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

Desarrollo de un Sistema de Gestión Syslog en Cloud.

Autor: Alejandro Rodríguez Calzado
Tutor: Pablo Nebrera Herrera

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

Sevilla, 2016

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

Desarrollo de un Sistema de Gestión Syslog en Cloud.

Autor:
Alejandro Rodríguez Calzado

Tutor:
Pablo Nebrera Herrera
Profesor titular

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

Trabajo Fin de Grado: Desarrollo de un Sistema de Gestión Syslog en Cloud.

Autor: Alejandro Rodríguez Calzado

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 todas las personas que me han apoyado desde el primer día y sé que seguirán haciéndolo.

Agradecimientos

En el momento de escribir estas líneas, me doy cuenta de que estoy a punto de dar por cerrada una etapa de mi vida. Una etapa de mi vida en la que he pasado por muchísimos momentos durísimos que han requerido de mí un esfuerzo sobrenatural. Sin embargo, no consigo acordarme sino de los maravillosos momentos que se han intercalado entre esos otros que finalmente acaban por olvidarse: las risas con los amigos de la facultad, los días de relax con Silvia, la expresión de mis padres y hermano rebosante de orgullo al ver que consigo mis objetivos, etc. Todos estos momentos tienen en común la compañía de personas fabulosas que sin duda merecen un enorme agradecimiento por mi parte.

Gracias a mis padres, María Jesús y Antonio, por hacer todo lo posible e imposible por ayudarme a conseguir este sueño, por ser un ejemplo a seguir y, por enseñarme que la única llave que abre todas las puertas en esta vida, es el esfuerzo.

Gracias a mi hermano, Álvaro, por ser para mí un ejemplo de superación y por apoyarme en todo momento. Estoy orgulloso de ti, sigue así y llegarás lejos.

Gracias a mi pareja, Silvia, por secarme las lágrimas en los malos momentos y saltar conmigo en los momentos de alegría. Por enseñarme a disfrutar la vida como nunca antes lo había hecho y disfrutarla junto a mí. Te quiero.

Gracias también al resto de mi familia y la familia de Silvia, cada uno de los cuáles han supuesto un apoyo constante durante estos cuatro años y me han brindado su ayuda siempre que ha sido necesaria.

Gracias a los que en su día conocí como compañeros de clase y hoy puedo llamar amigos: Fran, Caye, Sergio, Ricardo, Franchi, Miguel, Ana "Posu", Miguel Ángel… Gracias por hacer más llevaderos los cursos y por los atracones de pizza acompañados de risas.

Por último, gracias a mi tutor, Pablo Nebrera, por confiar en mí para este proyecto, guiarme durante su desarrollo y darme la oportunidad de formar parte del equipo de redBorder. Gracias a ellos también, por resolver cualquier duda que plantease y tratarme como uno más desde el primer momento.

A todos ellos,

GRACIAS.

Alejandro Rodríguez Calzado

Estudiante del Grado en Ingeniería de las Tecnologías de Telecomunicación

Sevilla, 2016

Resumen

Este trabajo fin de grado se centra en el desarrollo de un sistema de gestión de logs para la plataforma de visibilidad de red en tiempo real y ciberseguridad de la empresa redBorder®. Dicho sistema pasará a llamarse redBorder Vault.

Comienza con una investigación sobre logs y el protocolo Syslog, para posteriormente definir la estructura del sistema y los servicios usados para el desarrollo del mismo. Entre estos servicios se usan algunos como Rsyslog, Kafka o Chef, que también son tema central de algunos apartados, ya que sobre ellos se sustenta el desarrollo de este proyecto. Posteriormente se definen los aspectos más técnicos de la implementación realizada y de la integración del sistema en la plataforma redBorder.

Para la implementación del sistema, se desarrolla inicialmente una configuración para que los diferentes servicios trabajen conjuntamente en un entorno Centos sin modificaciones. Esta configuración debe permitir recibir logs de diferentes fuentes y usando diferentes protocolos de transporte, parsear toda la información posible del log y enviar a un topic de Kafka en un formato común (JSON) que facilite el procesado de la información.

Para la integración en redBorder, usaremos Chef que es un framework de automatización de infraestructuras de sistemas capaz de facilitar la tarea de instalación y configuración de herramientas. Gracias a Chef, todos los nodos que den soporte a nuestro sistema se configurará automáticamente y mantendrán siempre la última versión de esta configuración.

Abstract

This end-of-degree project focuses on developing a log management system for the real-time network visibility and cybersecurity platform of redBorder® team. This system will be called redBorder Vault.

It begins with an investigation into logs and Syslog protocol, and defines the structure of the system and services used for development. Some of these services are Rsyslog, Kafka or Chef, that are also main theme of some sections in this document. Afterward technical aspects of the implementation and system integration in redBorder platform are defined.

To implement the system, we have initially developed a configuration so that different services work together in a Centos environment without modification. This configuration should allow receiving logs from different sources and using different transport protocols, parsing all possible log information and sending it to a Kafka topic in a common format (JSON) to make easier to process information.

For integration into redBorder, we'll use Chef. Chef is a system infrastructure automation framework that makes easier to install and configure tools. Thanks to Chef, all nodes that support our system are automatically set up with the last configuration version.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xv**

Índice de Tablas **xix**

Índice de Figuras **xxi**

1 Introducción **1**
1.1 Motivación *1*
1.2 Situación actual *2*
1.3 Objetivos *2*
1.4 Metodología *3*

2 Logs **5**
2.1 ¿Qué es un log? *5*
2.1.1 Categorías 5
2.2 ¿Por qué son importantes? *6*
2.2.1 Administración de recursos 6
2.2.2 Detección de intrusos 7
2.2.3 Solución de problemas 7
2.2.4 Análisis forense 8
2.3 Tipos, formatos y sintaxis *8*
2.3.1 Sintaxis 11
2.3.2 Contenido 11
2.4 Ciclo de vida de los logs *12*
2.4.1 Generación de logs 12
2.4.2 Transmisión y recolección 12
2.4.3 Filtrado y normalización 14

3 El protocolo Syslog **17**
3.1 Definiciones y arquitectura *17*
3.2 Formato del mensaje Syslog *21*
3.2.1 Longitud del mensaje 21
3.2.2 Cabecera 21
3.2.3 Datos estructurados 24
3.2.4 Mensaje 26
3.2.5 Ejemplos 26

4 Servicios y herramientas usados **29**
4.1 Rsyslog *29*
4.1.1 imuxsock 31
4.1.2 imfile 31
4.1.3 imudp 31

4.1.4 imtcp 31
4.1.5 pmrfc3164 31
4.1.6 pmrfc5424 31
4.1.7 mmjsonparse 31
4.1.8 mmpstrucdata 31
4.1.9 mmnormalize 32
4.1.10 mmrfc5424addhmac 32
4.1.11 omfile 32
4.1.12 omfwd 32
4.1.13 omkafka 32
4.2 Kafka *32*
4.3 Zookeeper *34*
4.4 Chef *35*
4.5 K2http *37*
4.6 Liblognorm *37*
4.6.1 Lognormalizer 37
4.7 Logger *37*
4.8 Certtool *37*

5 Estructura del sistema **39**

6 Desarrollo y pruebas en Centos **43**
6.1 Configuración del cliente. *45*
6.1.1 UDP 46
6.1.2 TCP 47
6.1.3 TLS 48
6.2 Configuración del servidor *50*
6.2.1 Recepción de mensajes Syslog 51
6.2.2 Parseado de cabecera Syslog 52
6.2.3 Cálculo del hash 53
6.2.4 Parseado de datos estructurados 53
6.2.5 Filtrado de mensajes 54
6.2.6 Parseado del mensaje 55
6.2.7 Definición de la plantilla de salida 55
6.2.8 Copia local en fichero 57
6.2.9 Enviar a Kafka 57

7 Integración en redBorder **59**
7.1 Modificaciones del cookbook para Rsyslog *59*
7.1.1 attributes 60
7.1.2 resources 61
7.1.3 providers 61
7.1.4 recipes 64
7.1.5 templates 64
7.2 Reglas iptables *65*
7.3 Otras modificaciones *65*

8 Parseadores y resultados **67**
8.1 Parseador ssh *68*
8.2 Parseador iptables *69*
8.3 Parseador Cisco Wireless LAN Controller *70*
8.4 Parseador Apache-access *71*
8.5 Otros casos *73*

9 Presupuesto **75**

10 Planificación **77**

11 Conclusiones **79**
11.1 Líneas de desarrollo *79*
Referencias **81**
Índice de Conceptos **83**
Glosario **85**
Anexo A: RedBorder Client Proxy – Instalación y configuración inicial **87**
Anexo B: Rsyslog – Instalación **95**
Anexo C: Rsyslog – Generación de certificados **101**
Anexo D: Rsyslog – Configuración del Client Proxy **107**
Anexo E: Zookeeper – Instalación y Configuración **121**
Anexo F: Kafka – Instalación y Configuración **125**
Anexo G: Chef – Cookbook del Client Proxy **131**

ÍNDICE DE TABLAS

Tabla 2-1. Resumen tipos de logs. 10
Tabla 3-1. Códigos de recursos en el protocolo Syslog. 22
Tabla 3-2. Códigos de severidad en el protocolo Syslog. 22
Tabla 3-3. SD-IDs registrados en IANA. 24
Tabla 3-4. SD-PARAMs especificados para el SD-ID “timeQuality”. 25
Tabla 3-5. SD-PARAMs especificados para el SD-ID “origin”. 25
Tabla 3-6. SD-PARAMs especificados para el SD-ID “meta”. 25
Tabla 9-1. Presupuesto del proyecto. 75

ÍNDICE DE FIGURAS

Figura 1-1. Interfaz de usuario de Loogly. 2
Figura 1-2. Arquitectura general de redBorder. 3
Figura 2-1. Visor de eventos de Windows. 10
Figura 2-2. Recolección tradicional de logs en un servidor. 13
Figura 2-3. Recolección distribuida de logs en un servidor. 14
Figura 3-1. Arquitectura de capas Syslog. 18
Figura 3-2. Algunos posibles escenarios Syslog. 19
Figura 3-3. Escenario Syslog multiprotocolo. 20
Figura 3-4. Formato de timestamp Syslog. 23
Figura 4-1. Algunos de las posibles fuentes y destinos de Rsyslog. 30
Figura 4-2. Workflow de Rsyslog. 30
Figura 4-3. Elementos de Kafka. 32
Figura 4-4. Particionado de un topic. 33
Figura 4-5. Grupos de consumidores de Kafka. 34
Figura 4-6. Diagrama de comunicación entre Zookeeper y Kafka. 35
Figura 4-7. Componentes principales de Chef. 36
Figura 5-1. Dashboard de redBorder. 39
Figura 5-2. Estructura modular. 41
Figura 6-1. Escenario de desarrollo y prueba de configuración de Rsyslog. 44
Figura 6-2. Diagrama de flujo del procedimiento seguido en el cliente. 45
Figura 6-3. Captura Wireshark: Syslog sobre UDP. 47
Figura 6-4. Captura Wireshark: Syslog sobre TCP. 47
Figura 6-5. Proceso de creación de certificados. 48
Figura 6-6. Captura Wireshark: Syslog sobre TLS. 49
Figura 6-7. Diagrama de flujo que describe el proceso seguido en el servidor. 50
Figura 7-1. Diagrama de flujo acción “:config”. 62
Figura 8-1. Consumidor Kafka ecuchando en el topic “rb_vault”. 68
Figura 10-1. Diagrama de Gantt del proyecto. 77
Figura A-1. Opciones del instalador de redBorder Client Proxy. 87
Figura A-2. Configuración de instalación de redBorder Client Proxy. 88
Figura A-3. Etapa de verificación de dependencias. 89
Figura A-4. Etapa de instalación de paquetes. 89

Figura A-5. Etapa de ejecución de scripts post-instalación. 90
Figura A-6. Información mostrada al iniciar el Client Proxy. 90
Figura A-7. Apartado “Sensors” del manager de redBorder. 91
Figura A-8. Diálogo emergente “Claim a sensor”. 91
Figura A-9. Apartado “Sensors” del manager de redBorder (2). 91
Figura A-10. Apartado “Sensors” del manager de redBorder (3). 92
Figura A-11. Ejecución del script “rb_get_services.sh”. 92
Figura A-12. Vista “Monitor” del manager de redBorder. 93
Figura B-1. Actualización de Rsyslog fallida. 95
Figura B-2. Descarga del descriptor de repositorio de Rsyslog. 96
Figura B-3. Instalación de Rsyslog correcta. 96
Figura B-4. Verificación de la versión de Rsyslog. 96
Figura B-5. Descarga del código fuente de Rsyslog. 97
Figura B-6. Mensaje de dependencia no encontrada. 98
Figura B-7. Módulos habilitados en la ejecución de “configure”. 98
Figura E-1. Descarga de Zookeeper. 121
Figura E-2. Iniciar el servidor Zookeeper. 122
Figura E-3. Estado del servicio Zookeeper. 123
Figura F-1. Descarga de Kafka. 125
Figura F-2. Iniciar el servidor Kafka. 126

1 INTRODUCCIÓN

We're not in an information age anymore. We're in the information management age.

- Chris Hardwick -

Cada vez son más complejas las infraestructuras de red usadas por las empresas ya sea para uso privado o para ofrecer sus servicios a los clientes. Esta complejidad implica un mayor esfuerzo a la hora de realizar cualquier tarea de mantenimiento, mejora o solución de problemas.

Los logs nos cuentan la historia de nuestro sistema, qué ha ocurrido en cada instante y qué lo ha provocado. Todos y cada uno de los equipos existentes en una red (routers, switches, firewalls, controllers, servidores, …) generan logs y los almacenan para que los administradores de sistemas puedan consultarlos en caso de ser necesario. Esto puede suponer varios problemas entre los que podemos destacar los siguientes:

- Cada aplicación y cada equipo genera logs en un formato propio.
- Los logs están descentralizados.
- Los administradores de sistemas no tienen permiso para acceder a cada dispositivo o servidor, o bien, no saben dónde están los logs que le pueden servir de ayuda en cada momento.

Estos y otros problemas pueden ser solucionados usando un software de gestión de logs como se pretende hacer en este proyecto.

1.1 Motivación

Actualmente la plataforma redBorder ofrece en su manager múltiples vistas (cada vista se centra en un tipo de dato) para facilitar la visibilidad de la red de sus clientes por medio de diferentes modelos de gráficas y el uso de dashboards generados a partir de estas.

La idea de este proyecto nace de la necesidad de integrar una herramienta de gestión de logs en la plataforma redBorder. De esta forma, los administradores de red usuarios de redBorder no tendrían la necesidad de recurrir a otro software para gestionar sus logs y acceder a la incontable información que estos le pueden ofrecer. Así mismo, podrían usar toda la potencia de visibilidad del manager de redBorder aplicada a la gestión de logs, lo que supone una gran ayuda en cualquier tarea que requiera consultar los logs.

Teniendo en cuenta las ventajas que esto supondría, se encarga al autor de este proyecto la tarea de recolectar los logs y hacerlos llegar en el formato correcto al manager de redBorder para que el equipo de diseño web desarrolle una vista capaz de presentar estos datos.

1.2 Situación actual

El proyecto de redBorder Vault se inicia por el autor de esta memoria partiendo desde cero y siguiendo las pautas establecidas por la empresa. Aun siendo un proyecto nuevo, podemos encontrar varios referentes de otras empresas que nos pueden servir como ejemplo.

Podemos mencionar por ejemplo a Loggly, que ofrece un servicio de logging[1] en Cloud entre cuyas características se encuentran la centralización de logs, filtrado por campos, generación de gráficas, etc.

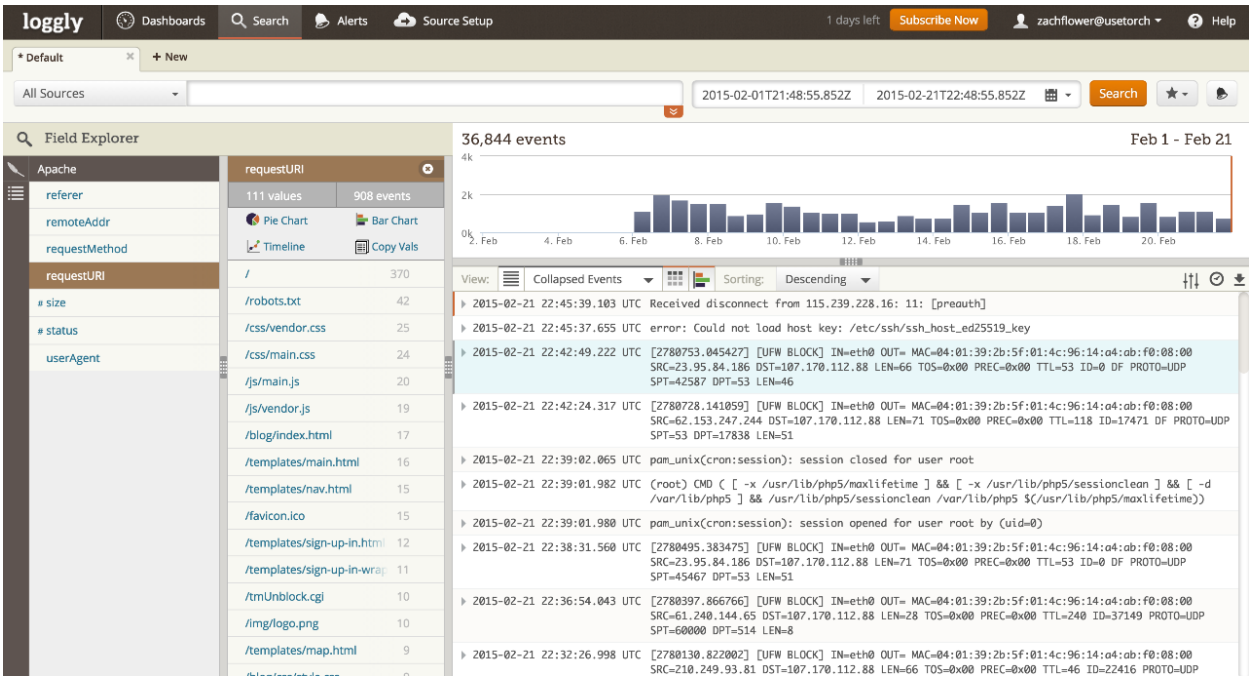


Figura 1-1. Interfaz de usuario de Loogly.

En la imagen anterior podemos ver la interfaz de usuario de Loggly compuesta por una barra de búsqueda, un panel lateral de filtrados por campos, una gráfica que contabiliza el número de eventos en el tiempo, y un listado de logs ordenados de más reciente a más antiguo.

Los desarrolladores web de la empresa redBorder se centrarán próximamente en la creación de una vista exclusiva en el manager para redBorder Vault que ofrezca al usuario una interfaz al estilo de Loggly u otros proveedores de servicios de logging en Cloud. Se pretende facilitar al usuario el filtrado de logs atendiendo a valores de ciertos campos y la generación de gráficas del formato de las ya presentes en el manager de redBorder.

