPD server. It does
# not allow access to the root filesystem outside of /usr/share and /var/www.
# The former is used by web applications packaged in Debian,
# the latter may be used for local directories served by the web server. If
# your system is serving content from a sub-directory in /srv you must allow
# access here, or in any related virtual host.
<Directory />
        Options FollowSymLinks
        AllowOverride None
        Require all denied
</Directory>

<Directory /usr/share>
        AllowOverride None
        Require all granted
</Directory>

<Directory /var/www/>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
</Directory>

# AccessFileName: The name of the file to look for in each directory
# for additional configuration directives.  See also the AllowOverride
# directive.
AccessFileName .htaccess
```

```
        # The following lines prevent .htaccess and .htpasswd files from being
        # viewed by Web clients.
        #
        <FilesMatch "^\.ht">
                Require all denied
        </FilesMatch>


        #
        # The following directives define some format nicknames for use with
        # a CustomLog directive.
        #
        # These deviate from the Common Log Format definitions in that they use %O
        # (the actual bytes sent including headers) instead of %b (the size of the
        # requested file), because the latter makes it impossible to detect partial
        # requests.
        #
        # Note that the use of %{X-Forwarded-For}i instead of %h is not recommended.
        # Use mod_remoteip instead.
        #
        LogFormat "%v:%p %h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" vhost_combined
        #LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" combined

        LogFormat "%h %l %u %t \"%r\" %>s %O" common
        LogFormat "%{Referer}i -> %U" referer
        LogFormat "%{User-agent}i" agent



        # Include of directories ignores editors' and dpkg's backup files,
        # see README.Debian for details.

        # Include generic snippets of statements
        #IncludeOptional conf-enabled/*.conf

        # Include the virtual host configurations:
        #IncludeOptional sites-enabled/*.conf

        # vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

ANEXO B: CONFIGURACIÓN RSYSLOG

B.1.Configuración Rsyslog para el servicio SSH

parse_ssh.conf

```
#############################################################
#              PROYECTO FIN GRADO                           #
#-----------------------------------------------------------#
#       Titulo: Normalización, almacenamiento               #
#               y correlación de eventos                    #
#       Autor: Jaime Márquez Fernández                      #
#                                                           #
#       Universidad:Escuela Superior Ingenieros             #
#                   Universidad Sevilla                     #
#       Curso: 2015/2016                                    #
#                                                           #
#############################################################

# Este fichero de configuración se encarga de obtener los logs
# del servicio ssh y extraer la información mediante el uso de
# reglas(liblognorm).
# Posteriormente se realiza el envio de dicha información al sistema
# Apache Kafka y se realiza una copia en un fichero local.

#module(load="omkafka") No hace falta cargarlo de nuevo
module(load="mmnormalize")
module(load="omelasticsearch")

####Plantilla para los logs del servicio ssh####
template(name="norm_ssh"
         type="list"){
         constant(value="{")
constant(value="\"message\":\"")                    property(name="msg" format="json")
constant(value="\",\"raw_message\":\"")             property(name="rawmsg")
constant(value="\",\"host\":\"")                    property(name="hostname" format="json")
constant(value="\",\"fromhost\":\"")                property(name="fromhost" format="json")
constant(value="\",\"fromhost-ip\":\"")             property(name="fromhost-ip" format="json")
constant(value="\",\"syslogtag\":\"")               property(name="syslogtag" format="json")
constant(value="\",\"program_name\":\"")            property(name="programname" format="json")
constant(value="\",\"pri\":\"")                     property(name="pri" format="json")
constant(value="\",\"pri-text\":\"")                property(name="pri-text" format="json")
constant(value="\",\"iut\":\"")                     property(name="iut" format="json")

constant(value="\",\"syslogfacility\":\"")          property(name="syslogfacility" format="json")
constant(value="\",\"syslogfacility_text\":\"")     property(name="syslogfacility-text" format="json")
constant(value="\",\"syslogseverity\":\"")          property(name="syslogseverity" format="json")
constant(value="\",\"syslogseverity_text\":\"")     property(name="syslogseverity-text" format="json")
constant(value="\",\"syslogpriority\":\"")          property(name="syslogpriority" format="json")
constant(value="\",\"syslogpriority_text\":\"")     property(name="syslogpriority-text" format="json")

constant(value="\",\"@timegenerated\":\"")          property(name="timegenerated" dateFormat="rfc3339"
constant(value="\",\"protocol-version\":\"")        property(name="protocol-version" format="json")
constant(value="\",\"structured-data\":\"")         property(name="structured-data" format="json")
constant(value="\",\"app-name\":\"")                property(name="app-name" format="json")
constant(value="\",\"procid\":\"")                  property(name="procid" format="json")
constant(value="\",\"msgid\":\"")                   property(name="msgid" format="json")
constant(value="\",\"inputname\":\"")               property(name="inputname" format="json")

constant(value="\",\"auth\":\"")                     property(name="$!auth" format="json")
constant(value="\",\"method\":\"")                   property(name="$!auth-method" format="json")
constant(value="\",\"user\":\"")                     property(name="$!username" format="json")
constant(value="\",\"ip\":\"")                       property(name="$!src-ip" format="json")
constant(value="\",\"port\":\"")                     property(name="$!src-port" format="json")
constant(value="\",\"protocol\":\"")                 property(name="$!protocol" format="json")
constant(value="\"}\\n")
        }
```

```
        #Extracción campos mediante reglas, usando el modulo liblognorm
        *.*             action(type="mmnormalize" ruleBase="/etc/rsyslog.d/ssh.rules")
        if $programname == 'sshd' then
        {
                action( type = "omelasticsearch"
                                template = "norm_ssh"
                                server = "localhost"
                                serverport = "9200"
                                searchIndex = "logs"
                                dynSearchIndex = "off"
                                searchType = "ssh"
                                dynSearchType = "off"
                                asyncrepl = "off"
                                usehttps = "off"
                                timeout = "1m"
                                bulkmode = "off"
                                )
        &       action( type="omfile"
                                file="/var/log/norm_ssh.log"
                                template="norm_ssh"
                                )



        #Acción para enviar los logs a Apache Kafka
        &   action(type="omkafka" topic="ssh_log" broker="localhost"
                    confParam=["compression.codec=snappy",
                                "socket.timeout.ms=10",
                                "socket.keepalive.enable=true"]
                    template="norm_ssh"

                                )
                stop
        }
```

ssh.rules

```
#############################################################
#              PROYECTO FIN GRADO                           #
#-----------------------------------------------------------#
#       Titulo: Normalización, almacenamiento               #
#                       y correlación de eventos            #
#       Autor: Jaime Márquez Fernández                      #
#                                                           #
#       Universidad:Escuela Superior Ingenieros             #
#                       Universidad Sevilla                 #
#       Curso: 2015/2016                                    #
#                                                           #
#############################################################

prefix=

rule=: %auth:word% %auth-method:word% for %username:word% from %src-ip:ipv4% port %src-port:number%
%protocol:word%
rule=: %auth:word% %auth-method:word% for invalid user %username:word% from %src-ip:ipv4%
port %src-port:number% %protocol:word%
```

B.2.Configuración Rsyslog para el servicio Apache2

parse_apache.conf

```
#############################################################
#       PROYECTO FIN GRADO                                  #
#-----------------------------------------------------------#
#       Titulo: Normalización, almacenamiento               #
#                       y correlación de eventos            #
#       Autor: Jaime Márquez Fernández                      #
#                                                           #
#       Universidad:Escuela Superior Ingenieros             #
#                           Universidad Sevilla             #
#       Curso: 2015/2016                                    #
#                                                           #
#############################################################

#Este fichero de configuración es utilizado para obtener los logs
#del servidor web Apache(en formato JSON) y sus campos significativos
#para su posterior almacenamiento en el sistema Apache Kafka


#Carga de los módulos necesarios
module(load="omkafka")
module(load="mmjsonparse")


####Plantilla para los logs con formato JSON del servicio Apache####
template(name="norm_apache" type="list"){
        property(name="$!all-json")
        constant(value="\n")
}


#Utilizamos el modulo mmjsonparse para obtener los campos.
#Si se obtienen con éxito realizamos el envio a Apache Kafka
# y una copia en un fichero local para tener redundancia
local6.info      action(type="mmjsonparse")

if $parsesuccess == "OK" then{

   action( type="omfile"
            file="/var/log/norm_apache.log"
            template="norm_apache"
            )

&   action(type="omkafka" topic="apache2_log" broker="localhost"
            confParam=["compression.codec=snappy",
                        "socket.timeout.ms=10",
                       "socket.keepalive.enable=true"]
            template="norm_apache"
           )
    stop
}
```

ANEXO C: MOTOR CORRELACIÓN

C.1.pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>


    <groupId>pfg</groupId>
    <artifactId>PFG_CorrelationService</artifactId>
    <version>1.0-SNAPSHOT</version>

    <repositories>
    <repository>
        <id>wso2-nexus</id>
        <name>WSO2 internal Repository</name>
        <url>http://maven.wso2.org/nexus/content/groups/wso2-public/</url>
        <releases>
            <enabled>true</enabled>
            <updatePolicy>daily</updatePolicy>
            <checksumPolicy>ignore</checksumPolicy>
        </releases>
    </repository>

    <repository>
        <id>wso2.releases</id>
        <name>WSO2 internal Repository</name>
        <url>http://maven.wso2.org/nexus/content/repositories/releases/</url>
        <releases>
            <enabled>true</enabled>
            <updatePolicy>daily</updatePolicy>
            <checksumPolicy>ignore</checksumPolicy>
        </releases>
    </repository>
    </repositories>


    <dependencies>
        <!-- Kafka -->
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka_2.10</artifactId>
            <version>0.8.2.0</version>
        </dependency>

        <!-- Jackson -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.4.1.1</version>
        </dependency>

        <!-- Siddhi -->
        <dependency>
            <groupId>org.wso2.siddhi</groupId>
```

```
        <artifactId>siddhi-core</artifactId>
        <version>3.0.0-alpha</version>
    </dependency>

    <dependency>
        <groupId>org.wso2.siddhi</groupId>
        <artifactId>siddhi-query-api</artifactId>
        <version>3.0.0-alpha</version>
    </dependency>

    <dependency>
        <groupId>org.wso2.siddhi</groupId>
        <artifactId>siddhi-query-compiler</artifactId>
        <version>3.0.0-alpha</version>
    </dependency>

    <!-- Yaml -->
    <dependency>
        <groupId>org.jyaml</groupId>
        <artifactId>jyaml</artifactId>
        <version>1.3</version>
    </dependency>

    <!-- Testing -->
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-all</artifactId>
        <version>1.10.19</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>

    <!—API Rest -->
    <dependency>
        <groupId>org.glassfish.grizzly</groupId>
        <artifactId>grizzly-http-server</artifactId>
        <version>2.3.16</version>
    </dependency>