1.3 Objetivos

Para que esta futura vista sea posible, es requisito previo la realización de este proyecto, ya que el manager debe obtener los datos (logs) desde los distintos elementos, en un formato común.

Teniendo en cuenta esto, el objetivo principal de este proyecto es configurar e integrar en la plataforma redBorder las herramientas necesarias para recolectar logs de diferentes fuentes, parsear y normalizar estos logs, y hacerlos llegar al manager de redBorder en formato JSON para mostrarlos al usuario.

[1] Cuando hablamos de logging nos referimos al acto de registrar un evento en un mensaje log y almacenarlo en algún lugar, ya sea localmente o remotamente.

Para entender esto mejor, es necesario una introducción a los elementos de la plataforma redBorder (se desarrollará más detalladamente en el capítulo 5). En la figura que aparece a continuación se puede observar como los datos recogidos en la red del cliente no se envían directamente al manager, sino que pasan por un equipo llamado Client Proxy. Este equipo es el encargado de recolectar los datos de la red del cliente, prepararlos y enviarlos al manager por medio de un túnel cifrado. El cliente puede acceder a las vistas del manager por medio de un navegador web.

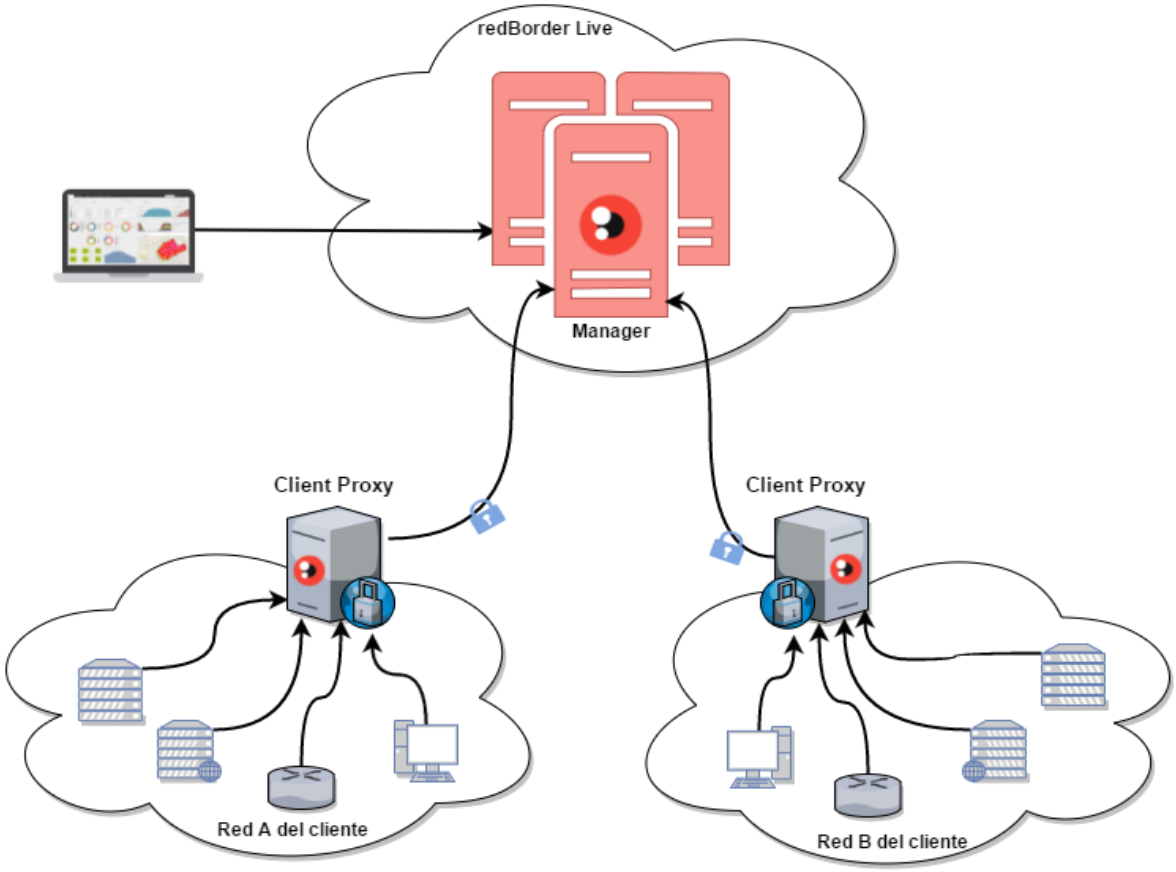


Figura 1-2. Arquitectura general de redBorder.

Una vez explicada la arquitectura general de la plataforma redBorder, podemos repetir cuál es el objetivo principal del proyecto, pero esta vez refiriéndolo a esta arquitectura: se debe habilitar el Client Proxy para recibir logs de los diferentes equipos de la red del cliente, parsear esto logs, normalizarlos, y enviarlos por el túnel cifrado al manager de redBorder.

Otros de los objetivos de este proyecto son ofrecer al cliente la posibilidad de cifrar el envío de logs desde sus equipos al Client Proxy y escribir el mayor número posible de parseadores para abarcar todos los tipos de logs posibles.

Cabe mencionar que el desarrollo de la vista para redBorder Vault en el manager no entra dentro del alcance del proyecto que nos ocupa, y que esta será llevada a cabo por desarrolladores web que ya tienen experiencia en el desarrollo de este manager.

1.4 Metodología

En primer lugar, se llevará a cabo una investigación sobre logs, normativas y protocolos asociados a estos. Se investigará también las herramientas que formarán parte de este proyecto para aprender sobre su uso.

Una vez concluida la fase de investigación, se desarrollarán en un entorno de pruebas las configuraciones necesarias para que los distintos servicios trabajen conjuntamente y cumplan los objetivos propuestos para el proyecto.

Cuando se consiga un sistema funcional se pasará a integrarlo en la plataforma redBorder, de forma que los cambios realizados sean permanentes y estén presentes cada vez que se genere una nueva versión de algún elemento de la plataforma. También será necesario que cuando se produzca una modificación en la configuración de los servicios que componen nuestro sistema, los diferentes nodos que dan soporte a este actualicen también su configuración.

Por último, se trabajará en desarrollar parseadores para logs de diferentes formatos.

2 LOGS

You can have data without information, but you cannot have information without data.

- Daniel Keys Moran -

No podemos empezar este proyecto sin introducir previamente el concepto más importante de este. Se trata del concepto de log, el cuál será repetido constantemente en esta memoria por ser la gestión de estos la finalidad de este trabajo.

Durante este capítulo hablaremos sobre lo que es un log y porque son tan importantes. También hablaremos sobre su ciclo de vida: donde se generan, cómo se transmiten, dónde se almacenan, … Al acabar este capítulo el lector debe tener una visión general del mundo de los logs, que le servirá para entender el resto de la memoria.

2.1 ¿Qué es un log?

Cuando hablamos de mensaje log [1] (de ahora en adelante pasaremos a llamarlo log) nos referimos al mensaje que genera un ordenador, dispositivo de red, aplicación, etc. en respuesta a algún tipo de estímulo.

Estos estímulos pueden ser muy variados dependiendo de la fuente que genere el log. Por ejemplo, los sistemas UNIX generan un log cada vez que alguien inicia sesión o cierra sesión con cualquier usuario, los controladores de APs generan un log cuando algún equipo se conecta o desconecta de alguno de los puntos de acceso de la red, los servidores web generan un log cuando un cliente accede a alguna de las páginas web que alberga.

Podemos mencionar infinidad de ejemplos como los anteriores y en cada ejemplo la información que contiene el log es diferente y nos intenta explicar por qué se ha generado el log. Retomando los ejemplos anteriores, en el caso de los sistemas UNIX, el log nos indicaría el usuario con el que se ha iniciado o cerrado la sesión; en el caso del controlador, el log nos indica la dirección MAC de la estación que se ha conectado y a qué punto de acceso; en el caso del servidor web, el log nos indica que página ha sido accedida, desde que IP, con que navegador, etc.

2.1.1 Categorías

Podemos clasificar los logs en las siguientes categorías [2] atendiendo al nivel de atención que requieren:

- *Debug:* son generados por las aplicaciones con la finalidad de ayudar a los desarrolladores a identificar problemas y depurar el código de dicha aplicación.

- *Informational:* este tipo de logs están pensados para hacer saber a los usuarios y administradores que ha ocurrido algo sin gravedad. Por ejemplo, cuando un servidor se reinicia se genera un log. Esto en principio no tendría mayor importancia, a no ser que el reinicio se produjese en una situación en la que no debería, por ejemplo, cuando el servidor está en producción y no se está realizando mantenimiento.
- *Notice:* generados por eventos que no son frecuentes, pero no son considerados errores, por lo que su importancia, al igual que el tipo anterior, dependerá del contexto.
- *Warning:* estos logs están pensados para situaciones donde falte algo que el sistema necesita, pero la ausencia de esto no tiene impacto sobre el correcto funcionamiento del sistema. También son usados para indicar que ocurrirá un error si no se realiza alguna acción. Un ejemplo de esto sería cuando se ejecuta una aplicación con un número incorrecto de argumentos, pero, aun así, la aplicación puede ejecutarse.
- *Error:* se usan para notificar los errores ocurridos en el sistema. Por ejemplo, un sistema operativo generará un log de tipo error cuando una aplicación haya excedido su límite de almacenamiento y sus intentos de escritura están fallando.
- *Critical:* notifican algo que debe ser corregido, pero no es necesario que sea inmediatamente, por ejemplo, la pérdida de conexión con el ISP secundario mientras la conexión con el primario funciona correctamente.
- *Alert:* se usan para notificar algo que debe ser corregido inmediatamente, por ejemplo, la perdida de conexión con el ISP principal y secundario. Están relacionados también con el dominio de los dispositivos y sistemas de seguridad, que generan un log de este tipo cuando detectan conexiones maliciosas.
- *Emergency:* notifican eventos por los que el sistema ha quedado inservible. No deben ser usados por aplicaciones.

Cabe mencionar que el significado de estos niveles (exceptuando debug y emergency), son relativos a la aplicación que genera el log. Por ejemplo, no tiene la misma importancia un log de nivel warning generado por la aplicación de gestión de correo electrónico que un log del mismo nivel generado por el kernel de Linux.

2.2 ¿Por qué son importantes?

Los logs están infravalorados [3] en muchos entornos empresariales. Tanto es así que a veces los logs son ignorados hasta el punto de que sólo se acude a ellos cuando queda poca memoria en el disco duro, y muchas veces se borran sin consultarlos previamente, sin tener en cuenta que estos logs podrían contener información que nos diga porqué el disco duro está lleno.

Esto es debido a que analizar los logs no es una tarea sencilla, más bien es una tarea que en la mayoría de los casos nos supone mucho trabajo. Los logs tienen multitud de formatos y tamaño, y a veces puede ser difícil extraer información de ellos (como veremos en la sección 2.4). A esto se suma que algunos entornos se acumulan varios GB de logs a la semana o incluso al día.

A pesar de esto, debemos recordar que los logs nos proporcionan información muy importante sobre lo que está ocurriendo en nuestra red y que gracias a los servicios de gestión de logs, la terea de análisis de logs no es tan compleja.

A continuación, mencionaremos algunos de los casos de uso que nos proporcionan los logs e intentaremos ilustrarlos con ejemplos.

2.2.1 Administración de recursos

La información contenida en los logs nos permite determinar el estado de los equipos de nuestra red, los fallos que ocurren, tanto de software como de hardware, y que están haciendo los servicios que se están ejecutando en nuestros equipos.

Un ejemplo de esto lo encontramos cuando queremos ver si un equipo conectado a la red está funcionando correctamente o está bloqueado o apagado. Una forma de averiguarlo es haciendo uso del protocolo ICMP, pero esta técnica no es infalible. Hacer ping satisfactoriamente a un equipo conectado a la red te dice que la tarjeta de red está configurada correctamente y que está encendida, pero puede ser que el sistema esté bloqueado.

Una forma más segura de comprobar esto sería mirar logs como los que se presentan en el siguiente ejemplo:

Ejemplo 2-1 *[4]*. *Mensajes logs generados periódicamente por el demonio Syslog de UNIX.*

```
Jul 1 16:04:53 windbag -- MARK --
Jul 1 16:24:53 windbag -- MARK --
Jul 1 16:44:53 windbag -- MARK --
```

Estos logs son mensajes de estado generados periódicamente por el demonio Syslog de los sistemas UNIX. Esto nos asegura que el sistema está operando lo suficiente como para que el demonio Syslog escriba estos logs.

2.2.2 Detección de intrusos

Los dispositivos de seguridad de red como IPS, IDS o Firewalls generan logs que reflejan los eventos relacionados con la detección de intrusos, pero no solo estos dispositivos generan logs útiles para esta tarea.

Por ejemplo, el siguiente log es generado por el servicio ssh de un equipo cualquiera conectado a la red:

Ejemplo 2-2. *Mensaje log generado por el servicio ssh.*

```
Feb 23 11:41:37 redborder.proxy: sshd[45983]: Failed password for
ilegal user admin from 10.0.30.138 port 37960 ssh2
```

Este log muestra un intento fallido de acceso al sistema con el usuario “admin”. En el log se indica que el usuario es ilegal debido a que no se corresponde con ningún usuario existente en el sistema. Esto podría deberse a un ataque usando un ssh-scanner, el cual intenta acceder al sistema usando un diccionario de pares usuario-contraseña que se usan comúnmente. Si además de este log, encontráramos varios logs de este tipo en un corto intervalo temporal, la evidencia del ataque es clara.

Podemos poner otro ejemplo de como los logs nos ayudan con la detección de intrusos:

Ejemplo 2-3. *Mensaje log generado al crear una nueva cuenta de usuario en un sistema UNIX.*

```
Mar 15 16:17:21 redborder.proxy: adduser[36471]: new user:
name=mysql, uid=0, gid=0, home=/home/mysql, Shell=/bin/bash
```

Este log se genera cuando se crea una cuenta nueva en un sistema UNIX. La cuenta tiene nombre de usuario “mysql” pero el identificador de usuario es 0 (identificador del usuario root). Este tipo de ataques tratan de crear cuentas que pueden usar más tarde para acceder al sistema y para ello usan nombres de usuario que puedan pasar desapercibidos ya que el administrador puede pensar que el usuario ha sido creado por algún servicio. Pero lo que interesa al intruso es mantener sus privilegios de administrador, por eso el nuevo usuario tiene el mismo identificador que el usuario root.

2.2.3 Solución de problemas

Los logs son también una gran ayuda para solucionar los problemas de nuestras aplicaciones [5]. Diagnosticar problemas de servicios que no están conectados a una terminal y, por tanto, no pueden mostrar un mensaje de error, sería una tarea muy difícil sin examinar los archivos de logs.

Como ejemplo para ilustrar esto, supongamos que detectamos que nuestro servidor web Apache no muestra correctamente nuestra página web. Si miramos en el archivo de logs de Apache (/var/log/apache2/error.log) quizás encontremos logs como los siguientes:

Ejemplo 2-4. *Logs de error de Apache.*

```
  [10-Jan-2016 12:35:46 America/Los_Angeles] PHP Parse error: syntax
error, unexpected '}' in /home/web/public_html/index.php on line 30
  [10-Jan-2016 12:36:13 America/Los_Angeles] PHP Parse error: syntax
error, unexpected ':', expecting ',' or ';' in
/home/web/public_html/index.php on line 74
```

Estos logs nos informan claramente de que nuestro código PHP tiene errores de sintaxis en la línea del fichero que se especifica. Teniendo en cuenta esto, simplemente tendríamos que abrir nuestro fichero PHP y solucionar los errores.

2.2.4 Análisis forense

El análisis forense es el proceso consistente en construir una imagen de lo que ha ocurrido una vez ha concluido un evento. Para construir esta imagen, es crítica la credibilidad de la información usada. Los logs pueden ser una parte esencial en el análisis forense.

Una vez guardados en la memoria, los logs no son alterados durante el transcurso del uso normal del sistema, por lo que pueden constituir una fuente de información más fiable que otros datos del sistema que son más susceptibles a ser alterados o corrompidos.

Además, normalmente los logs guardan una marca temporal que indica el momento en el que se generó. Gracias a esto, podemos crear una secuencia lógica de eventos ordenados en el tiempo.

Por si fuese poco, los logs presentan la ventaja de que pueden ser enviados a un servidor central de logs (mediante un servicio de gestión de logs). Esto nos proporciona una fuente de evidencia externa, y que puede suponer una fuente más fiable en caso de que se ponga en duda la integridad de la información de la fuente original (por ejemplo, si un intruso ha modificado o borrado los logs originales).

2.3 Tipos, formatos y sintaxis

La sintaxis y el formato de un log define como se genera, transporta, almacena y analiza. En el caso más sencillo, cada log podría tratarse como una cadena de texto, y el interesado en consultar esos logs podría llevar a cabo una búsqueda de texto en los archivos de logs. Sin embargo, para el análisis automático de logs, es necesario que el productor y el consumidor de eventos se pongan de acuerdo en el formato y la sintaxis de los logs.

En este proyecto nos centraremos en el formato llamado Syslog[2], que será objeto de estudio en el siguiente capítulo. Este formato de logs se le puede atribuir el tipo ASCII, ya que los logs están formados por este tipo de caracteres y se almacenan en ficheros que pueden abrirse con un simple editor de textos. A continuación, podemos ver un ejemplo del formato syslog:

Ejemplo 2-5. *Log en formato Syslog.*

```
  <165>1 2003-10-11T22:14:15.003Z mymachine.example.com eventslog -
ID47 - An application event log entry...
```

A simple vista, podemos ver que el log del ejemplo anterior nos indica el momento en el que fue generado (2003-10-11T22:14:15.003Z), el equipo (mymachine.example.com) y la aplicación (eventslog) que lo generó, y un mensaje ("An application event log entry..."). Como podemos observar, el log es fácilmente entendible por un humano, siempre que el mensaje sea claro.

Servidores web, equipos de red, y la mayoría de las aplicaciones de todas las plataformas almacenan sus logs en ficheros de texto que pueden leerse fácilmente, no obstante, debemos hacer aquí una distinción entre formatos legibles por humanos y formatos basados en caracteres ASCII.

[2] El término "syslog" se usa comúnmente para referirnos al protocolo de transporte de logs y al formato de logs.

Por ejemplo, existen aplicaciones que generan sus log en formato XML. El formato XML, aunque está compuesto de caracteres legibles para los humanos, puede resultar difícil de entender cuando nos encontramos con numerosas etiquetas anidadas.