    <dependency>
        <groupId>org.glassfish.jersey.containers</groupId>
        <artifactId>jersey-container-grizzly2-http</artifactId>
        <version>2.17</version>
    </dependency>

    <dependency>
        <groupId>org.glassfish.jersey.core</groupId>
        <artifactId>jersey-server</artifactId>
        <version>2.17</version>
    </dependency>
```

```
        <!-- JSON -->
        <dependency>
            <groupId>org.codehaus.jackson</groupId>
            <artifactId>jackson-mapper-asl</artifactId>
            <version>1.9.13</version>
        </dependency>

        <dependency>
            <groupId>org.json</groupId>
            <artifactId>json</artifactId>
            <version>20141113</version>
        </dependency>

        <dependency>
            <groupId>com.google.code.findbugs</groupId>
            <artifactId>jsr305</artifactId>
            <version>3.0.0</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                    <finalName>${project.artifactId}-${project.version}-
selfcontained</finalName>
                    <appendAssemblyId>false</appendAssemblyId>
                    <archive>
                        <manifest>
                            <mainClass>pfg.CorrelationService</mainClass>
                        </manifest>
                    </archive>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-release-plugin</artifactId>
                <version>2.5.1</version>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.18.1</version>
                <configuration>
                    <systemPropertyVariables>
                        <log4j.configuration>log4j-
tests.xml</log4j.configuration>
```

```
                    </systemPropertyVariables>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>1.2.1</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>exec</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <executable>java</executable>

<includeProjectDependencies>true</includeProjectDependencies>

<includePluginDependencies>false</includePluginDependencies>
                    <classpathScope>compile</classpathScope>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>7</source>
                    <target>7</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

C.2.CorrelationService

CorrelationService.java

```
package pfg;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import pfg.Kafka.KafkaConsumerManager;
import pfg.Kafka.KafkaProducer;
import pfg.Siddhi.SiddhiHandler;
import pfg.Rest.RestManager;
import pfg.Siddhi.StartChecking;
import pfg.Util.ConfigData;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Map;


/**
 * Created by jmf on 13/05/15.
 */
public class CorrelationService {
    public static final String DEFAULT_CONFIG_FILE = "./config/configFileCS.yml";
    private static final Logger log =
LoggerFactory.getLogger(CorrelationService.class);

    public static void main(String[] args) throws InterruptedException, IOException {
        ConfigData configData = new ConfigData();
        Map configMap;
        //Obtenemos la configuración, mediante fichero, de: kafka, zookeeper y REST
        //Si no se pasa ningún fichero, fijamos la configuración por defecto
        if (args.length >= 1)
            configMap=configData.load(args[0]);
        else
            configMap=configData.load(DEFAULT_CONFIG_FILE);

        //Si la configuración ha sido cargada exitosamente iniciamos el servicio
        if(configMap != null) {
            //Informamos de la configuración que será establecida
            log.info("Zookeeper: {} ", configMap.get("zookeeperURL").toString());
            log.info("ZookeeperGroup: {}", configMap.get("zookeeperGroup").toString());
            log.info("zookeeperSessionTimeout: {} ",
configMap.get("zookeeperSessionTimeout").toString());
            log.info("zookeeperSyncTime: {} ",
configMap.get("zookeeperSyncTime").toString());
            log.info("zookeeperCommitInterval: {} ",
configMap.get("zookeeperCommitInterval").toString());
            log.info("zookeeperOffsetReset: {} ",
configMap.get("zookeeperOffsetReset").toString());


            log.info("Kafkabroker: {}", configMap.get("kafkaBrokers").toString());
            log.info("kafkaSerializer: {}",
configMap.get("kafkaSerializer").toString());
            log.info("kafkaACK: {}", configMap.get("kafkaACK").toString());
            log.info("consumerTopic: {}", configMap.get("consumerTopic").toString());
            log.info("producerTopic: {}", configMap.get("producerTopic").toString());

            log.info("RestURI: {}", configMap.get("RestURI").toString());
```

```
            //Creamos un consumidor con las propiedades anteriormente inicializadas
            final KafkaConsumerManager kafkaConsumer = new KafkaConsumerManager(
                    configMap.get("kafkaBrokers").toString(),
                    configMap.get("kafkaSerializer").toString(),
                    configMap.get("kafkaACK").toString(),
                    configMap.get("consumerTopic").toString(),
                    configMap.get("zookeeperURL").toString(),
                    configMap.get("zookeeperGroup").toString(),
                    configMap.get("zookeeperSessionTimeout").toString(),
                    configMap.get("zookeeperSyncTime").toString(),
                    configMap.get("zookeeperCommitInterval").toString(),
                    configMap.get("zookeeperOffsetReset").toString()
                    );

            //Creamos un productor
            final KafkaProducer kafkaProducer = new
KafkaProducer(configMap.get("kafkaBrokers").toString(),
                    configMap.get("kafkaSerializer").toString(),
                    configMap.get("kafkaACK").toString(),
                    configMap.get("producerTopic").toString());

            //Creamos el manejador de siddhi
            SiddhiHandler siddhiHandler = new SiddhiHandler(kafkaConsumer,
kafkaProducer);

            StartChecking startChecking = new StartChecking(siddhiHandler);

            // RestManager inicia la API REST y redirige las peticiones de
inclusion/eliminación
            // que los usuarios añadirán a Siddhi.
            RestManager.startServer(siddhiHandler,
configMap.get("RestURI").toString());


            //Iniciamos el hilo que comprobará las peticiones de inicio
            startChecking.run();

            //Iniciamos el motor de correlación
            //siddhiHandler.start(executionPlan, queries);
            //siddhiHandler.recharge();
            //Finalizamos el motor  tras un tiempo de guarda
            //siddhiHandler.stop();
        }
        else
            log.error("Configuration is not loaded correctly");


    }

}
```