Ejemplo 2-6. *Log en formato IDMEF [6].*

```
<?xml version="1.0" encoding="UTF-8"?>

   <idmef:IDMEF-Message xmlns:idmef="http://iana.org/idmef" version="1.0">
     <idmef:Alert messageid="abc123456789">
       <idmef:Analyzer analyzerid="hq-dmz-analyzer01">
         <idmef:Node category="dns">
           <idmef:location>Headquarters DMZ Network</idmef:location>
           <idmef:name>analyzer01.example.com</idmef:name>
         </idmef:Node>
       </idmef:Analyzer>
       <idmef:CreateTime ntpstamp="0xbc723b45.0xef449129">
         2000-03-09T10:01:25.93464-05:00
       </idmef:CreateTime>
       <idmef:Source ident="a1b2c3d4">
         <idmef:Node ident="a1b2c3d4-001" category="dns">
           <idmef:name>badguy.example.net</idmef:name>
           <idmef:Address ident="a1b2c3d4-002" category="ipv4-net-mask">
             <idmef:address>192.0.2.50</idmef:address>
             <idmef:netmask>255.255.255.255</idmef:netmask>
           </idmef:Address>
         </idmef:Node>
       </idmef:Source>
       <idmef:Target ident="d1c2b3a4">
         <idmef:Node ident="d1c2b3a4-001" category="dns">
           <idmef:Address category="ipv4-addr-hex">
             <idmef:address>0xde796f70</idmef:address>
           </idmef:Address>
         </idmef:Node>
       </idmef:Target>
       <idmef:Classification text="Teardrop detected">
         <idmef:Reference origin="bugtraqid">
           <idmef:name>124</idmef:name>
           <idmef:url>http://www.securityfocus.com/bid/124</idmef:url>
         </idmef:Reference>
       </idmef:Classification>
     </idmef:Alert>
</idmef:IDMEF-Message>
```

Si comparamos el ejemplo anterior (log en formato XML) con el ejemplo 2.5 (log en formato syslog), enseguida nos daremos cuenta de que el log del ejemplo 2.6 presenta una dificultad mucho mayor para su comprensión por parte de un humano. En cambio, este último log presenta la ventaja de facilitar el procesamiento por parte de un ordenador.

Además de formatos basados en texto, también existen formatos binarios. El ejemplo más común de este formato es "Windows Event Log", el cual para poder ser entendido necesita convertirse previamente a un formato legible por humanos. Esto se realiza mediante la herramienta "Visor de eventos" de Windows, como se puede ver en la figura 2-1.

Aunque los logs basados en texto son más simples de entender para los humanos, existen razones por la que a veces se elige un formato binario: rendimiento y espacio.

Los log binarios necesitan menos capacidad de procesamiento para ser parseados, y tienen campos y tipos de datos claramente definidos, lo que permite un análisis más eficiente. En cambio, un parseador de logs basados en caracteres ASCII tiene que procesar más datos, y casi siempre se basan en coincidencia de patrones para extraer la información útil. Además, los logs binarios ocupan normalmente menos espacio en memoria (y pueden ser comprimidos), así que necesitan menos capacidad de procesamiento para darles formato y escribirlos en memoria.

Figura 2-1. Visor de eventos de Windows.

Otra importante distinción entre los tipos de logs es si el formato es abierto o propietario. Si el formato es abierto, está documentado en algún documento estándar (ISO, ANSI) o documento de referencia (RFC). Un formato propietario puede o no estar documentado y normalmente suelen usarlos un fabricante sólo en sus equipos.

En la tabla 2-1 podemos encontrar un resumen de los diferentes aspectos que hemos tratado en este punto sobre los diferentes tipos de logs.

Tabla 2-1. Resumen tipos de logs.

	XML	Syslog	Fichero de texto	Binario
Modo de consumo	Procesamiento por ordenador.	Procesamiento manual en su mayoría.	Sólo procesamiento manual.	Sólo procesamiento por ordenador.
Caso de uso	Seguridad.	Logging operacional y depuración.	Depuración.	Alto rendimiento.
Ejemplo	Cisco IPS	Mayoría de routers y switches.	Depurar aplicaciones.	Firewall Checkpoint.
Uso recomendado	Se usa cuando un conjunto de información estructurada necesita ser transferida a un consumidor para analizarla.	Se suele usar añadiéndole estructuras tipo clave-valor para simplificar el análisis automático.	Se suele usar añadiéndole estructuras tipo clave-valor para simplificar el análisis automático.	Suele usarse sólo en casos de necesidad de rendimiento muy alto.
Desventajas	Rendimiento bajo y mensajes log demasiado grandes.	La falta de estructura dificulta el análisis automático.	Es el caso más complejo y costoso para análisis automático.	No son legibles por humanos sin una herramienta para convertir los binarios en texto.

2.3.1 Sintaxis

Todos los logs, sea cual sea su formato, tienen también una sintaxis. La sintaxis se refiere a como estructuramos lo que queremos decir y no a que palabras especificas usar. Cada log tiene una estructura propia, pero, basándonos en los formatos de logs más usados, podemos establecer un conjunto de campos que suelen aparecer en todos los logs:

- Fecha y hora (timestamp) en la que se generó el mensaje. Esto nos permite ordenar los mensajes cronológicamente y analizar cómo se sucedieron los eventos.
- Equipo que generó el log. Esto es muy útil cuando tenemos varios equipos enviando logs a un servidor central, ya que nos permite distinguir a que equipo pertenece cada log.
- Aplicación o componente que generó el log. Nos permite distinguir dentro de cada equipo, que parte de este generó el mensaje.
- Rigor (severity), prioridad, o importancia del log.
- Si el log está relacionado con cualquier actividad que implique el uso de usuario, indicará el nombre de usuario usado para el acceso.

La sintaxis es importante a la hora de realizar cualquier tipo de análisis automático de información. Esto se debe a que antes de realizar un análisis, necesitamos dividir el log en partes de acuerdo a su sintaxis. Para ello, existen multitud de herramientas relacionadas con el análisis de logs, en las que se han incluido patrones que definen la sintaxis de varios tipos de logs. Un ejemplo de ello es la herramienta que usaremos en este proyecto, Rsyslog, que cuenta con patrones predefinidos para reconocer la sintaxis del formato syslog (tanto el obsoleto definido en la RFC 3164, como el actual definido en la RFC 5424).

2.3.2 Contenido

Además de los campos mencionados en la subsección anterior, los logs suelen incluir un campo de contenido. En este campo es donde reside la verdadera información que se intenta transmitir con el log.

En algunos casos, como en el protocolo syslog, el contenido del mensaje es de estructura libre a diferencia del resto de campos. Esto dificulta el análisis automático ya que el contenido del log puede ser muy variado.

Los logs, pueden contener información sobre la actividad de usuario: quién ha accedido al sistema, qué estuvieron haciendo, cuando cerraron su sesión, etc. Los logs pueden darte información sobre que están estropeadas o van a estropearse, como los errores en el disco duro. Los logs pueden informarnos sobre qué aplicaciones se están ejecutando correctamente y darnos información sobre utilización de recursos y rendimiento. Los logs también pueden contener información sobre cambios de esta, encendidos, apagados, reinicios, etc. Y los logs pueden a veces contener información sobre intentos de intrusión en nuestro sistema.

Estos son sólo algunos ejemplos (los más comunes) de la variedad de información que puede contener un log. A continuación, basándonos en estos ejemplos vamos a realizar una clasificación de logs dependiendo del contenido de estos:

- *Gestión de cambios*: Registros de cambios en el sistema, cambios de componentes, actualizaciones, cambios en las cuentas, y todo aquello que pueda ser objeto de un proceso de gestión de cambios. Este tipo de logs normalmente pueden ser divididos en registros de añadidos, borrados, actualizaciones y modificaciones.
- *Autenticación y autorización*: Registros de decisiones de autenticación y autorización (por ejemplo, un intento de acceso correcto o fallido al sistema) y acceso al sistema de usuarios privilegiados. Este tipo constituye el caso más común de log relacionado con la seguridad, y debe ser generado por cada sistema.
- *Acceso a datos:* Registro de acceso a componentes de aplicación y datos (archivos y tablas en base de datos). Este tipo de logs está estrechamente relacionado con el anterior, y busca mantener la privacidad de los usuarios informando de quién accede a los datos en cada momento.
- *Gestión de amenazas:* Registros de alertas de intrusos u otras actividades que violan las políticas de seguridad. Estos logs son producidos por dispositivos de red con funcionalidades de seguridad

(firewall, IPS, …).

- *Gestión de rendimiento y capacidad:* Registros relacionados con la gestión de rendimiento y capacidad del sistema, incluyendo memoria y utilización de la capacidad de procesamiento y otros recursos finitos.
- *Gestión de continuidad y disponibilidad del negocio:* Registros relacionados con copias de seguridad, redundancia y utilización de las características de continuidad del negocio. Los logs generados por los servidores cuando están siendo apagados o encendidos también entran dentro de este tipo.
- *Errores y fallos:* Registros de errores de sistemas que pueden o no demandar una acción por parte del administrador del equipo.
- *Depuración:* Registros de mensajes de depuración generados por las aplicaciones.

2.4 Ciclo de vida de los logs

En esta sección describiremos los estados por los que normalmente pasa un log: donde se genera, como se transmite, como se almacena y algunas acciones como filtrar y normalizar que son comunes antes del análisis.

2.4.1 Generación de logs

Poco podemos decir sobre la generación de logs que no hayamos dicho ya. Los logs provienen de muchas fuentes de las cuales las más comunes son las que se enumeran a continuación:

- Sistemas Unix y Windows.
- Aplicaciones.
- Routers.
- Switches.
- Sistemas de seguridad: firewalls, IPS, IDS...
- Puntos de acceso Wi-Fi.
- Controladores de puntos de acceso.
- Servidores VPN.

Esta lista es sólo una pequeña parte de una lista interminable de fuentes de logs. El punto clave aquí es que todo dispositivo, ordenador y aplicación de nuestra red tiene la capacidad de generar logs. Teniendo en cuenta esta gran variedad existente de fuentes podemos apreciar ahora porqué existen tantos formatos de logs distintos y la dificultad que esto puede conllevar.

Las fuentes de logs pueden clasificarse en dos categorías: fuentes push-based y fuentes pull-based.

En las fuentes push-based, el dispositivo o aplicación envía un log al disco local o a través de la red. Si se envía a través de la red, debemos tener un colector de logs preparado para recibir este mensaje. Las tres fuentes push-based principales son Syslog, SNMP y Windows Event Log, que se corresponden con protocolos sobre los que se transmite el mensaje.

En la categoría pull-based, una aplicación recibe el mensaje log desde su fuente original. Este método siempre se basa en un modelo cliente-servidor. Muchos de los sistemas que operan de esta manera, almacenan los logs recibidos en una base de datos en lugar de almacenarlos en ficheros de texto.

2.4.2 Transmisión y recolección

La transmisión y recolección de logs es conceptualmente muy simple. Un ordenador o dispositivo implementa un subsistema de logging a través del cual puede generar un mensaje cada vez que lo crea necesario. Por otro lado debemos tener un lugar donde enviar y recolectar los logs. Este lugar suele llamarse servidor de logs. Un servidor de logs es un equipo, generalmente un sistema Unix o un servidor Windows, donde enviamos y

recolectamos logs con el objetivo de centralizar la información. Utilizar un servidor de logs tiene bastantes ventajas, entre las que podemos destacar las siguientes:

- Nos permite almacenar logs provenientes de múltiples localizaciones en un lugar centralizado.
- Nos ofrece la posibilidad de almacenar una copia de seguridad de nuestros logs que puede sernos útil en caso de fallo del sistema que los generó.
- Nos facilita el análisis de toda la información que contienen todos los logs de nuestra red.

La forma más común de enviar mensajes log es usando el protocolo Syslog, del cual hablaremos en el siguiente capítulo. El protocolo Syslog es un estándar para el intercambio de mensajes log. Se usa principalmente en sistemas Unix, pero también podemos encontrarlo en sistemas Windows y otras plataformas no basadas en Unix. Básicamente, lo que nos dice este protocolo es que hay un cliente que envía mensajes Syslog (con el formato especificado por el protocolo) a un servidor usando UDP o TCP si se desea garantizar la entrega de los mensajes. La tarea principal del servidor es recibir los mensajes log en formato Syslog y almacenarlos en el disco local, donde podrán ser analizados.

Syslog no es el único mecanismo para la transmisión y recolección de logs. Por ejemplo, Microsoft ha implementado su propio sistema de logging para Windows, Windows Event Log. A pesar de esto, existen aplicaciones que se ejecutan sobre Windows Event Log y que permiten convertir los logs en formato Windows a Syslog, para poder enviarse a un servidor Syslog, que son mucho más comunes.

Otro ejemplo de mecanismo para la transmisión de logs, es el protocolo SNMP. Este es un protocolo para la gestión de dispositivos de red. El protocolo se basa en dos conceptos: trap y pulling. Un trap es una forma de mensaje log que un dispositivo o equipo emite cuando algo ha ocurrido. Este trap se manda a un equipo gestor que sería el análogo a un servidor de logs en Syslog. El concepto de polling se refiere a la posibilidad de realizar una petición desde un gestor a un dispositivo, siempre preguntando por variables predefinidas.

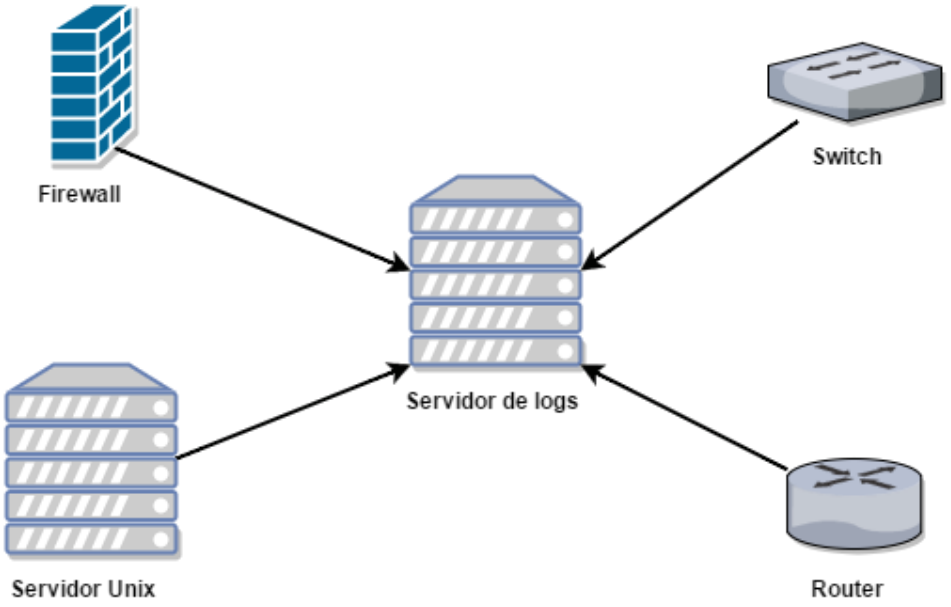


Figura 2-2. Recolección tradicional de logs en un servidor.

La figura 2-2 muestra el esquema que hemos estado dibujando hasta ahora, donde todos los equipos de nuestra red envían sus logs a un servidor centralizado. Sin embargo, este esquema es bueno si la red es pequeña o no está extendida geográficamente, pero si este no es el caso, quizás tengamos la necesidad de tener un conjunto de servidores distribuidos. Para evitar tener en varios servidores nuestros logs, la solución pasa por usar un equipo intermedio entre las fuentes de logs y el servidor.

La figura 2-3 muestra un esquema alternativo en el que se muestra un colector de logs reenviando estos a un servidor de logs central. En caso de que el colector de logs este en una localización diferente al servidor, la conexión entre ambos suele encriptarse para evitar que alguien no deseado pueda acceder a los logs.

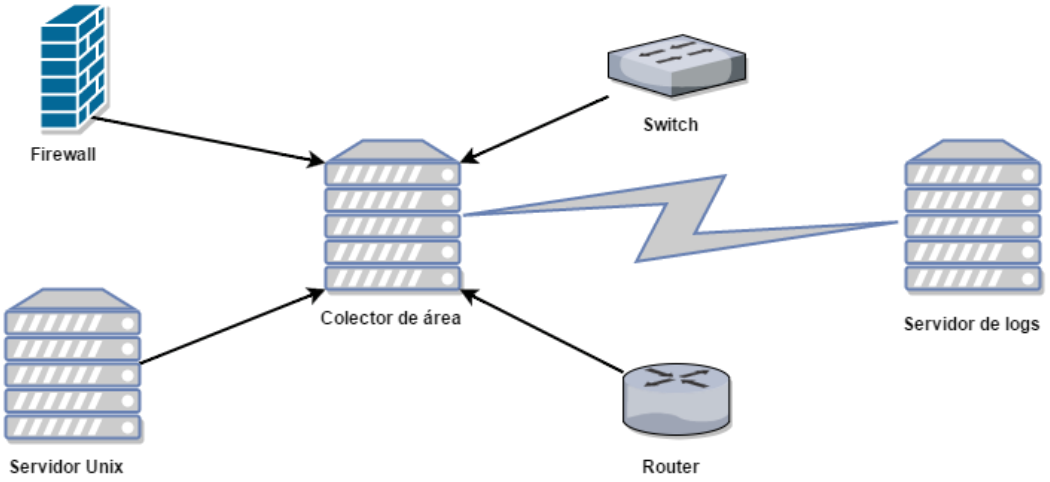


Figura 2-3. Recolección distribuida de logs en un servidor.

Esta configuración tiene varias ventajas:

- *Redundancia:* Los logs son almacenados en varios lugares (colector de área y servidor central).
- *Almacenamiento y reenvío:* El colector de área puede almacenar los logs en caso de perder la conexión con el servidor central y enviarlos cuando la conexión se restablezca.
- *Autenticación:* Tenemos la posibilidad de que el colector de área se autentique a los emisores para que estos tengan certeza de a quien están enviando sus logs y, además, los emisores también pueden autenticarse para que el colector sólo acepte logs de fuentes de confianza.
- *Privacidad:* Muchas veces se da el caso de que un dispositivo o sistema que envía logs al servidor no tiene la capacidad de encriptar sus mensajes. Con esta configuración esto no es un problema ya que, si el colector se encuentra en la misma red local que ese dispositivo, este puede enviar los logs sin encriptar al colector, el cual es capaz de reenviar estos logs al servidor de forma segura.

Como se describe en el capítulo 5, para este proyecto se sigue un esquema como el último adaptado a la estructura de la plataforma redBorder.

2.4.3 Filtrado y normalización

A continuación, pasaremos a explicar los conceptos de filtrado y normalización, pero antes debemos mencionar que tanto el filtrado como la normalización pueden llevarse a cabo en cualquiera de los actores implicados en el esquema expuesto en la figura 2-3: puede llevarse a cabo en la fuente de logs, si esta lo permite, en el colector de área y en el servidor central.

Una vez hemos configurado nuestros equipos para generar y enviar mensajes log, el siguiente paso es filtrar y normalizar estos mensajes. El filtrado se encarga de incluir o excluir mensajes log basándose en el contenido de estos mensajes. Algunas fuentes de logs tienen la capacidad de realizar este filtrado y en otros casos será necesario efectuarlo en otro agente como por ejemplo el colector de área, que se basará en reglas definidas por el usuario para realizar el filtrado. No existen recomendaciones sobre que logs filtrar y cuáles no, eso depende de las necesidades en cada momento. Por ejemplo, mientras se está realizando un mantenimiento sería recomendable filtrar los logs generados al reiniciar el sistema que está en proceso de mantenimiento, ya que no nos aportan información.