configFileCS.yml

```
#Fichero de configuración del servicio de correlación

#Zookeeper
zookeeperURL: localhost:2181
zookeeperGroup: log_management
zookeeperSessionTimeout: 400
zookeeperSyncTime: 200
zookeeperCommitInterval: 1000
zookeeperOffsetReset:smallest

#Kafka
kafkaBrokers: localhost:9092
kafkaSerializer: kafka.serializer.StringEncoder
kafkaACK: 1

#Kafka Consumer
consumerTopic: correlation_service

#Kafka Producer
producerTopic: correlation_service_out

#REST
RestURI: http://localhost:8888/CS/
```

C.3.Kafka

KafkaConsumerManager.java

```
package pfg.Kafka;

import kafka.consumer.ConsumerConfig;
import kafka.consumer.KafkaStream;
import kafka.javaapi.consumer.ConsumerConnector;
import pfg.CorrelationService;
import pfg.Siddhi.SiddhiHandler;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Esta clase se encarga de manejar los consumidores y
 * establecer la configuracion Kafka y Zookeeper de éstos,
 * para que se puedan conectarse a Kafka.
 *
 * @author Jaime Márquez Fernández
 */

public class KafkaConsumerManager {
    private ConsumerConnector consumer;
    private  ExecutorService executor;


    private String zookeeper;
    private String groupId;
    private String topic;
    private String kafkaBroker;


    private Object[] JsonSchema;

    public KafkaConsumerManager(String a_kafkaBroker,String a_kafkaSerializer,String
a_kafkaAcks, String a_topic, String a_zookeeper, String a_groupId,
String a_zookeeperTimeout,String a_zookeeperSyncTime, String a_zookeeperCommitInterval,
String a_zookeeperOffsetReset) {
        this.zookeeper = a_zookeeper;
        this.groupId = a_groupId;
        this.topic = a_topic;
        this.kafkaBroker = a_kafkaBroker;
        consumer = kafka.consumer.Consumer.createJavaConsumerConnector(
                createConsumerConfig(a_kafkaBroker, a_kafkaSerializer, a_kafkaAcks,
                        a_zookeeper, a_groupId, a_zookeeperTimeout,
a_zookeeperSyncTime, a_zookeeperCommitInterval, a_zookeeperOffsetReset ));


    }

    public void shutdown() {
```

```
        if (consumer != null) consumer.shutdown();
        if (executor != null) executor.shutdown();
        try {
            if (!executor.awaitTermination(5000, TimeUnit.MILLISECONDS)) {

                Logger.getLogger(KafkaProducer.class.getName()).log(Level.INFO,
                        "Timed out waiting for consumer threads to shut down, exiting
uncleanly");
            }
        } catch (InterruptedException e) {
            Logger.getLogger(CorrelationService.class.getName()).log(Level.SEVERE,
                    "Interrupted during shutdown, exiting uncleanly",e);
        }
    }

    public void run(int a_numThreads, SiddhiHandler siddhiHandler) {
        Map<String, Integer> topicCountMap =new HashMap<String, Integer>();
        topicCountMap.put(topic, new Integer(a_numThreads));
        Map<String, List<KafkaStream<byte[], byte[]>>> consumerMap =
consumer.createMessageStreams(topicCountMap);
        List<KafkaStream<byte[], byte[]>> streams = consumerMap.get(topic);


        // Creamos el manejador del hilo en el que se lanzará el consumidor
        executor = Executors.newFixedThreadPool(a_numThreads);

        //Creamos un objeto encargado de consumir los mensajes
        int threadNumber = 0;
        for (final KafkaStream stream : streams) {
            KafkaConsumer consumer = new KafkaConsumer(stream, threadNumber,
siddhiHandler);
            executor.submit(consumer);
            threadNumber++;
        }
    }

    public void setJsonSchema(Object[] jsonSchema){

        this.JsonSchema = jsonSchema;


    }

    public Object[] getJsonSchema(){
        return this.JsonSchema;
    }

    private static ConsumerConfig createConsumerConfig(String a_kafkaBroker,String
a_kafkaSerializer,String a_kafkaAcks, String a_zookeeper, String a_groupId,
String a_zookeeperTimeout,String a_zookeeperSyncTime, String a_zookeeperCommitInterval,
String a_zookeeperOffsetReset) {

        Properties props = new Properties();

        //Propiedades kafka
        props.put("metadata.broker.list", a_kafkaBroker);
        props.put("serializer.class", a_kafkaSerializer);
        props.put("request.required.acks", a_kafkaAcks);

        //Propiedades zookeeper
        props.put("zookeeper.connect", a_zookeeper);
        props.put("group.id", a_groupId);
```

```
        props.put("zookeeper.session.timeout.ms", a_zookeeperTimeout);
        props.put("zookeeper.sync.time.ms", a_zookeeperSyncTime);
        props.put("auto.commit.interval.ms", a_zookeeperCommitInterval);
        props.put("auto.offset.reset", a_zookeeperOffsetReset);

        return new ConsumerConfig(props);
    }

}
```