La forma óptima de realizar el filtrado sería filtrar en cada fuente los logs no deseados, ya que de esta forma los logs que no queremos enviar al servidor no viajan por la red ni consumen los recursos de esta. En caso de que algunas de las fuentes no soporten filtrado de logs, sería recomendable realizarlo en el colector de área, ya que es el elemento más cercano a la fuente.

Normalización es el acto de coger mensajes log con distintos formatos y convertirlos a un formato común. Cuando tenemos los logs provenientes de diferentes fuentes en un formato común, es mucho más fácil manipular y analizar los datos. Cabe destacar que la normalización es independiente de las fuentes y el protocolo usado para enviar los logs. Es recomendable realizar la normalización en los colectores de área para

repartir la carga entre varios nodos en lugar de concentrarla en el servidor central.

A continuación, vamos a poner un ejemplo para comprender mejor el concepto de normalización. Como ya se ha mencionado en secciones anteriores, algunos formatos de mensajes log indican la prioridad del mensaje entre los datos que transportan y otros formatos no la indican, o simplemente tienen una escala de prioridades distinta. Necesitamos normalizar las distintas escalas de prioridades. Una estrategia básica es coger la prioridad de los logs recibidos y mapearlas a una escala común. Un ejemplo de escala común podría ser el siguiente:

- *Prioridad baja:* Eventos meramente informacionales que no necesitan ninguna acción por parte del administrador.
- *Prioridad media:* Eventos que necesitan una acción por parte del administrador, pero no de forma inmediata. Por ejemplo, los sistemas IPS tienen la capacidad de bloquear tráfico cuando detectan contenido malicioso. Si el IPS bloquea un flujo de tráfico importante el administrador tendrá que averiguar porque ha ocurrido esto, sin embargo, si el flujo de datos es secundario el administrador puede decidir tomar acciones más tarde.
- *Prioridad alta:* Eventos que necesitan una acción inmediata. Por ejemplo, un servidor que se reinicia fuera del periodo de mantenimiento, un IPS alertando de un posible robo de información, un router informando de que se ha perdido la conexión con el IPS principal, etc.

La normalización se complementa con el concepto de parseo. Parsear consiste en analizar una entrada de texto a fin de determinar su estructura y convertir la entrada de texto en una estructura de datos que es apropiada para ser procesada. A continuación, vamos a comentar un ejemplo más completo de normalización:

Ejemplo 2-7. *Mensaje log normalizado en formato JSON.*

```
  {
    "Prioridad":"Media",
    "Timestamp":"1454850707",
    "Fuente":"server.web",
    "Aplicación":"adduser",
    "Id proceso":"14965",
    "Tipo":"new user",
    "Nombre de usuario":"user1",
    "uid":"124",
    "gid":"136",
    "Directorio home":"/home/user1",
    "Shell":"/bin/bash",
    "Mensaje original":"Feb 7 13:11:47 server.web: adduser[14965]: new
user: name=user1, uid=124, gid=136, home=/home/user1, Shell=/bin/bash"
  }
```

En el ejemplo 2-7 vemos un log en formato JSON que ha pasado por un proceso de parseo y normalización. El mensaje original se puede encontrar en el último campo del JSON. Esto tiene la ventaja de preservar el mensaje original sin ningún cambio. Cabe mencionar las siguientes observaciones:

- El mensaje ha pasado de ser una cadena de texto a presentarse en formato JSON, siendo mucho más fácil de manejar por las diferentes herramientas de análisis automático.
- La prioridad no aparece en el mensaje original por lo que se ha decidido poner prioridad media (basándonos en la escala definida anteriormente) por criterio propio.
- Se ha cambiado el formato de la fecha al formato “Unix Time Stamp”. Este formato representa el número de segundos trascurridos desde 00:00:00 UTC del 1 de enero de 1970 hasta el momento en que se crea la marca temporal.

Si todos los mensajes log que nos llegan los parseamos y almacenamos o reenviamos en este u otro formato, podemos decir que estamos normalizando los logs, así como sus campos (timestamp, prioridad, …).

3 EL PROTOCOLO SYSLOG

There were 5 Exabytes of information created between the dawn of civilization through 2003, but that much information is now created every 2 days.

- Eric Schmidt -

En el capítulo anterior ya hemos visto algunos ejemplos de logs que siguen el formato syslog. En esta sección trataremos más a fondo este protocolo que es clave en nuestro proyecto. Syslog es un estándar para logging que define una arquitectura basada en capas que permite separar el contenido del mensaje del transporte de este y facilita la fácil extensión de cualquier capa. También describe un formato para los mensajes log que permite a las aplicaciones generar logs estructurados para facilitar el parseo de estos. No describe ninguna forma de almacenar los mensajes syslog.

Syslog es un estándar muy popular y ampliamente soportado en la mayoría de sistemas operativos y frameworks. Está oficialmente soportado por casi todas las versiones de los sitemas Linux, Unix y Mac OS. Windows no trae soporte nativo para Syslog, pero sí que es soportado usando alguna de las muchas librerías desarrolladas por otras empresas.

Este protocolo fue definido inicialmente en la RFC 3164 [7] y posteriormente en la RFC 5424 [8] que será la referencia a seguir para este capítulo.

3.1 Definiciones y arquitectura

En esta sección se explicará la arquitectura de capas definida por el protocolo Syslog. Para ello primero explicaremos los conceptos[3] asociados a esta arquitectura y posteriormente pasaremos a colocarlos dentro de la arquitectura de capas.

En el estándar Syslog se definen varios conceptos asociados a la función que se realiza dentro del contexto syslog:

- *Originator:* entidad que genera el contenido que posteriormente será enviado en un mensaje syslog.
- *Collector:* entidad que recibe los mensajes syslog y no los reenvía, sino que los almacena para un análisis posterior. Cuando esta función se asigna a una máquina en particular, esta suele recibir el nombre de servidor syslog[4].
- *Relay:* entidad que reenvía mensajes syslog provenientes de otros relays o originators hacia uno o

[3] Para referirnos a estos conceptos utilizaremos los nombres en inglés especificados en la RFC 5424, ya que algunos de estos nombres no cuentan con una traducción coherente en este conexto.

[4] Es el equivalente al equipo que se llamó servidor de logs en el capítulo anterior

varios collectors u otros realys[5].

- *Transport sender:* entidad que pasa los mensajes syslog generados por un originator a un protocolo de transporte específico para su envío.
- *Transport receiver:* entidad que toma los mensajes syslog de un protocolo de transporte específico y lo entrega a una entidad con la función de collector o relay.

Cada una de estas funciones se encuadran dentro de una de las tres capas conceptuales que define Syslog:

- *Contenido syslog:* información contenida en un mensaje syslog.
- *Capa de aplicación syslog:* controla la generación, interpretación, encaminamiento y
- almacenamiento de los mensajes syslog.
- *Capa de transporte syslog:* manda y recibe los mensajes syslog a través de la red.

En la figura 3-1 se muestra una representación gráfica de la arquitectura de capas definida por el protocolo syslog. Al dividir las distintas funciones en varias capas, esta estructura simplifica el desarrollo de aplicaciones que soporten este protocolo y su posterior actualización.

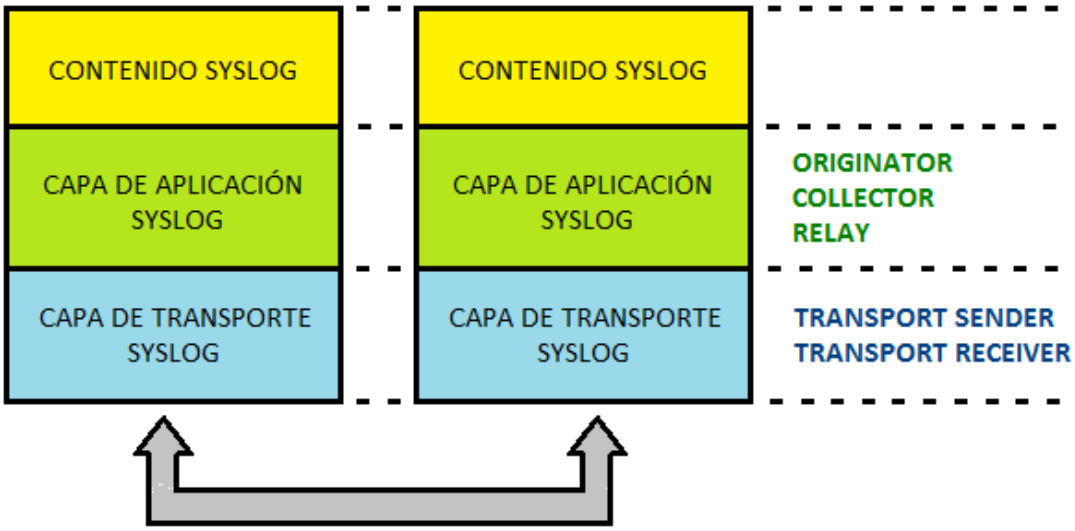


Figura 3-1. Arquitectura de capas Syslog.

A continuación, vamos a comentar algunas consideraciones a tener en cuenta basándonos en los ejemplos de posibles escenarios Syslog que aparecen en la RFC 5424 y que se han representado en la figura 3-2:

- Los equipos que tienen la funcionalidad de originator envían mensajes syslog a relays o collectors sin conocimiento de que función desempeña la entidad a la que está mandando el mensaje. Además, deben poder ser configurados para enviar el mismo mensaje syslog a varios relays o collectors.
- Los equipos que cumplen la funcionalidad de relay además de reenviar los mensajes syslog que reciben pueden procesarlo (parsear, normalizar, ...) antes de reenviarlos e incluso pueden generar sus propios mensajes syslog y enviarlos a un relay posterior o un collector. Además, no es obligatorio que reenvíen todos los mensajes que reciben, sino que pueden realizar un filtrado para seleccionar los mensajes syslog que envían al collector o el próximo relay.
- Las funciones de procesado mencionadas para el relay también pueden ser realizadas en el collector antes de almacenar los logs que recibe.

[5] Aunque en este contexto tiene una finalidad más general, el equipo que llamamos colector de área en el capítulo anterior tenía esta función.

- Aunque hasta ahora hemos hablado de las funcionalidades de originator, relay y collector como procesos que se ejecutan en equipos separados, es posible que estas tres funcionalidades residan en el mismo equipo.

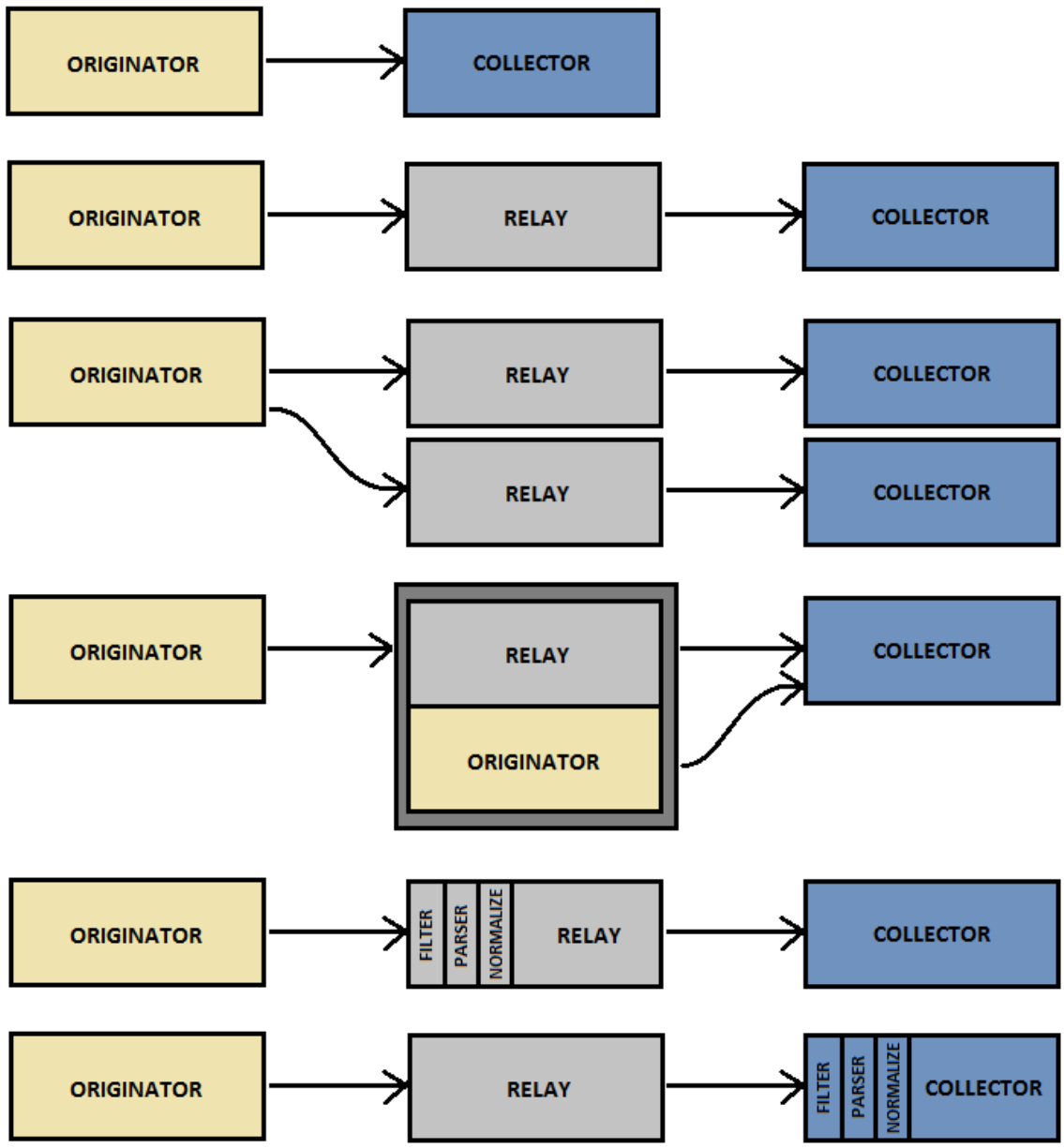


Figura 3-2. Algunos posibles escenarios Syslog.

Otro de los puntos que se aclara en la RFC 5424 sobre el protocolo Syslog es que no proporciona asentimientos de los mensajes que se han enviado. Sin embargo, esta funcionalidad puede estar soportada por el protocolo de transporte que se use para enviar los mensajes syslog. El transporte de mensajes syslog se define en otros documentos:

- *RFC 5426* [9]*:* Transporte de mensajes syslog sobre el protocolo UDP.
    - Las implementaciones de este protocolo deben poder recibir mensajes syslog en el puerto 514 UDP si actúa como receiver o enviarlo a este puerto en caso de actuar como originator o relay. Los mensajes pueden enviarse desde cualquier puerto.
    - Cada datagrama UDP puede contener sólo un mensaje syslog.
    - La IP del datagrama UDP no debe interpretarse como el identificador del equipo que generó

el mensaje syslog, ya que este datagrama puede provenir de un relay. El propio mensaje syslog contiene el identificador del equipo que lo genero.

- El checksum de los datagramas UDP debe estar activado.

- *RFC 6587* [10]*:* Transporte de mensajes syslog sobre el protocolo TCP.
  - Este protocolo no tiene un puerto TCP estándar asignado. En la práctica los administradores de red suelen coger o el puerto 514 TCP, que está actualmente asignado al protocolo Shell, o un múltiplo de 1000 añadiéndole 514.
  - Una vez el emisor de mensajes syslog ha inicializado una sesión con el receptor, durante la fase de intercambio de mensajes, cada segmento TCP contiene un mensaje syslog.
  - Al igual que en el caso de UDP, la IP del segmento IP no debe considerarse el identificador del equipo que generó el mensaje.
- *RFC 5425* [11]*:* Transporte de mensajes syslog sobre TLS.
  - Se define como puerto TCP por defecto el 6514.
  - Los equipos con la función de transport sender y transport receiver deben implementar autenticación basada en certificado. Esto consiste en validar el certificado y verificar que el otro equipo tiene la clave privada correspondiente.
  - Es posible incluir varios mensajes syslog en un sólo paquete TLS o transferir un mensaje syslog en múltiples paquetes TLS.
  - De nuevo, la IP de los paquetes TLS no identifica el generador del mensaje syslog.

Aunque en el documento que describe el protocolo Syslog no se especifica que deba usarse obligatoriamente ningún protocolo de transporte, sí que se obliga a que todas las implementaciones de este protocolo soporten[6] un protocolo de transporte basado en TLS como se especifica en la RFC 5425. Además de obligar a soportarlo recomienda el uso de este protocolo siempre que sea posible. También se recomienda el soporte del protocolo UDP para mantener la compatibilidad, ya que en la versión anterior del protocolo syslog se usaba este protocolo.

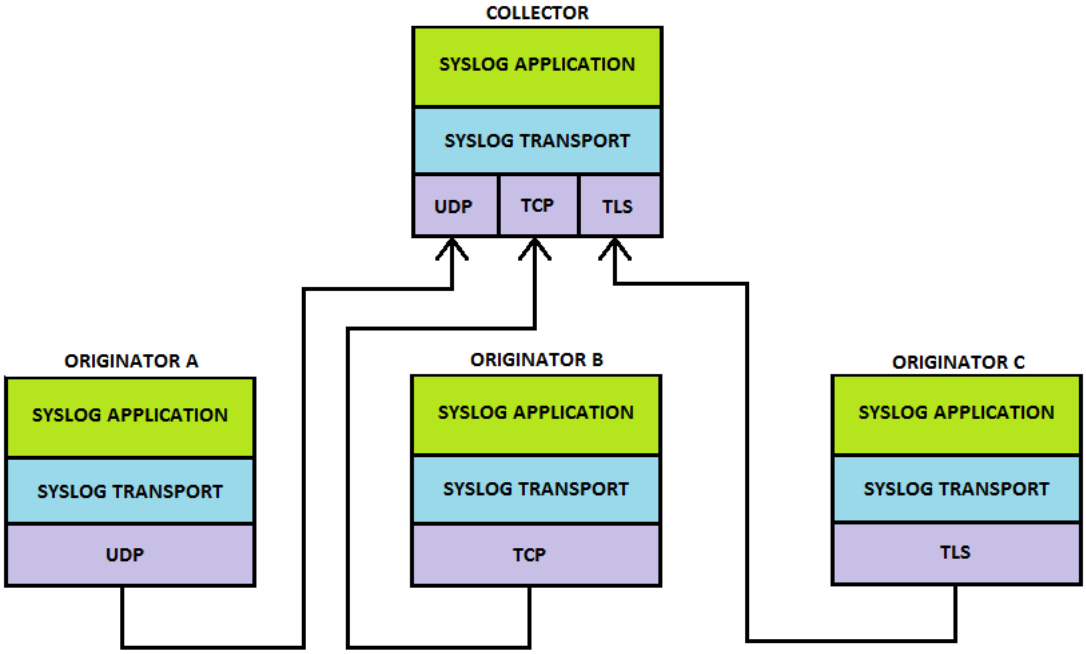


Figura 3-3. Escenario Syslog multiprotocolo.

[6] Que soporte TLS no quiere decir que no se pueda usar otro, pero si debe darse la opción de usar este.

Por último, aclarar que el uso de un protocolo de transporte no implica que los otros protocolos de transporte no puedan ser usados simultáneamente. Por ejemplo, la figura 3-3 muestra tres originators que envían mensajes syslog a un único collector. Cada uno de ellos envían sus mensajes haciendo uso de un protocolo de transporte distinto: el originator A usa el protocolo UDP, el originator B usa el protocolo TCP y el originator C hace uso del protocolo TLS. Para que esto sea posible, el collector debe tener la capacidad de aceptar mensajes transportados por los tres protocolos mencionados.

3.2 Formato del mensaje Syslog

En esta sección hablaremos sobre el formato para los mensajes Syslog descrito en la RFC 5424. El mensaje Syslog se divide en tres partes: cabecera, datos estructurados y mensaje. Cada uno de estas partes tiene una serie de campos que describiremos en el desarrollo de esta sección. También hablaremos de la longitud del mensaje y mostraremos algunos ejemplos extraídos de la RFC que nos servirán para tener una visión completa del formato.

La estructura general del formato es la siguiente:

<PRI>VERSION TIMESTAMP HOSTNAME APP-NAME PROCID MSGID STRUCTURED-DATA MSG

3.2.1 Longitud del mensaje

No existe un límite en la longitud de los mensajes syslog, sino que están establecidos por el protocolo de transporte usado para enviar los mensajes. Cada protocolo establece la longitud máxima de los mensajes que transportan, pero este máximo debe ser de al menos 480 Bytes.

La RFC define varios niveles en cuanto a la longitud máxima permitida por los protocolos de transporte usados. Como requisito absoluto impone que todos los protocolos de transporte usados para enviar mensajes syslog deben aceptar mensajes de a hasta 480 octetos, pero en caso de no existir ninguna circunstancia particular que lo impida, deberán soportar mensajes de hasta 2048 octetos de longitud y se deja como opcional el soporte de mensajes de longitudes mayores a esta cifra.

3.2.2 Cabecera

A continuación, hablaremos de la cabecera de los mensajes Syslog. Esta cabecera constituye la información común a todos los mensajes Syslog y está diseñada para proporcionar interoperabilidad con la anterior especificación del protocolo Syslog.

La cabecera está compuesta por los campos que se enumeran a continuación (con los mismos identificadores que se establecen en la RFC 5424) y que explicaremos en los siguientes apartados de esta subsección: PRI, VERSION, TIMESTAMP, HOSTNAME, APP-NAME, PROCID, MSGID.

3.2.2.1 PRI

Este campo representa la prioridad del mensaje. Está formado por un número de máximo tres dígitos rodeado por los caracteres ‘menor que’ (<) y ‘mayor que’ (>) en la forma “<PRI>”. Dicho número representa el valor de la prioridad del mensaje y se calcula a partir de otros dos valores: el servicio que genera el mensaje (facility) y la severidad (severity) del mensaje.

Estos dos valores no tienen una asignación obligatoria, pero existe una recomendación de uso en la RFC 5424 que expondremos a continuación.

Los valores para los servicios (facility) deben estar comprendidos entre 0 y 23 (ambos inclusive) y las asignaciones recomendadas por los autores del estándar se muestran en la tabla 3-1.

Además de indicar el servicio que genera el mensaje, debemos indicar la importancia que tiene el mensaje. Esto se hace mediante el valor de severidad (severity) del mensaje que debe ser un valor comprendido entre 0 y7 (ambos inclusive). La relación entre sus valores numéricos y su descripción se presentan en la tabla 3-2.

Tabla 3-1. Códigos de recursos en el protocolo Syslog.

Código numérico	Servicio (facility)
0	Kernel
1	Mensajes de nivel de usuario
2	Sistema de correo
3	Demonios del sistema
4	Mensajes de seguridad/autorización
5	Mensajes generados internamente por el demonio syslog
6	Subsistema de impresión
7	Subsistema de noticias sobre la red
8	Subsistema UUCP (Unix to Unix Copy Protocol)
9	Demonio de reloj
10	Mensajes de seguridad/autorización
11	Demonio de FTP (File Transfer Protocol)
12	Subsistema de NTP (Network Time Protocol)
13	Inspección de logs
14	Alerta sobre logs
15	Demonio de reloj
16	Uso local 0
17	Uso local 1
18	Uso local 2
19	Uso local 3
20	Uso local 4
21	Uso local 5
22	Uso local 6
23	Uso local 7

Tabla 3-2. Códigos de severidad en el protocolo Syslog.

Código numérico	Severidad (severity)
0	Emergencia: el sistema está inutilizable
1	Alerta: se necesita una acción inmediata
2	Crítico: condiciones críticas
3	Error: condiciones de error
4	Peligro: condiciones de peligro
5	Aviso: condiciones significativas no peligrosas
6	Información: mensajes informativos
7	Depuración: mensajes de bajo nivel

Para calcular el valor de la prioridad, debemos multiplicar por 8 el valor asociado al servicio (facility) que genera el mensaje syslog y sumarle el valor de la severidad (severity) de este. Cuanto menor sea el valor calculado para el campo PRI, mayor es la prioridad del mensaje, teniendo el valor 0 como el más prioritario y el valor 191 como el menos prioritario.

3.2.2.2 VERSION

Este campo denota la versión de la especificación del protocolo Syslog que se está usando en el mensaje. El valor de este campo se incrementa cada vez que se cambia algún aspecto del formato del mensaje Syslog. Los posibles valores de este campo están controlados por IANA en un registro llamado "Syslog Version Values" [12] y en el caso de la versión actual el valor correspondiente es 1.

3.2.2.3 TIMESTAMP

Este campo representa una marca temporal que indica cuando fue creado el mensaje y se incluye con un formato derivado de la RFC 3339 [13]. Una representación genérica de este formato podría ser la mostrada en la figura 3-4.

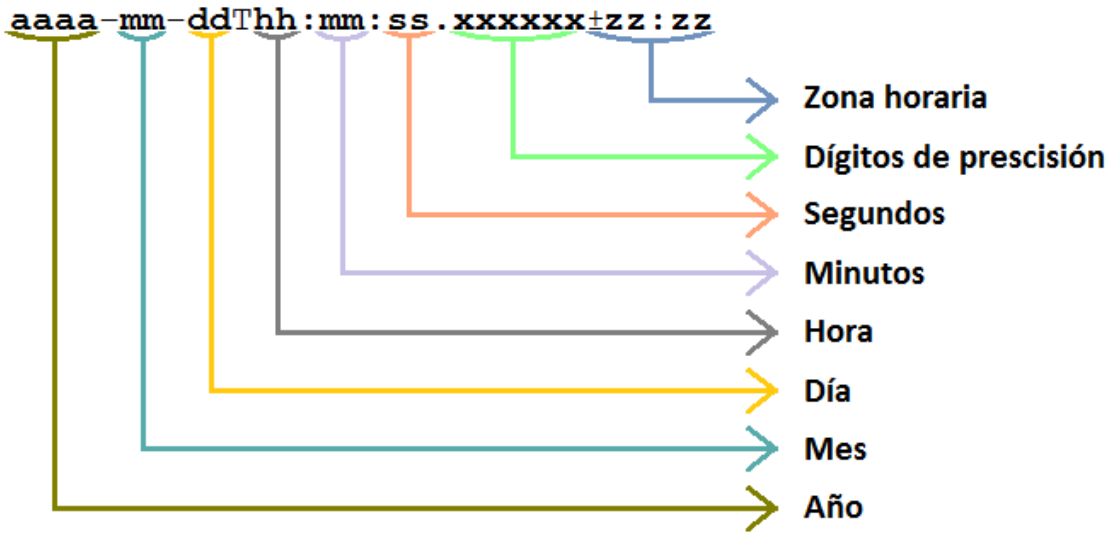


Figura 3-4. Formato de timestamp Syslog.

Algunas consideraciones sobre este formato se enumeran a continuación:

- El carácter 'T' se usa como separador entre la fecha y la hora y, es obligatorio usar el carácter en mayúscula.
- Los dígitos de precisión son opcionales y como máximo se pueden usar 6 dígitos.
- La zona horaria se especifica según el estándar UTC. Si la marca temporal está expresada en UTC en lugar de usar una hora local, esto se especificará usando el carácter 'Z' en lugar de "±zz:zz" como se indica en el formato descrito en la imagen 3-4.

3.2.2.4 HOSTNAME

El campo hostname identifica el equipo que generó originalmente el mensaje syslog. Para este campo se debe seguir el formato FQDN (Fully Qualified Domain Name) especificado en la RFC 1034 [14].

En la práctica, no todos los equipos pueden proporcionar un FQDN, por lo que el estándar Syslog permite sustituir este por valores de otro tipo. Para ello, se indica una lista con un orden de preferencia para los tipos de datos que hay que respetar, de forma que, si un equipo en particular tiene acceso a varios de los tipos, debe escoger para el campo hostname el tipo de mayor preferencia.

El orden de preferencia es el siguiente:

1. FQDN.
2. Dirección IP estática.
3. Hostname del equipo.
4. Dirección IP dinámica.
5. Valor nulo (representado por el carácter '-').

3.2.2.5 APP-NAME

El campo APP_NAME es una cadena de texto que identifica al dispositivo o aplicación que genera el mensaje Syslog. Está pensado para usarlo a la hora de filtrar los mensajes syslog en un relay o collector.

En caso de que el equipo que genera el mensaje Syslog sea incapaz de proporcionar esta información, ya sea a causa de las políticas locales o porque la información no está disponible, se debe sustituir por el valor nulo.

3.2.2.6 PROCID

El campo PROCID no dispone de un significado específico, sino que depende de la implementación. Se usa a menudo para proporcionar el nombre o identificador de proceso asociado al sistema Syslog. De esta forma, es posible detectar discontinuidades en el envío de mensajes si nos percatamos de un cambio en el identificador de proceso del sistema Syslog.

Otro uso común para este campo es identificar a que grupo de mensajes pertenece el mensaje actual.

3.2.2.7 MSGID

El campo MSGID es una cadena de texto que debe identificar el tipo de mensaje. Por ejemplo, un firewall podría usar el MSGID "TCPIN" para tráfico TCP entrante y el MSGID "TCPOUT" para tráfico TCP saliente. Este campo también está pensado para usarlo en filtros. Si no se dispone de un valor para este campo, debe ser sustituido por el valor nulo.

3.2.3 Datos estructurados

Los datos estructurados son un mecanismo provisto por el estándar Syslog para expresar información en un formato definido, fácilmente parseable e interpretable. Esta parte del mensaje Syslog puede contener ninguno (en cuyo caso se sustituirá por el valor nulo), uno o varios elementos de datos estructurados a los que se les da el nombre "SD_ELEMENT". Un SD-ELEMENT consiste en un nombre (SD-ID) y un número indefinido de pares nombre-valor (SD-PARAM), todo entre corchetes ("[…]").

Como regla general, un collector puede ignorar los SD-ELEMENTS mal formados y un relay debe reenviar los datos estructurados mal formados sin ninguna modificación.

El SD-ID debe identificar el tipo y el propósito de un SD-ELEMENT. El SD-ID debe ser único dentro de un mismo mensaje. En lo que respecta al formato del SD-ID, tenemos dos:

- Los nombres que no contienen el carácter '@' están reservados para ser asignados por el IETF, por lo que un SD-ID de este tipo no es válido si no está registrado previamente en IANA. En la tabla 3-3 se muestran los SD-IDs registrados inicialmente, aunque a día de hoy ya se han registrado algunos más.

Tabla 3-3. SD-IDs registrados en IANA.

SD-ID	Descripción
timeQuality	Se usa para proporcionar información acerca del nivel de certeza que tiene el campo TIMESTAMP del mensaje.
origin	Se usa para proporcionar información del equipo generador del mensaje de una forma más sencilla de analizar automáticamente.
meta	Se usa para proporcionar meta-información sobre el mensaje.

- SD-ID definidos por el usuario con el formato nombre@<número de empresa privado>. La parte que precede al carácter '@' es una cadena de caracteres sin formato definido. El número que sigue al carácter '@' debe estar registrado por IANA.

Por su parte, cada SD-PARAM tiene el formato PARAM-NAME="PARAM-VALUE". Podemos especificar los PARAM-NAME que queramos dentro de un SD-ELEMENT, a excepción de aquellos cuyo SD-ID contiene el carácter '@' que están controlados por IANA. También podemos repetir PARAM-NAME en dos SD-ELEMENTS distintos.

Dentro del PARAM-VALUE debemos preceder del carácter ' \ ' los caracteres ' \ ', ' " ' y '] ', con la finalidad de evitar errores de parseo.

En la tabla 3-3 mostramos los SD-ID registrados inicialmente en IANA. Ahora, en las tablas 3-4, 3-5 y 3-6 describimos brevemente los SD-PARAMs registrados en IANA para cada uno de estos SD-ID:

Tabla 3-4. SD-PARAMs especificados para el SD-ID "timeQuality".

PARAM-NAME	Descripción
tzKnown	Se usa para indicar si el generador del mensaje conoce su zona horaria. Si es conocida el valor debe ser 1 y 0 si no lo es.
isSynced	Se usa para indicar si el generador del mensaje está sincronizado con alguna fuente de tiempo externa y fiable. El valor debe ser 1 en caso afirmativo y 0 en caso contrario.
syncAccuracy	Se usa para indicar el nivel de precisión que tiene el generador del mensaje en su sincronización de tiempo. Debe especificar el máximo número de microsegundos que su reloj puede desincronizarse entre intervalos de sincronización.

Tabla 3-5. SD-PARAMs especificados para el SD-ID "origin".

PARAM-NAME	Descripción
ip	Se usa para indicar la dirección IP del equipo que generó el mensaje. Puede ser útil cuando en el campo HOSTNAME de la cabecera se da un valor distinto a la IP como, por ejemplo, un FQDN.
enterpriseId	Se usa para especificar un número de empresa privado (SMI Network Management Private Enterprise Code) gestionado por IANA.
software	Identifica el software que generó el mensaje, que no tiene por qué coincidir con el campo APP-NAME de la cabecera.
swVersion	Identifica la versión del software que generó el mensaje y está especificado en el parámetro anterior.

Tabla 3-6. SD-PARAMs especificados para el SD-ID "meta".

PARAM-NAME	Descripción
sequenceId	Se usa para monitorizar la secuencia en la que se envían los mensajes provenientes de un equipo.
sysUpTime	Se usa para indicar el tiempo desde el último reinicio del sistema que genera los mensajes.
language	Se usa para indicar el lenguaje que se usa en el mensaje.

3.2.4 Mensaje

La última parte que define el formato del mensaje Syslog contiene un mensaje en formato libre que proporciona la información esencial sobre el evento que originó el log.

Al ser de formato libre, es a parte más compleja a la hora de parsear los logs provenientes de diferentes fuentes.

3.2.5 Ejemplos

En esta subsección expondremos algunos de los ejemplos de mensajes syslog con formato correcto que aparecen en la RFC 5424 para aclarar las posibles dudas del lector acerca del formato que hemos comentado durante el transcurso de esta sección.

El primer ejemplo (ejemplo 3-1) muestra un mensaje Syslog que carece de datos estructurados. Como podemos comprobar fijándonos en el valor del campo VERSION, el formato se corresponde con el comentado durante el desarrollo de esta sección.

Ejemplo 3-1. *Mesaje Syslog sin datos estructurados*

```
 <34>1 2003-10-11T22:14:15.003Z mymachine.example.com su - ID47 -
BOM'su root' failed for lonvick on /dev/pts/8
```

El valor del campo PRI es 34 y podemos obtener a partir de él, el valor del servicio (facility) y el valor de la severidad (severity). En este caso, el valor de la severidad es 2 (3 bits menos significativos del campo PRI) y el valor del servicio es 4 (el resto de bits del campo PRI). Recordemos que un valor de 4 en el servicio significa "mensajes de seguridad/autorización" y un valor de 2 para la severidad significa "mensaje de condiciones críticas". Como vemos esto está correctamente aplicado a este log ya que se corresponde con un intento fallido de acceder al usuario root del sistema, por lo que podemos estar ante un intento de intrusión.

Como podemos apreciar por el campo TIMESTAMP el mensaje se generó a las 10:14:15pm del día 11 de octubre de 2003. El timestamp está expresado en la zona horaria de referencia y no en la zona horaria local, como se pude apreciar por el carácter 'Z' que aparece al final del mismo.

En el campo HOSTNAME el equipo que generó el mensaje se identifica mediante su propio hostname como "mymachine.example.com" por lo que deducimos que no tendría acceso a un FQDN ni a una dirección IP estática.

La aplicación que generó el log es "su", como podemos observar en el campo APP-NAME de la cabecera del ejemplo. El campo PROCID es desconocido por lo que en su lugar aparece el valor nulo (carácter '-'), en cambio el campo MSGID sí que se especifica, obteniendo el valor "ID47".

Una vez acaba la cabecera Syslog deberíamos encontrarnos con los datos estructurados, pero en este caso el equipo que generó el log no introdujo ningún SD-ELEMENT, por lo que los datos estructurados son sustituidos por el carácter nulo.

El mensaje es "'su root' failed for lonvick on /dev/pts/8", pero comienza con una marca ("BOM") que indica que el mensaje tiene codificación UTF-8. Esta marca realmente son caracteres no imprimibles, pero en los ejemplos de la RFC 5424 aparecen representados de esta forma.

El siguiente ejemplo (ejemplo 3-2) es algo más completo que el anterior ya que no falta ninguno de los campos que componen el formato:

Ejemplo 3-2. *Mesaje Syslog con datos estructurados*

```
 <165>1 2003-10-11T22:14:15.003-07:00 192.0.2.1 evntslog 8710 ID47
[exampleSDID@32473 iut="3" eventSource="Application"
eventID="1011"][examplePriority@32473 class="high"] BOMAn application
event log entry...
```

Como se puede apreciar, en esta ocasión el timestamp no está expresado en la zona horaria de referencia, sino

que se expresa en una zona horaria local (-07:00), Otra de las diferencias es que en este ejemplo en el campo HOSTNAME aparece una dirección IP y el campo PROCID no es desconocido, sino que aparece un número (8710) que seguramente se corresponda con el identificador de proceso de la implementación Syslog que envió el mensaje.

La parte más interesante de este ejemplo es que existen datos estructurados. Concretamente tenemos dos SD-ELEMENTs identificados por los SD-IDs “exampleSDID@32473” y “examplePriority@32473”. Cada uno de los SD-ELEMENTS está delimitado por corchetes ([…]), no existiendo ningún espacio ni otro carácter que separe ambos SD-ELEMENTs.

El primero de los SD-ELEMENTs contiene 3 SD-PARAMs con sus respectivos PARAM-NAMEs y PARAM-VALUEs y el segundo de los SD-ELEMENTs sólo contiene un SD-PARAM.