KafkaConsumer.java

```
package pfg.Kafka;

import kafka.consumer.ConsumerIterator;
import kafka.consumer.KafkaStream;

import org.codehaus.jackson.map.ObjectMapper;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.wso2.siddhi.core.stream.input.InputHandler;
import pfg.Siddhi.SiddhiHandler;

import java.util.Map;

/**
 * Esta clase se encarga de consumir los mensajes guardados en Apache Kafka
 * y generar los objetos de entrada al motor de correlación(Siddhi).
 *
 * @author Jaime Márquez Fernández
 *
 */



public class KafkaConsumer implements Runnable {
    private static final Logger log = LoggerFactory.getLogger(KafkaConsumer.class);

    private KafkaStream m_stream;
    private int m_threadNumber;
    private InputHandler inputHandler;
    private Object[] JsonSchema;
    private Object[] JsonAttributes;
    private SiddhiHandler siddhiHandler;


    public KafkaConsumer(KafkaStream a_stream, int a_threadNumber, SiddhiHandler
siddhiHandler) {
        this.m_threadNumber = a_threadNumber;
        this.m_stream = a_stream;
        this.siddhiHandler = siddhiHandler;
    }

    public void run() {
        ConsumerIterator<byte[], byte[]> it = m_stream.iterator();
        String linea;
```

```
        log.info("Kafka consumer is starting...");
        //Mientras el iterador indique que hay más elementos que leer
        //Seguiremos generando entradas en el motor
        while (it.hasNext()) {


            //Obtenemos la linea leida por el consumidor
            linea = new String(it.next().message());
            log.info("Thread: {}\t Message consumed:{}", m_threadNumber,linea);

            ObjectMapper mapper = new ObjectMapper();


            try {
                //Obtenemos el JSON de la linea leida
                Map<String, Object> json = mapper.readValue(linea, Map.class);
                JsonSchema = new Object[json.size()];
                JsonAttributes = new Object[json.size()];
                int i = 0;

                //Extraemos esquema y valor de atributos del JSON obtenido
anteriormente
                for (Map.Entry entry : json.entrySet()) {
                    JsonSchema[i] = entry.getKey();
                    JsonAttributes[i] = entry.getValue();

                    log.debug(JsonSchema[i].toString() + ":" +
JsonAttributes[i].toString());
                    i++;
                }



                //Comprobamos que el manejador de entrada del motor esté listo
                if(siddhiHandler.getInputHandler() != null) {

                    //Comprobamos si el manejador sigue siendo el mismo
                    if (inputHandler != siddhiHandler.getInputHandler()){

                        //Obtenemos el manejador de entrada desde el conector a siddhi
                        inputHandler = siddhiHandler.getInputHandler();
                    }

                    //Enviamos el valor de los atributos extraidos del objeto JSON al
input de Siddhi
                    inputHandler.send(JsonAttributes);
                }
                else
                    log.error("Siddhi input is out of service");

            } catch (Exception ie) {
                log.error(null, ie);
            }
        }

        log.info("Shutting down Thread: " + m_threadNumber);
    }
}
```

KafkaProducer.java

```
package pfg.Kafka;

import java.io.IOException;
import java.util.*;
import java.util.logging.Level;
import kafka.common.KafkaException;
import kafka.javaapi.producer.Producer;
import kafka.producer.KeyedMessage;
import kafka.producer.ProducerConfig;
import org.codehaus.jackson.map.ObjectMapper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.wso2.siddhi.core.event.Event;
/**
 * Esta clase tiene como función transformar la salida del motor de correlación
 * a mensajes Kafka
 *
 * @author Jaime Márquez Fernández
 */
public class KafkaProducer {
    private static final Logger log = LoggerFactory.getLogger(KafkaConsumer.class);
    private Producer producer;
    private KeyedMessage<String, String> data;
    private Object[] JsonSchema;
    private ObjectMapper mapper;
    private String topic;


    public KafkaProducer(String kafkabroker, String serializerProducer, String
numberACKS, String producerTopic) {

        this.topic = producerTopic;
        //Propiedades para poder conectar el productor a Kafka
        Properties props = new Properties();
        props.put("metadata.broker.list", kafkabroker);
        props.put("serializer.class", serializerProducer);
        //props.put("partitioner", "example.producer.SimplePartitioner");
        props.put("request.required.acks", numberACKS);

        ProducerConfig config = new ProducerConfig(props);

        producer = new Producer<String, String>(config);
        mapper = new ObjectMapper();

    }

    public void prepareProducer(Object[] jsonSchema){
        this.JsonSchema = jsonSchema;
    }
    public void useProducer(Event event) throws IOException {

        Map<String, Object> HashJson = new HashMap<String, Object>();

        //Generamos el JSON de salida con los datos ofrecidos por el motor de
correlación
        for (int i = 0; i< event.getData().length; i++)
        {
            HashJson.put(JsonSchema[i].toString(), event.getData(i).toString());
```

```
        }

        log.info(mapper.writeValueAsString(HashJson));

        //Creamos el mensaje Kafka que será enviado
        data = new KeyedMessage<String, String>(topic,
mapper.writeValueAsString(HashJson));


        try {
            //Enviamos el mensaje producido
            producer.send(data);


        }catch (KafkaException ke){
            log.error( "Kafka's message sending is not completed", ke);

        }

    }
    public void closeProducer() {
        log.info("Shutting down KafkaProducer");
        producer.close();
    }
}
```