Por último, comentar que, aunque no se ha puesto en ningún ejemplo, es posible generar un log en formato Syslog sin el mensaje (la última de las partes) ya que esta es opcional al igual que los datos estructurados. Esto puede parecer que no tiene mucho sentido, pero sería totalmente razonable si la información que queremos transmitir con el log la introducimos en los datos estructurados íntegramente, lo que facilitaría en gran medida el procesado posterior del mensaje.

4 SERVICIOS Y HERRAMIENTAS USADOS

I think, fundamentally, open source does tend to be more stable software. It's the right way to do things.

- Linus Torvalds -

En este capítulo hablaremos de los servicios que forman parte de la arquitectura desarrollada para este proyecto y, de las herramientas de las que nos hemos servido para cualquier necesidad que se haya planteado a lo largo del transcurso de este proyecto, ya sea para testear, generar certificados, etc…

Nos limitaremos a describir la funcionalidad de cada herramienta o servicio, sin entrar en detalles de qué papel ha jugado en nuestro proyecto o que configuración se ha usado, ya que esto es objeto de estudio de capítulos posteriores.

Tampoco hablaremos sobre herramientas secundarias que, aunque hayan sido útiles para el proyecto, no han tenido una implicación directa en sus resultados. Algunos ejemplos de estas herramientas son: Sublime como editor de texto, Git como herramienta de control de versiones, VirtualBox como software de virtualización, tcpdump/wireshark como analizador de tráfico, etc…

4.1 Rsyslog

Rsyslog es un software open source para la administración de logs que implementa el protocolo Syslog. Actualmente se distribuye como sistema de logging por defecto en la mayoría de sistemas Linux, ya que representa una gran mejora respecto del obsoleto syslogd.

Ofrece un alto rendimiento que lo capacita para funcionar en entornos empresariales con alta afluencia de datos, pero al mismo tiempo es un software liviano que puede funcionar en equipos sin muchos recursos. Tampoco escasea en características de seguridad, ofreciendo al usuario multitud de opciones para configurar el grado de seguridad deseado.

Gracias a su diseño modular, Rsyslog es capaz de recibir logs de una amplia variedad de fuentes (amplia en número y en tipo), transformarlos y enviar los resultados a diferentes destinos. La figura 4-1 (obtenida del sitio web de rsyslog [15]) muestra algunas de las fuentes y destinos que pueden conectarse usando Rsyslog.

La capacidad para recibir datos de estas fuentes o enviarlos a estos destinos, se implementa normalmente mediante un módulo, de modo que si, por ejemplo, queremos enviar logs a elasticsearch deberemos cargar su módulo previamente en la configuración de Rsyslog.

Existen cinco tipos de módulos para Rsyslog: módulos de entrada (input modules), módulos de parseo (parser modules), módulos de modificación de mensaje (message modification modules), módulos de generación de cadenas de caracteres (string generator modules) y módulos de salida (output modules).

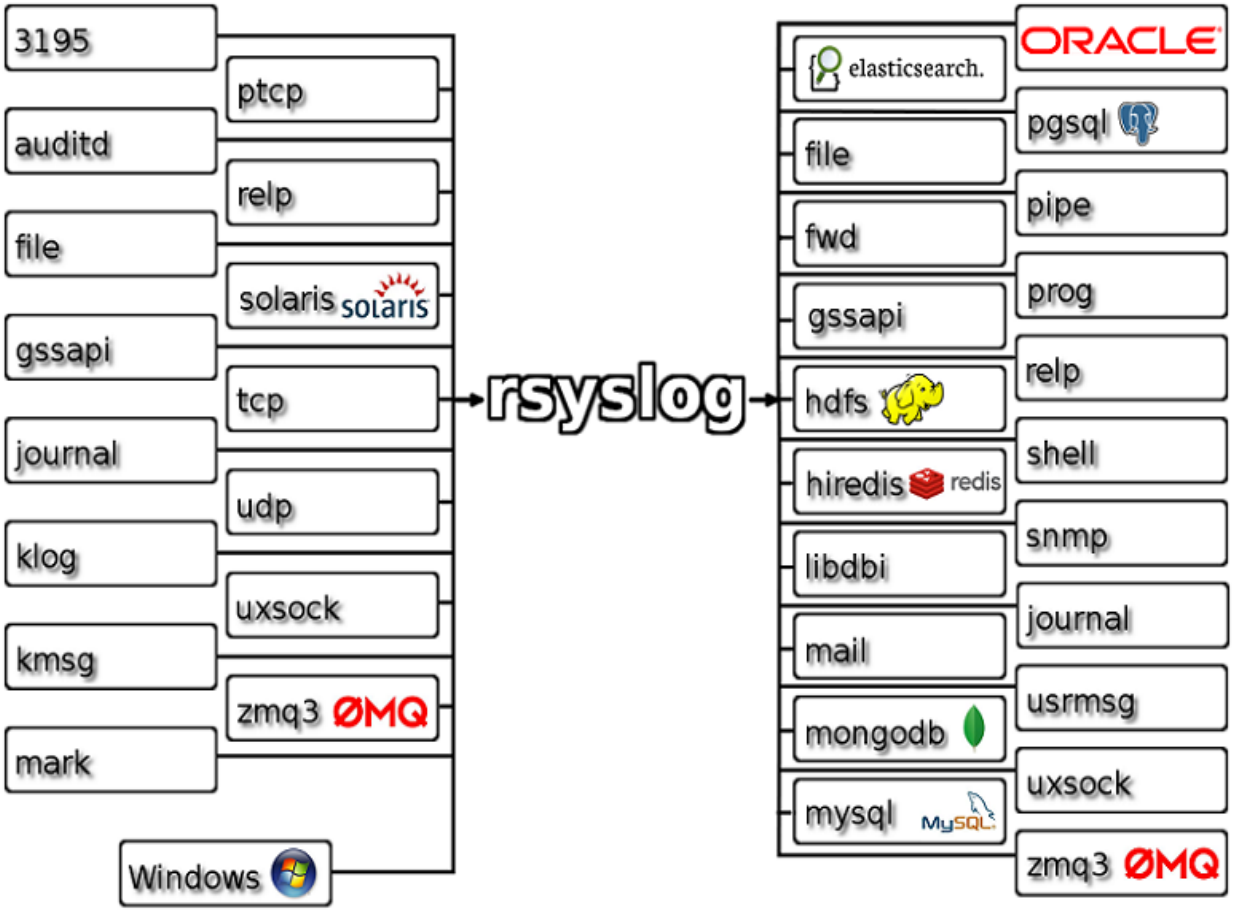


Figura 4-1. Algunos de las posibles fuentes y destinos de Rsyslog.

Cada tipo de módulo actuará accediendo o modificando el mensaje siguiendo un orden establecido por el flujo de trabajo de Rsyslog [16]. Este flujo de trabajo se muestra en la figura 4-2 y como puede verse en esta, los logs se reciben siempre a través de módulos de entrada y a continuación, pasan 1 o más veces por módulos de parseo, que se encargan de generar una representación interna del mensaje, y por módulos de modificación de mensajes. La representación en memoria del log se pasa por uno o varios módulos de salida, generando cada uno de estos su salida correspondiente.

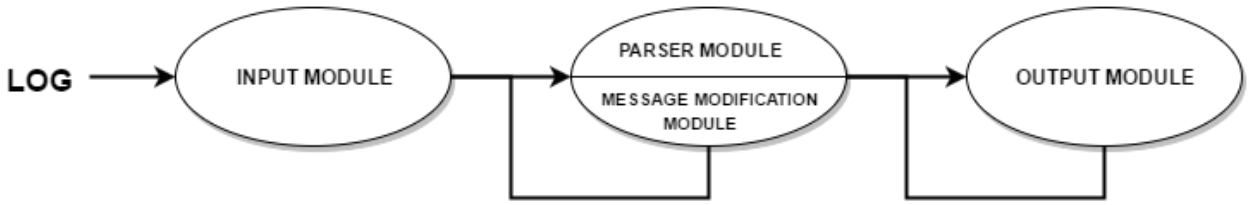


Figura 4-2. Workflow de Rsyslog.

Los módulos de generación de cadenas de caracteres no aparecen en el diagrama anterior debido a que no son una parte obligatoria de Rsyslog. En caso de ser usados, operarían al mismo tiempo que los módulos de salida, ya que son ejecutados durante la creación de las plantillas de salida[7].

En las siguientes subsecciones hablaremos de la funcionalidad de los módulos que se han usado durante el desarrollo de este proyecto.

[7] Las plantillas de salida definen el formato que tendrá el log al enviarlo mediante el módulo de salida.

4.1.1 imuxsock

Proporciona a Rsyslog la capacidad de recibir mensajes syslog a través de los sockects locales UNIX. Las llamadas al sistema syslog(3)[8] utilizan este mecanismo para hacer llegar los mensajes syslog al servicio Rsyslog.

4.1.2 imfile

Proporciona a Rsyslog la capacidad de leer un archivo estándar de texto y procesarlo para convertirlo en mensajes logs. Se considera archivo de texto estándar a un archivo que sólo contiene caracteres imprimibles y con líneas delimitadas por el carácter de nueva línea "LF".

El archivo de texto se lee línea a línea y cada una de esas líneas se para por filtros para determinar qué acciones deben ser llevadas a cabo. Las líneas en blanco son ignoradas. Conforme se van escribiendo nuevas líneas en el fichero, se van leyendo y procesando.

4.1.3 imudp

Proporciona a Rsyslog la capacidad de recibir mensajes Syslog mediante el protocolo UDP. Es posible configurar Rsyslog para escuchar en varios puertos UDP simultáneamente.

4.1.4 imtcp

Proporciona a Rsyslog la capacidad de recibir mensajes syslog mediante el protocolo TCP. Además, es posible recibir mensajes encriptados haciendo uso del driver GTLS.

4.1.5 pmrfc3164

Usado por Rsyslog para parsear mensajes que siguen el formato Syslog tradicional definido en la RFC3164 [7]. Este módulo es utilizado por defecto cuando llega un mensaje con este formato.

4.1.6 pmrfc5424

Al igual que el módulo anterior, este es usado por defecto cuando se recibe un mensaje en el actual formato Syslog definido en la RFC5424.

4.1.7 mmjsonparse

Aunque está definido como un módulo de modificación de mensaje, este módulo es en realidad un parseador de mensajes en formato JSON que siguen la especificación CEE/lumberjack. Si el mensaje comienza con la "CEE cookie"[9], la estructura JSON que viene a continuación es parseada y cargada en memoria.

La estructura JSON debe estar bien formada y a continuación de esta no puede existir ningún tipo de mensaje. Si alguna de estas condiciones no se cumple, la estructura JSON no será parseada.

4.1.8 mmpstrucdata

Este módulo al igual que el anterior es en realidad un módulo de parseo. Se encarga de parsear los datos estructurados contenidos en un mensaje con formato definido en la RFC5424 (se habló de ello en la subsección 3.2.3) y cargarlos en memoria en forma de estructura JSON.

---

[8] La llamada al sistema syslog() [19] es usada por las aplicaciones para generar un mensaje syslog y enviarlo directamente al gestor de logs del sistema. Otra opción es que las aplicaciones guarden sus logs en un fichero que posteriormente es leído por el sistema gestor de logs.

[9] La llamada "CEE cookie" se corresponde con la secuencia de caracteres "@cee:"

4.1.9 mmnormalize

Este módulo nos permite hacer uso de la herramienta liblgnorm para ejecutar parseadores definidos por el usuario en ficheros de reglas mediante la sintaxis especificada en la documentación de esta herramienta. Gracias a este módulo, es posible parsear los mensajes sin estructura y cargarlos en memoria de una forma normalizada para facilitar el acceso a sus datos.

4.1.10 mmrfc5424addhmac

Este módulo modifica los mensajes con formato Syslog descrito en la RFC 5424 para añadir un nuevo SD-ELEMENT que contendrá un hash que puede ser usado para garantizar la integridad del mensaje original. Para generar el hash utiliza openssl, pudiendo usar cualquiera de las funciones hash que implementa esta herramienta.

4.1.11 omfile

Este módulo ofrece a Rsyslog la capacidad de escribir mensajes log en un fichero de texto que se encuentre en el sistema de archivos local. Cada log enviado a un archivo ocupa una línea de este. El nombre del fichero puede ser estático o generado dinámicamente según el contenido de los logs, por ejemplo, un archivo podría usar el hostname o la IP del equipo que está generando logs.

4.1.12 omfwd

Este módulo proporciona a Rsyslog la capacidad de enviar logs a través de la red usando el protocolo UDP o TCP según se indique en su configuración. Además, ofrece la posibilidad de usar el protocolo TLS para encriptar los mensajes mediante el uso del driver GTLS.

4.1.13 omkafka

Este módulo implementa un productor Kafka dando a Rsyslog la capacidad de escribir mensajes log en un topic de Kafka siguiendo una plantilla de salida definida por el usuario.

4.2 Kafka

Kafka [17] es un sistema de mensajería de tipo publicador/suscriptor desarrollado inicialmente por Linkedin y posteriormente por Apache. Sus principales características son que es un servicio de mensajería distribuido, particionado y replicado, que categoriza los mensajes en topics. Un topic no es más que una categoría a la cual se publican mensajes y de la cual se pueden consumir.

Tenemos que diferenciar entre los procesos que publican a un topic, denominados publicadores, y los procesos que se suscriben a topics, denominados consumidores. Tanto consumidores como productores se comunican con el cluster de Kafka y no directamente entre ellos como se puede apreciar en la figura 4-3.

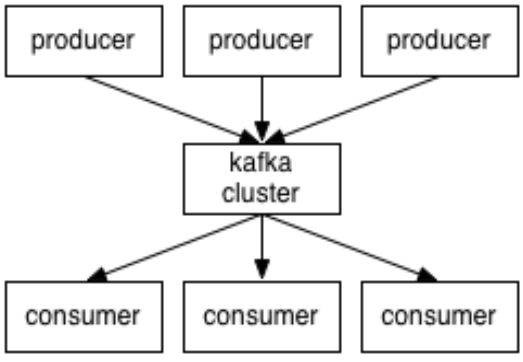


Figura 4-3. Elementos de Kafka.

Kafka funciona como un cluster compuesto por uno o más servidores, cada uno de los cuales se denomina bróker. Es importante mencionar que un mismo servidor podría correr varias instancias de Kafka por lo que en ese servidor existirían varios brokers de Kafka, es decir, se llama bróker a cada uno de los procesos de Kafka pertenecientes a un cluster, no a cada una de las máquinas del cluster.

Para cada topic, el cluster de Kafka puede dividirlo en varias particiones. Cada parición es una secuencia ordenada de mensajes, representados por un número secuencial denominado offset. El offset identifica a cada mensaje dentro de una partición. Un mensaje ha llegado a la partición antes que otro mensaje si su offset es menor que el offset del otro mensaje ya que el offset se va asignando según el orden de llegada y este orden se mantiene.

Los productores publican mensajes en el topic que elijan y son responsables de seleccionar la partición dentro del topic a la que estará destinado el mensaje.

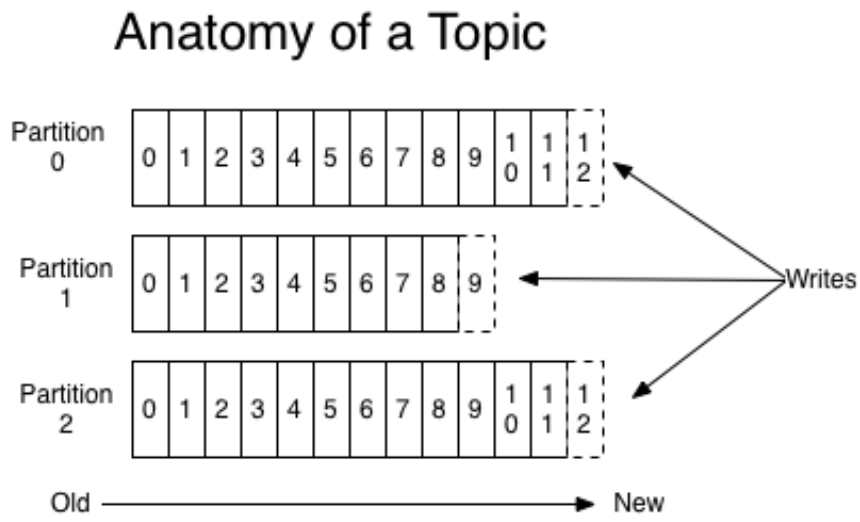


Figura 4-4. Particionado de un topic.

El cluster de Kafka mantiene todos los mensajes publicados por un periodo de tiempo configurables independientemente de si estos mensajes han sido o no consumidos.

Cada consumidor controla su propio offset. En la mayoría de los casos, un consumidor avanzará su offset linealmente conforme va leyendo mensajes de la partición, pero como la posición del offset es controlada por el consumidor, este puede decidir consumir los mensajes en otro orden distinto al orden natural.

Las particiones de un topic se distribuyen en los servidores del cluster de Kafka. Cada partición es replicada en un número configurable de brokers para ganar tolerancia a fallos. Cada partición tiene asignado un bróker que tiene la función de líder y puede tener asignados brokers que cumplen la función de seguidores del bróker líder. El bróker líder maneja todas las escrituras y lecturas a la partición mientras que los seguidores se encargan de replicar al líder para que en caso de fallo uno de ellos pueda ocupar su lugar. Cada broker puede actuar como líder para varias particiones y como seguidor para otras, de esta forma se reparte la carga dentro del cluster.

Para balancear carga entre consumidores Kafka usa el concepto de grupo de consumidores. Los consumidores se etiquetan a sí mismos con un grupo de consumidores y cada mensaje publicado en un topic es entregado a una sola instancia de consumidor dentro de cada grupo de consumidores suscritos. En la figura 4-5 podemos ver un ejemplo de un topic con 4 particiones repartidas en dos servidores y 6 consumidores repartidos en dos grupos de consumidores. Como se puede apreciar en el diagrama, cada partición envía una copia de los mensajes publicados en ellas a un consumidor de cada grupo y además se balancean los mensajes de las distintas particiones entre los consumidores de un grupo.

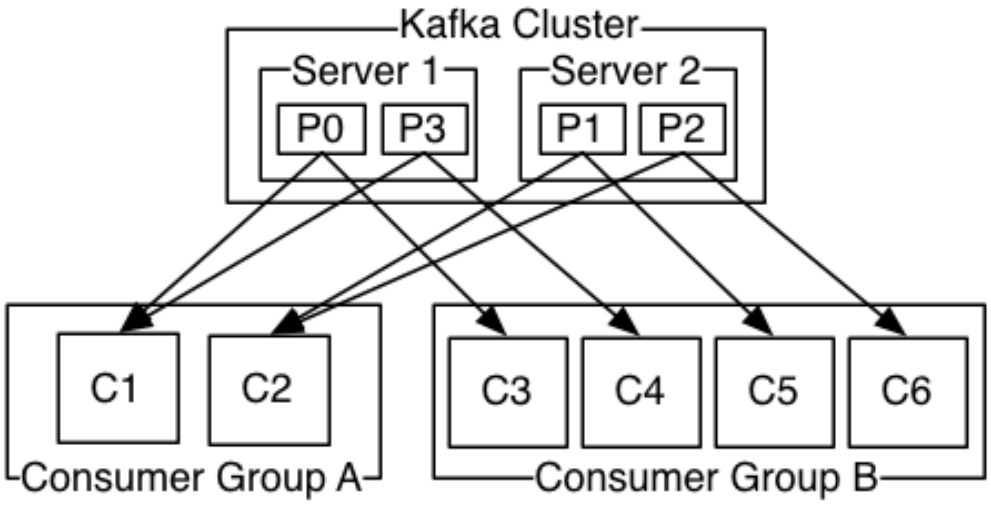


Figura 4-5. Grupos de consumidores de Kafka.

Haciendo uso de los grupos de consumidores podemos llegar a dos situaciones peculiares:

- Si todas las instancias de consumidores están en el mismo grupo de consumidores, Kafka funciona como un sistema de cola de mensajes balanceando carga entre los consumidores de ese grupo.
- Si todas las instancias de consumidores están en grupos de consumidores distintos, Kafka se comporta como un sistema publicador/suscriptor y todos los mensajes se envían a todos los consumidores.

4.3 Zookeeper

Zookeeper [18] es un servicio open source de coordinación de alto rendimiento para aplicaciones distribuidas desarrollado por Apache. Entre otras cosas, ofrece servicios de gestión de configuraciones, naming, sincronización, grupo de servicios, etc.

Zookeeper es ideal para distribuir servicios en clusters con un gran número de nodos, consiguiendo dotar al servicio de tolerancia a fallos, alta disponibilidad mediante redundancia en los nodos y alta escalabilidad. Además, permite a los desarrolladores implementar las funciones de coordinación mediante una API.

Cuando tenemos un número de nodos gestionados por Zookeeper, este categoriza los nodos en líder (o coordinador) y seguidores. Los nodos seguidores se comunican con el líder para sincronizarse, manteniendo una copia exacta del estado actual del líder, de forma que, si el nodo líder cae, Zookeeper se encarga de cambiar a otro nodo líder manteniendo la disponibilidad del sistema. Es importante precisar que las peticiones por parte de los clientes pueden ser enviadas a cualquier nodo (líder o seguidor).

Es importante comentar que Kafka usa Zookeeper para gestionar y coordinar sus brokers. De esta forma todos los productores y consumidores son notificados cada vez que ocurre un cambio en el cluster Kafka. Esto es que cada vez que se añade un nuevo bróker al cluster todos los productores que generan mensajes en el cluster de Kafka y todos los consumidores que reciben estos mensajes tienen constancia de ello al igual que cuando un bróker deja de funcionar. De este modo conseguimos que nuestro cluster de Kafka tenga una gran tolerancia a fallos. En la figura 4-6 podemos ver el esquema general de un servicio Kafka funcionando con Zookeeper.

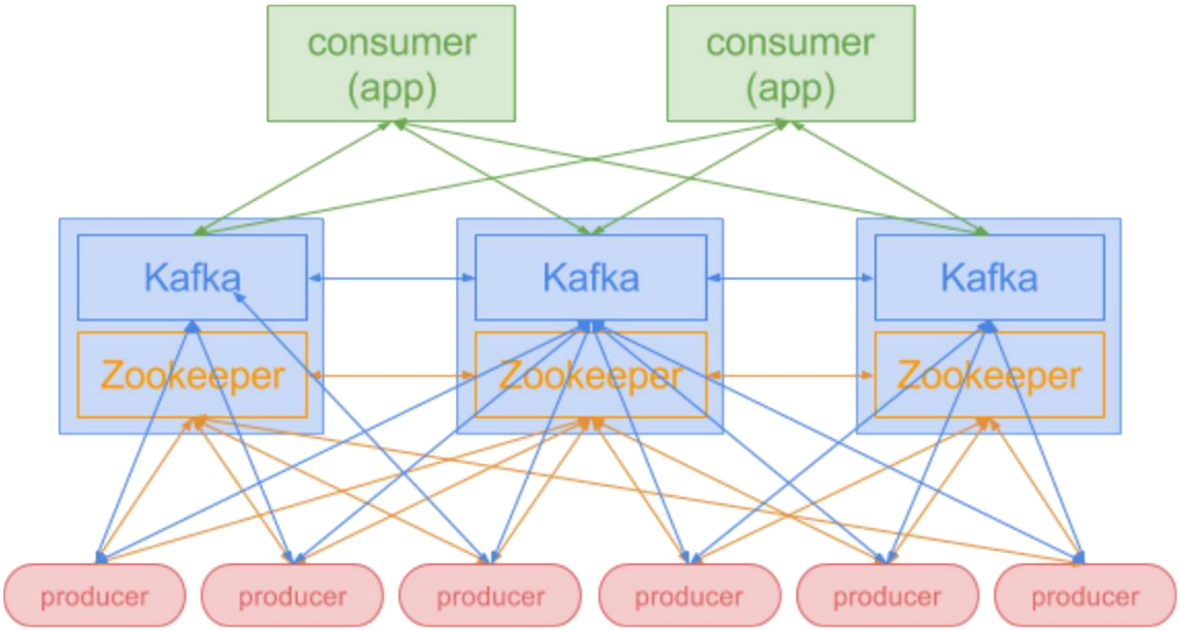


Figura 4-6. Diagrama de comunicación entre Zookeeper y Kafka.

En la figura se puede apreciar como los procesos Zookeeper se comunican entre ellos para mantener coordinados en todo momento los brokers de Kafka. Además de esto, y de comunicar a consumidores y productores el estado de los brokers, Zookeeper se encargar también de mantener el offset de cada uno de los consumidores. También ofrece la posibilidad de balancear la carga proveniente de los productores en el cluster.

4.4 Chef

Chef [19] es un framework de automatización de infraestructuras de sistemas capaz de facilitar la tarea de instalación y configuración de herramientas en cualquier equipo físico, virtual o en la nube. Chef automatiza la instalación, configuración, despliegue y gestión de las aplicaciones en los equipos de nuestra red, sin importar el tamaño de esta.

En la figura 4-7 podemos ver un esquema que relaciona los principales componentes de Chef, siendo estos la estación de trabajo (workstation), el servidor Chef (Chef Server) y los nodos (nodes).

El funcionamiento general es el siguiente: desde las estaciones de trabajo los administradores escriben los llamados cookbooks y los suben al servidor chef. Los nodos que están bajo la gestión de Chef se conectan al servidor y descargan la última versión de los cookbooks, que serán ejecutados y esto hará que cada nodo se configure según lo descrito en los cookbooks.

A continuación, describiremos más detalladamente cada uno de estos componentes:

Una **estación de trabajo (workstation)** es un equipo configurado para ejecutar varias herramientas de Chef que le permiten sincronizar chef-repos usando una herramienta de control de versiones (como git), escribir cookbooks y recipes (usando patrones y sintaxis Ruby), subir cookbooks al servidor, interactuar con el servidor Chef, interactuar con los nodos, …

Siempre que configuramos un equipo para usarlo como estación de trabajo, se nos instala el llamado Chef Development kit, que es un paquete que contiene todo lo necesario para usar Chef: chef-client, chef, Ohai, … Un ejemplo de herramienta que viene con Chef Development Kit es Knife, una herramienta de línea de comandos que nos permite interactuar con los nodos o trabajar con objetos en el servidor Chef.

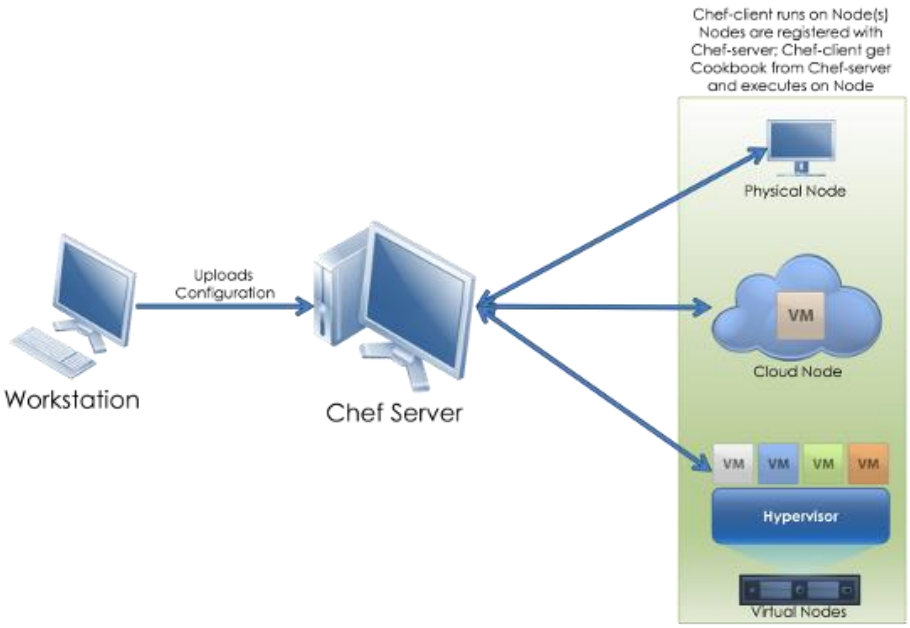


Figura 4-7. Componentes principales de Chef.

Un **cookbook** es la unidad fundamental de configuración dentro de Chef. Deben escribir en Ruby y su finalidad es definir un escenario y contener todo lo necesario para dar soporte a este escenario:

- *Recipes:* son los elementos de configuración fundamentales para Chef. Se componen de recursos, definidos usando patrones (nombre de recursos, pares nombre-valor, y acciones) y cuando es necesario, código de ayuda escrito en Ruby. Se usan para especificar los recursos a usar y el orden en que deben ser aplicados.
- *Attributes:* dentro de un cookbook podemos definir atributos que pueden ser usados para que la configuración varíe ligeramente entre un nodo u otro usando atributos propios del nodo como la ip, el hostname, etc.
- *Templates:* son plantillas escritas con Embedded Ruby (ERB) usadas para generar dinámicamente archivos de texto estático. Pueden contener expresiones y declaraciones Ruby, y son una buena forma de gestionar los archivos de configuración.

Además de poder escribir nuestros propios cookbooks la empresa Chef pone al servicio de los usuarios una comunidad llamada Chef Supermarket donde poder subir tus cookbooks o descargar los del resto de usuarios para usarlos como base para el tuyo propio.

El **servidor Chef,** almacena los cookbooks, las políticas aplicadas a los nodos, y metadatos que describen cada uno de los nodos que están siendo gestionados por chef-client. Los nodos usan chef-client para preguntar al servidor por los detalles de la configuración y realizan tanto trabajo de configuración como les sea posible en el propio nodo. De esta forma la carga de configuración se reparte entre los diferentes equipos de la red.

Un **nodo** es cualquier máquina (física, virtual, cloud, etc.) que está siendo gestionada por Chef. En cada uno de los nodos se instala el servicio chef-client. Cuando chef-client está siendo ejecutado, llevará a cabo todos los pasos necesarios para mantener al nodo en el estado esperado:

- Registrar y autenticar el nodo en el servidor chef.
- Construir el objeto nodo.
- Sincronizar los cookbooks.
- Compilar la colección de recursos cargando cada uno de los cookbooks solicitados.
- Realizar las acciones apropiadas para configurar el nodo.

4.5 K2http

El servicio K2http implementa un consumidor Kafka que recibe los mensajes de los topics especificados en su configuración y los envía al endpoint configurado usando el protocolo https, de esta forma nos permite enviar de forma segura mensajes tomados de Kafka.

Lo normal es usarlo junto con su equivalente http2k que recibe mensajes mediante el protocolo https e implementa un endpoint que envía estos mensajes a Kafka.

4.6 Liblognorm

Como ya se comentó en la subsección 4.1.9, liblognorm es una herramienta para parsear logs. Esto se realiza mediante unos patrones definidos por el usuario que reciben el nombre de reglas.

Las reglas son definidas por el usuario en archivos de texto plano y cada una de ellas concretan la sintaxis de un log usando texto estático o variables. Estas variables pueden ser de distinto tipo y caso de que la regla en cuestión concuerde con la sintaxis del log que se desea parsear, toman el valor que le corresponda según el contenido del log.

El resultado generado por esta herramienta se devuelve en un objeto JSON usando el nombre de las variables especificadas en la regla.

En cuanto a la sintaxis para definir una regla, liblognorm cuenta a día de hoy con dos versiones (v1 y v2) de las cuales para este proyecto se ha usado la v1, ya que la v2, aunque da más facilidades y posibilidades, aún tiene ciertos bugs que entorpecían la tarea de parseo, por lo que se decidió usar la versión anterior hasta que se solventasen estos problemas.

4.6.1 Lognormalizer

Lognormalizer es una sencilla herramienta cuyo uso principal es testear y depurar las reglas para liblognorm escritas por un usuario antes de su uso en un entorno real. Esta herramienta es usada por línea de comandos y, una vez especificado el fichero de reglas a usar, toma cada una de las líneas de la entrada estándar como un log, lo parsea usando liblognorm y muestra el resultado (objeto JSON) en la salida estándar.

4.7 Logger

Logger es una herramienta de testeo que nos permite crear por línea de comandos un log en formato Syslog y enviarlo a la herramienta de administración de logs del sistema. Mediante sus opciones nos permite precisar cada uno de los campos de la cabecera Syslog y el contenido del mensaje.

4.8 Certtool

Certtol es una herramienta de línea de comandos cuyo uso principal es la creación y manipulación de certificados y claves. Como se verá más adelante, el uso principal que se ha dado a esta herramienta ha sido la creación de CA, claves privadas y certificados autofirmados (x.509).

Para crear un certificado, la herramienta pregunta al usuario algunos datos interactivamente, de modo que se necesita la intervención del usuario. Por otro lado, a la hora de generar un certificado también se puede especificar una plantilla que contiene los datos que serán preguntados al usuario, de esta forma no es necesaria la intervención de este.

5 ESTRUCTURA DEL SISTEMA

With the Cloud, individuals and small businesses can snap their fingers and instantly set up enterprise-class services.

- Roy Stephan -

En este capítulo hablaremos sobre la estructura del proyecto realizado, la funcionalidad que cumplen los servicios comentados en el capítulo anterior dentro de este proyecto y, donde ubicar el proyecto dentro de los elementos que componen la arquitectura de redBorder.

La arquitectura de redBorder se basa en tres elementos principales: manager, sensores y Client Proxy. Cada uno de estos elementos juega un papel esencial en el flujo de trabajo del sistema redBorder.

El **manager** es el elemento de interacción con el usuario. En él se recibe toda la información obtenida de las redes del usuario, procesándola, analizándola y mostrándolas al usuario en forma de vistas (una vista por cada tipo de dato) a partir de las cuales se pueden obtener gráficas basadas en estos datos y añadirlas a dashboards. Podemos ver un ejemplo de dashboard en la figura 5-1.

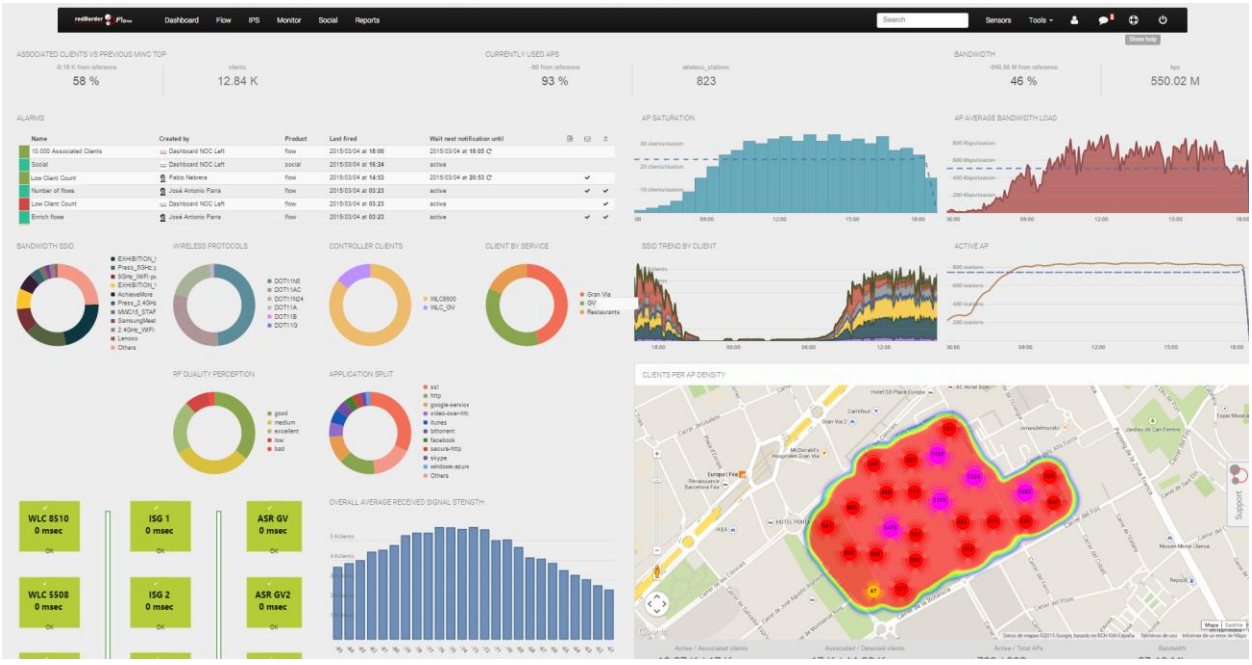


Figura 5-1. Dashboard de redBorder.

Para que esto sea posible los diferentes tipos de datos deben ser enviados al manager en un formato común: JSON. Al ser un formato estructurado, facilita enormemente el procesado de la información por parte del manager.

Además, el manager también sirve como centro de control para los administradores de red, ya que desde la propia interfaz web de este es posible configurar ciertos parámetros del resto de elementos de la estructura redBorder. Esto es posible gracias a que el manager corre un servidor chef usado para automatizar la configuración del resto de elementos de la estructura redBorder. De esta forma, todos los elementos tienen siempre la última configuración simplemente cambiando los cookbooks del servidor chef residente en el manager.

Aunque este elemento puede ser instalado en la infraestructura propia del cliente, recientemente la empresa redBorder, ante la creciente demanda de sus clientes, decidió ofrecer un servicio (redBorder Live) que alojase en la cloud los managers de sus clientes para que estos disfrutasen de las ventajas que conlleva el uso de este tipo de entornos. Este proyecto está desarrollado para este tipo de instalación ya que es hacia donde se está desplazando el modelo de negocio de esta y muchas otras empresas.

Los **sensores** son elementos que se encuentran en la red del cliente y tienen la finalidad de recoger información sobre la red del usuario y hacerla llegar al manager. La información puede ser de distintos tipos (Netflow, SNMP, IPS, etc.) y dependiendo de esto el sensor puede necesitar hardware especifico. Por ejemplo, en el caso de un IPS, debido a la gran capacidad de procesamiento necesaria, deberá ser implementado en un hardware más potente.