C.3.Siddhi

SiddhiHandler.java

```
package pfg.Siddhi;

import org.codehaus.jackson.map.ObjectMapper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.wso2.siddhi.core.stream.input.InputHandler;
import pfg.Kafka.KafkaConsumerManager;
import pfg.Kafka.KafkaProducer;
import pfg.Rest.RestListener;


import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;


/**
 * Created by jmf on 8/07/15.
 */
public class SiddhiHandler implements RestListener{
    private static final Logger log = LoggerFactory.getLogger(SiddhiHandler.class);

    private ObjectMapper mapper;
    private String queries;
    private String InputStream;
    private String nameInputStream;
    private KafkaProducer producer;
    private KafkaConsumerManager consumer;
    private Siddhi siddhi;
    private InputHandler inputHandler;
    private int threads = 1;
    private Map<String, Map<String, Object>> rawQueries = new HashMap<String,
Map<String, Object>>();
    private boolean isStart = false;
    private boolean run = false;

    private ArrayList outStream;
    private boolean update;

    public SiddhiHandler(KafkaConsumerManager kafkaConsumer, KafkaProducer
kafkaProducer){
        this.producer = kafkaProducer;
        this.consumer = kafkaConsumer;
        this.mapper = new ObjectMapper();
        this.outStream = new ArrayList();
        this.InputStream = "";
        this.queries = "";

    }

    public InputHandler getInputHandler() {
        return inputHandler;
    }
    public  String getQueries(){return rawQueries.toString(); }
    public boolean isStart() {
```

```
        return isStart;
    }

    public void startSiddhi(){
        //Preparamos e iniciamos el motor de correlación
        siddhi = new Siddhi(InputStream, queries, producer);
        siddhi.start(nameInputStream, outStream);
        inputHandler = siddhi.getInputHandler();

        // Arrancamos, si no lo estaba ya, el consumidor de Kafka que
        // será el encargado de generar las entradas del motor
        if (!run)
            run();
    }

    public void run(){
        this.run = true;
        try {
            //Obtenemos los logs del topic de kafka, que serán la entrada del motor
            consumer.run(threads, this);
            Thread.sleep(10000);
        } catch (InterruptedException ie) {
            log.error(null, ie);
        }

    }


    public void stopSiddhi(){
        if(siddhi != null)
            siddhi.stop();

    }

    public void stop(){
        if(consumer != null)
            consumer.shutdown();
        else
            log.error("Error while shutting down kafka KafkaConsumer");

        if (producer != null)
            producer.closeProducer();
        else
            log.error("Error while shutting down Kafka Producer");


    }
```

```
    @Override
    public boolean startExecutionPlan(String newInputStream) {
        Map<String, Object> inputStream;
        boolean result = false;
        try {

            //Si el stream de entrada no está vacio: se inicializa un nuevo stream.
            //Si está vacío y hay una definición previa guardada: se reinicia el plan
anterior.
            //Sino: retornamos falso que indica que no hay ningun plan que poner en
marcha
            if(!newInputStream.isEmpty()) {
                inputStream = mapper.readValue(newInputStream, Map.class);
                InputStream="";
                InputStream = inputStream.get("stream").toString();
                nameInputStream = inputStream.get("name_stream").toString();
                isStart = true;
                result = true;

                log.info("Restart is upload");
                log.info("String stream:{}", newInputStream);
                log.info("Input:{}", InputStream);
                log.info("Queries:{}", queries);
            }else if (!InputStream.isEmpty()){
                result = true;
                isStart = true;
                log.info("Restart is upload");
            }else{
                result = false;
            }


         } catch (IOException e) {
            log.debug("Exception! {}", e.getMessage());
            log.error("Couldn't parse JSON query {}", newInputStream);
    }


        return result;
    }


    public void start(){
        //Si hay peticiones de carga de un nuevo plan o reinicio del anterior:
        //-Se comprueba si hay que actualizar las queries
        //-Paramos el plan ejecución anterior e iniciamos
        // el actualizado

            if(isStart){
                //Bajamos el flag de reinicios pendientes
                isStart = false;
                log.info("Received Restart request");


                //Paramos el motor para su posterior reinicio
                this.stopSiddhi();
                log.info("Upgrade is starting...");

                // Comprobamos si hay que actualizar los componentes del motor,
asociados a las queries:
                // +Queries
                // +Stream salida
```

```
            if (update)
                updateQueries();

            log.info("Consumer:{}",consumer.toString());
            log.info("Producer:{}", producer.toString());

            //Iniciamos el motor
            this.startSiddhi();
            log.info("Consumer:{}",consumer.toString());
            log.info("Producer:{}", producer.toString());
            log.info("Siddhi:{}", siddhi.toString());
        }


}


public void updateQueries(){
    log.info("Updating queries. . . ");

    // Si hay peticiones pendientes de inclusión/eliminación
    // actualizamos el valor del string  que contiene las queries
    // y el array que contiene los streams de salidas asociados a dichas queries.
    if (update){
        queries ="";
        if(!outStream.isEmpty())
            outStream.clear();
        for( Object query : rawQueries.keySet().toArray()){
            queries = queries + rawQueries.get(query).get("query").toString();
            outStream.add(rawQueries.get(query).get("OutStream"));
        }
        update = false;
    }


}



public boolean add(String newQuery) {
    Map<String, Object> query;
    boolean result = false;

    try {
        query = mapper.readValue(newQuery, Map.class);

        String id = query.get("id").toString();

        // Comprobamos que el identificador de query, y por tanto la query,
        // no esté repetido
        if (rawQueries.containsKey(id)) {
            log.error("Query with id {} already exist", id);
        } else {

            // Si el stream de salida es válido, no está registrado aún,
            // actualizamos las queries
            if(!outStream.contains(query.get("OutStream"))){
                result = true;

                //Añadimos la query
                rawQueries.put(id, query);
```

```
                    // Subimos el flag de actualizaciones pendientes.
                    if(!update)
                        update = true;

                    log.info("Current queries: {}", rawQueries.toString());
                    log.info("Siddhi queries: {}", queries);
                    log.info("New query added: {}", query);
                }
                else
                    log.error("Query with outputStream{} is already used",
query.get("OutStream"));
            }
        } catch (IOException e) {
            log.debug("Exception! {}", e.getMessage());
            log.error("Couldn't parse JSON query {}", newQuery);
        }

        return result;
    }


    public boolean remove(String id) {
        boolean removed = (rawQueries.remove(id) != null);


        // Si ha sido eliminada con éxito, actualizamos las queries
        // y el stream de salida asociado a ésta
        if (removed) {

            // Subimos el flag de actualizaciones pendientes.
            if(!update)
                update = true;

            log.info("Query with the id {} has been removed", id);
            log.info("Current queries: {}", rawQueries);
        } else {
            log.error("Query with the id {} is not present", id);
        }

        return removed;
    }

}
```

Siddhi.java

```
package pfg.Siddhi;

import kafka.Kafka;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.wso2.siddhi.core.ExecutionPlanRuntime;
import org.wso2.siddhi.core.SiddhiManager;
import org.wso2.siddhi.core.event.Event;
import org.wso2.siddhi.core.stream.input.InputHandler;
import org.wso2.siddhi.core.stream.output.StreamCallback;
import org.wso2.siddhi.query.api.ExecutionPlan;
import pfg.Kafka.KafkaProducer;

import java.io.IOException;
import java.util.ArrayList;

/**
 * Created by jmf on 18/08/15.
 */
public class Siddhi {
    private static final Logger log = LoggerFactory.getLogger(Siddhi.class);

    private SiddhiManager siddhiManager;
    private ExecutionPlanRuntime executionPlanRuntime;
    private InputHandler inputHandler;
    private KafkaProducer kafkaProducer;


    public Siddhi(String inputStream, String queries, KafkaProducer producer){

        log.info("InputStream:{}\n Queries:{}\n", inputStream, queries);

        this.kafkaProducer = producer;
        this.siddhiManager = new SiddhiManager();
        this.executionPlanRuntime =
siddhiManager.createExecutionPlanRuntime(inputStream+queries);

    }
    public InputHandler getInputHandler(){
        return this.inputHandler;
    }
    public void start(String nameInputStream, ArrayList outStream){

        //Añadimos los callbacks
        //Creamos los callbacks para la salida generada por el motor
        addCallbacks(outStream);


        //Definimos el manejador de los eventos de entrada del motor
        inputHandler = executionPlanRuntime.getInputHandler(nameInputStream);
        log.info("Generating input handler");

        //Ejecutamos el plan de ejecución
        executionPlanRuntime.start();
        log.info("Starting execution Plan Runtime");

    }
    public void stop(){
```

```
        if (executionPlanRuntime != null)
            executionPlanRuntime.shutdown();

        if(siddhiManager != null)
            siddhiManager.shutdown();

    }

    public void addCallbacks(ArrayList OutStream){

        for (Object outStream: OutStream){
            log.info("Starting callback:{}", outStream.toString());
            addCallback(outStream.toString());
        }

    }

    public void addCallback(final String Stream){

        executionPlanRuntime.addCallback(Stream, new StreamCallback() {
            @Override
            public void receive(Event[] inEvents) {
                log.info("Callback{} is working", Stream);
                log.info("Producer{}", kafkaProducer.toString());
                log.info("ExecPlanRunTime{}", executionPlanRuntime);

                //Preparamos el productor con el esquema del Stream de salida

kafkaProducer.prepareProducer(executionPlanRuntime.getStreamDefinitionMap().get(Stream).g
etAttributeNameArray());

                for (Event e : inEvents) {

                    log.info("Consumed event:{} ", e);

                    //Iniciamos el productor
                    try {
                        kafkaProducer.useProducer(e);
                    } catch (IOException ioe) {
                        log.error(null, ioe);
                    }


                }
                //kafkaProducer.closeProducer();
            }
        });
        log.info("Callback with id {} is created", Stream);
        log.info("Producer{}", kafkaProducer.toString());
        log.info("ExecPlanRunTime{}", executionPlanRuntime);
    }
}
```

StartChecking.java

```
package pfg.Siddhi;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import pfg.Rest.RestListener;

/**
 * Created by jmf on 19/08/15.
 */
public class StartChecking implements Runnable {
    private static final Logger log = LoggerFactory.getLogger(StartChecking.class);
    private SiddhiHandler siddhiHandler;

    public StartChecking(SiddhiHandler siddhiHandler){
        log.info("Thread(StartChecking) is waiting start request...");
        this.siddhiHandler = siddhiHandler;
    }

    @Override
    public void run(){
        if(siddhiHandler != null) {
            while (true) {
                log.debug("Waiting requests...");
                if(siddhiHandler.isStart()){
                    log.info("Restart request was received");
                    siddhiHandler.start();
                }
            }
        }
        else
            log.error("Siddhi handler was not started");
    }
}
```

B.4.Rest

RestListener.java

```
package pfg.Rest;
public interface RestListener {
    boolean add(String newQuery);
    boolean remove(String id);
    String getQueries();
    boolean startExecutionPlan(String newExecutionPlan);
}
```

RestManager.java

```
package pfg.Rest;

import org.glassfish.grizzly.http.server.HttpServer;
import org.glassfish.jersey.grizzly2.httpserver.GrizzlyHttpServerFactory;
import org.glassfish.jersey.server.ResourceConfig;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.net.URI;

public class RestManager {
    private static final Logger log = LoggerFactory.getLogger(RestManager.class);

    // URI donde el servidor HTTP Grizzly escuchará las peticiones
    private static final String BASE_URI = "http://localhost:8888/myapp/";
    private static RestListener listener = null;
    private static ResourceConfig rc;
    private static HttpServer server;
    private RestManager() {}

    /**
     * Iniciamos el servidor HTTP Grizzly que ofrece los programas Java definidos en
esta aplicación
     *
     */
    public static void startServer(RestListener rl, String uri) {
        //Creamos una configuración de fuentes que busca en el paquete pfg.Rest
        //los proveedores de JAX-RS(servidor aplicaciones java)
        rc = new ResourceConfig().packages("pfg.Rest");

        // Establecemos el "listener" para comunicar siddhi con la API REST
        listener = rl;

        // Creamos e iniciamos una instancia del servidor que escuchará en la URI
definida
        server = GrizzlyHttpServerFactory.createHttpServer(URI.create(uri), rc);
        log.info("Starting server...");
    }

    public static void stopServer() {
        if (server != null) {
            server.shutdown();
        }
    }

    public static RestListener getListener() {
        return listener;
    }
}
```

Resource.java

```
package pfg.Rest;

import javax.inject.Singleton;
import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

@Singleton
@Path("res")
public class Resource {
    /**
     * Este método maneja peticiones HTTP POST para la inclusión de nuevas "queries"
     * en el motor de correlación.
     *
     * Utiliza una interfaz con el manejador del motor para comunicar dicha información.
     *
     * @param json Cadena en formato JSON.
     *
     * @return Respuesta HTTP con el código apropiado tras tratar la petición.
     *
     * @author Jaime Márquez Fernández
     */

    @POST
    @Path("/action=add")
    @Consumes(MediaType.APPLICATION_JSON)
    public Response addQuery(String json) {
        RestListener listener = RestManager.getListener();

        // Check if the listener accepted the data
        if (listener == null) {
            return Response.status(500).build();
        } else if (listener.add(json)) {
            return Response.status(200).entity(json).build();
        } else {
            return Response.status(202).entity(json).build();
        }
    }

    /**
     * Este método maneja peticiones HTTP GET para la obtención de las queries definidas
     * en el motor
     *
     * Utiliza una interfaz con el manejador del motor para comunicar dicha información.
     *
     *
     * @return Respuesta HTTP con el código apropiado tras tratar la petición.
     *
     * @author Jaime Márquez Fernández
     */

    @GET
    @Path("/queries")
    @Produces(MediaType.TEXT_PLAIN)
    public Response getQueries() {
        RestListener listener = RestManager.getListener();

        // Check if the listener accepted the data
        if (listener == null) {
```

```
            return Response.status(500).build();
        }
        else{
            return Response.status(200).entity(listener.getQueries()+'\n').build();
        }
    }


    /**
     * Este metodo maneja las peticiones HTTP POST para
     * iniciar el motor de correlación.
     *
     * Utiliza la interfaz con el manejador del motor para pasarle la información.
     *
     * @param newExecutionPlan Cadena en formato JSON.
     *
     * @return Respuesta HTTP con el código apropiado tras tratar la petición.
     *
     * @author Jaime Márquez Fernández
     */

    @POST
    @Path("/action=start")
    @Consumes(MediaType.APPLICATION_JSON)
    public Response start(String newExecutionPlan) {
        RestListener listener = RestManager.getListener();


        // Check if the listener accepted the data
        if (listener == null) {
            return Response.status(500).build();
        } else if (listener.startExecutionPlan(newExecutionPlan)) {
            return Response.status(200).entity(newExecutionPlan).build();
        } else {
            return Response.status(202).entity("Execution plan is not defined
yet\n").build();
        }
    }




    /**
     * Este método maneja las peticiones HTTP DELETE utilizadas parar eliminar la query
con ID= " id"
     * aplicada a un determinado plan de ejecución en el motor.
     *
     * @param id Se pasa como parámetro el identificador de la query que se desea
eliminar
     *
     * @return Respuesta HTTP con el codigo apropiado tras tratar la petición.
     */

    @DELETE
    @Path("/action=delete/query/{id}")
    public Response removeQuery(@PathParam("id") String id) {
        RestListener listener = RestManager.getListener();

        // Check if the listener accepted the operation
        if (listener == null) {
            return Response.status(500).build();
        } else if (listener.remove(id)) {
```

```
            return Response.status(200).build();
        } else {
            return Response.status(404).entity("Query with the id " + id + " is not
present").build();
        }
    }
}
```

B.5.Util

ConfigData.java

```
package pfg.Util;

import org.ho.yaml.Yaml;
import org.ho.yaml.exception.YamlException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Map;

/**
 * Created by jmf on 19/08/15.
 */
public class ConfigData {
    private static final Logger log = LoggerFactory.getLogger(ConfigData.class);

    private Map map;
    public ConfigData(){}

    public Map load(String configFile) {

        try {
            this.map = (Map<String, Object>) Yaml.load(new File(configFile));
        } catch (FileNotFoundException e) {
            log.error("Couldn't find config file {}", configFile);
            System.exit(1);
        } catch (YamlException e) {
            log.error("Couldn't read config file {}. Is it a YAML file?", configFile);
            System.exit(1);
        }
    return map;
    }
}
```