Por último, el **Client Proxy** es un nuevo elemento situado en la red del usuario y que se introdujo al mismo tiempo que se produjo la migración del manager a la cloud. Surge de la necesidad de mandar los datos de la red del usuario hasta el manager haciendo uso de una red pública como es Internet. El Client Proxy es un intermediario, entre los sensores y el manager, encargado de enviar los datos de la red del usuario al manager de forma segura. Esto se hace mediante el uso del protocolo HTTPS.

Es en este último elemento donde reside la mayor parte de la funcionalidad de nuestro proyecto, ya que en el Client Proxy será donde se recibirán los mensajes syslog de los sensores (que en este caso serán cualquier elemento de la red capaz de enviar mensajes syslog), se parsearán, se filtrarán, se normalizarán y se les dará formato JSON. De esta forma se obtendrá a partir de una entrada en formato syslog (el mensaje original), una salida en formato JSON apta para ser enviada al manager para su posterior procesamiento y análisis. Al realizar estas tareas en el Client Proxy, estamos repartiendo la carga de procesamiento entre los distintos equipos Client Proxy del cliente, en lugar de concentrarla en el manager.

Como se habrá podido deducir, será necesario configurar los servicios del Client Proxy para que cumplan la finalidad deseada. Para ello tendremos que modificar el cookbook del Client Proxy en el servidor chef del manager, ya que como se ha comentado, el Client Proxy recibe la configuración del manager mediante Chef la primera vez que se conecta y a partir de entonces ejecuta un proceso chef-client que es notificado cuando se produce un cambio en el cookbook.

Además, también será necesario modificar la configuración en los equipos que el cliente desee para mandar mensajes syslog al Client Proxy.

En la figura 5-2 vemos como el Client Proxy recibe logs de las diferentes fuentes presentes en la red del cliente haciendo uso del protocolo Syslog implementado en el servicio Rsyslog. Este último será el encargado de procesar los logs (parsear, filtrar, normalizar, …) para obtener la salida en formato JSON y enviarlos a un topic de Kafka (rb_vault). Los brokers del servicio Kafka están coordinados por Zookeeper.

Una vez tenemos los datos en el topic de Kafka, el servicio k2http se encarga de consumir de la cola Kafka y enviar al manager usando el protocolo HTTPS. En el manager el servicio http2k se encargará de recibir los mensajes y publicarlos en Kafka en el topic “rb_vault”.

Hasta aquí llega el alcance de nuestro proyecto. Para el indexado y procesado de los logs en formato JSON por parte del manager, se usará el mismo procedimiento usado para el resto de tipos de datos, no siendo necesario realizar grandes modificaciones en la configuración del manager. Por otra parte, para presentar los datos al usuario el equipo de desarrolladores web de redBorder creará una nueva vista adaptada a las características de este tipo de datos.

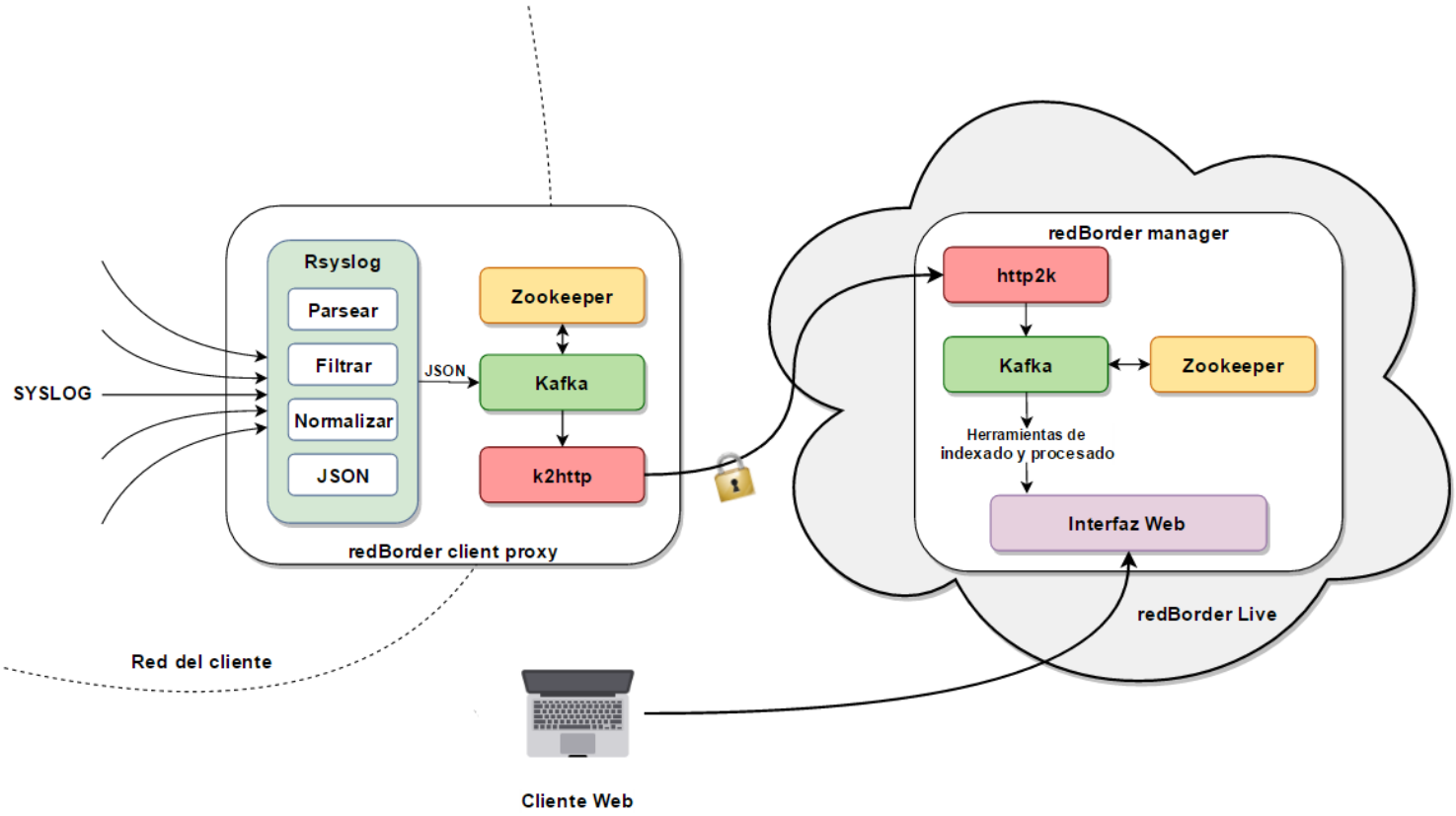


Figura 5-2. Estructura modular.

6 DESARROLLO Y PRUEBAS EN CENTOS

If data had mass, the Earth would be a black hole.

- Stephen Marsland -

Una vez terminada la fase de investigación de este proyecto (gran parte de esta se realizó en paralelo a la fase de desarrollo), comenzamos con este capítulo la fase de desarrollo llevada a cabo durante el transcurso de este proyecto.

El primer paso fue desarrollar una configuración para Rsyslog que se ajustase a las necesidades del sistema que pretendemos integrar en redBorder. Recordemos cuales eran estas necesidades:

- Recepción de logs usando el protocolo Rsyslog sobre UDP, TCP o TLS (a elección del usuario).
- Parseo de la cabecera Rsyslog (ambas versiones).
- Calculo de hash para garantizar la integridad del mensaje desde que llega al Client Proxy.
- Parseo de los datos estructurados de los mensajes que siguen el formato establecido en la RFC 5424.
- Posibilidad de filtrar logs atendiendo al contenido de la cabecera Rsyslog.
- Posibilidad de definir reglas de parseo para el mensaje (sin cabecera Rsyslog).
- Posibilidad de normalizar algunos campos de la cabecera Rsyslog, como, por ejemplo, el timestamp.
- Enviar los logs parseados y normalizados a un topic de Kafka en formato JSON para poder enviarlos al Manager mediante k2http.

Para desarrollar y probar la configuración, se utilizaron dos sistemas virtualizados sobre VirtualBox. El primero es un Centos 7 Minimal de 64 bits, que simula ser el Client Proxy de redBorder, de modo que en este se ejecuta Rsyslog (cumpliendo la función de servidor), Zookeeper y Kafka. El otro es un sistema Ubuntu 15.04 de 64 bits que tiene el papel de cliente y nos sirve para probar la configuración del Client Proxy que desarrollamos en el sistema Centos. La figura 6-1 muestra un esquema que representa el escenario de pruebas usado.

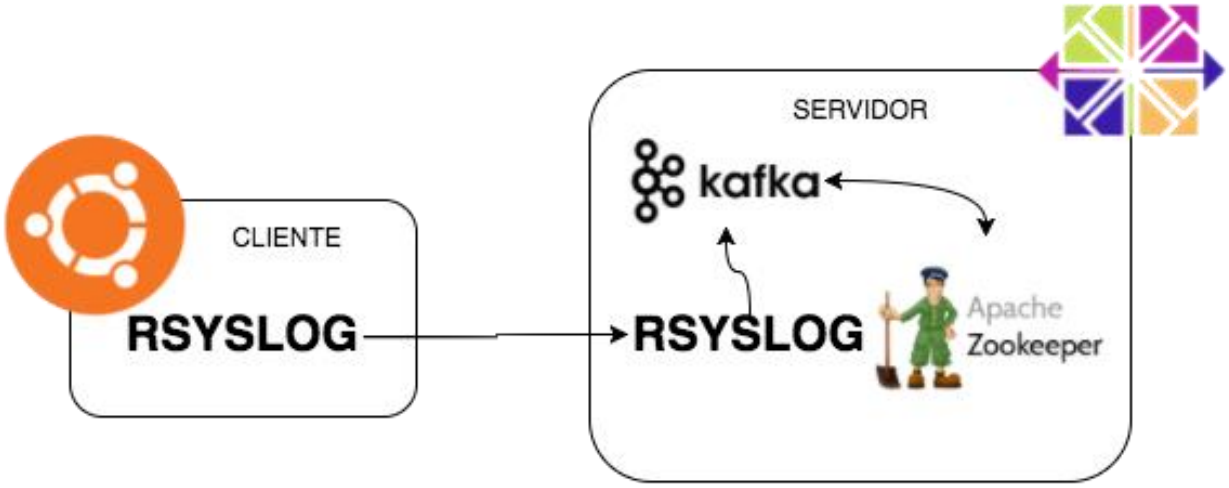


Figura 6-1. Escenario de desarrollo y prueba de configuración de Rsyslog.

Como paso previo a la configuración se necesita instalar Rsyslog (véase anexo B), con los módulos que sean necesarios además de los que se incluyen en la instalación por defecto (los módulos que necesiten instalar algún paquete extra se comentarán en este capítulo), en cliente y servidor. También debemos tener previamente instalado y configurado Zookeeper (véase anexo E) y Kafka (véase anexo F) en el servidor.

Además, es recomendable crear un usuario y grupo en el sistema para Rsyslog y crear directorios, con los permisos adecuados, para almacenar la configuración, reglas de parseo, certificados y logs remotos. Para realizar esto último, se ejecutaron la siguiente secuencia de comandos:

```
#Añadir grupo y usuario syslog
groupadd syslog
useradd -s /sbin/nologin syslog -g syslog

#Creación de directorios y asignación de syslog
#como usuario y grupo propietario.

#En este directorio se almacenarán los logs que
#sean recibido a través de la red.
mkdir -p /var/log/remote/
chown syslog:syslog /var/log/remote/

#Directorio de trabajo para Rsyslog. Aquí se
#almacenará la cola de logs pendientes de
#procesamiento.
mkdir /var/spool/rsyslog
chown syslog:syslog /var/spool/rsyslog/

#En este directorio se almacenarán los certificados
mkdir /etc/rsyslog.d/certificates
chown syslog:syslog /etc/rsyslog.d/certificates

#En este directorio se almacenarán los ficheros de
#reglas de parseo para liblognorm
mkdir /etc/rsyslog.d/rules
chown syslog:syslog /etc/rsyslog.d/rules
```

En este capítulo comentaremos los detalles más importantes de la configuración realizada tanto en el cliente (Ubuntu) como en el servidor (Centos), esta última será la que integremos en el Client Proxy de redBorder. Así mismo, en el anexo D se adjunta la configuración completa tanto de cliente como de servidor.

6.1 Configuración del cliente.

En esta sección comentaremos la configuración de Rsyslog usada para el cliente de prueba. Cabe destacar que sólo mencionaremos la parte de la configuración que nos permite enviar los logs al servidor, quedando la configuración necesaria para añadir fuentes de logs a criterio del usuario del servicio. Principalmente estas fuentes de logs serán archivos de logs creador por los servicios del sistema y, aunque se ha desarrollado una configuración de prueba para recolección de logs en el cliente, cada usuario puede necesitar gestionar logs de diferentes servicios.

La configuración del cliente será distinta según qué protocolo se desee usar para mandar los logs al servidor. En todos los casos, el formato de salida de los logs será el mismo y se corresponderá con el especificado en la RFC 5424. Para ello debemos añadir la siguiente plantilla a la configuración del cliente:

```
template(name="RbVaultFormat" type="string"
string="<%pri%>1 %timestamp:::date-rfc3339% %HOSTNAME% %app-
name% %procid% %msgid% [rbvault@39483 sensor-uuid=\"5fb462b0-
1ba5-4d06-97c4-c48c5a636a02\" tag=\"test\"] %msg%\n")
```

Para poder usar la plantilla en el resto de la configuración es necesario darle un nombre, en este caso el nombre escogido “RbVaultFormat”. Además, como se puede apreciar, debemos especificar el tipo de la plantilla. En este caso la platilla es de tipo string, pero como veremos en la configuración del servidor, existen otros tipos de plantillas.

La plantilla “RbVaultFormat” sigue el formato Syslog versión 1 y además se añade un SD-ELEMENT con dos SD-PARAMS, uno (sensor-uuid) por si el usuario desea identificar el sensor (cliente) mediante un UUID y otro (tag) para que el usuario pueda añadir una etiqueta a sus mensajes. El usuario es libre de completar esta plantilla con cuantos SD-ELEMENTs desee.

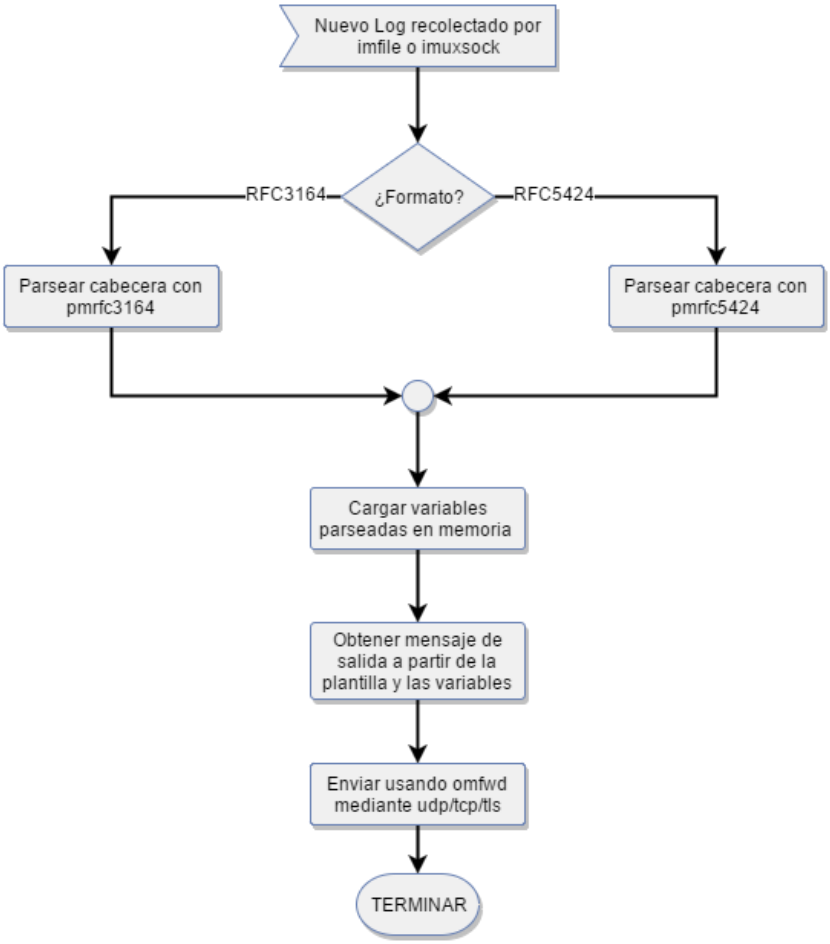


Figura 6-2. Diagrama de flujo del procedimiento seguido en el cliente.

El string de la plantilla es dinámico, de modo que, para cada mensaje se resuelven las variables que aparecen en esta (las variables son las especificadas entre cada pareja de caracteres '%'). Estas variables se obtienen de la cabecera Syslog. Cada mensaje que es leído de algún archivo de logs se pasa por el módulo "pmrfc3164" o "pmrfc5424" según corresponda y se parsea su cabecera Syslog, cargando en memoria las variables de las que nos servimos para la plantilla. Estos dos módulos están incluidos en la instalación por defecto de Rsyslog y además son cargados y usados por defecto. En la figura 6-2 se muestra un diagrama de flujo que representa este procedimiento seguido para cada log en el cliente.

También deberemos cargar el módulo omfwd con independencia del protocolo de transporte que se desee usar, ya que este módulo da soporte al envío de logs usando cualquiera de estos tres protocolos. Para cargar este módulo deberemos introducir en nuestra configuración lo siguiente:

```
module(load="omfwd"  Template="RbVaultFormat")
```

El primer parámetro (omfwd) nos sirve para especificar el módulo a cargar. El segundo parámetro es propio del módulo omfwd y nos sirve para especificar la plantilla que se usará por defecto.

Por último, antes de pasar a la configuración específica para cada protocolo de mensajes, debemos incluir en nuestra configuración las siguientes directivas que tienen como finalidad que nuestros mensajes no se eliminen de la cola de envío en caso de que el receptor no esté disponible. Además, también aseguramos que los mensajes que se encuentren encolados se guarden en el disco en caso de que el sistema necesite apagarse, de este modo, podremos enviar estos mensajes una vez el sistema vuelva a iniciarse.

```
$WorkDirectory /var/spool/rsyslog
$ActionQueueType LinkedList
$ActionQueueFileName srvfwd
$ActionResumeRetryCount -1
$ActionQueueSaveOnShutdown on
```

6.1.1 UDP

Para enviar logs desde el cliente al servidor usando el protocolo UDP, debemos introducir la siguiente acción en la configuración de nuestro cliente:

```
action(type="omfwd" Target="172.16.7.8" Port="514" Protocol="udp")
```

Como se puede observar, se especifica que la acción es de tipo "omfwd", es decir, que se hace uso de este módulo. Además, debemos especificar el destino de los mensajes mediante la IP y el puerto del servidor. Por último, si queremos hacer uso del protocolo UDP tendremos que especificarlo.

En la figura 6-3 podemos ver la captura de Wireshark de un paquete UDP que transporta un log enviado desde el cliente de prueba al servidor.

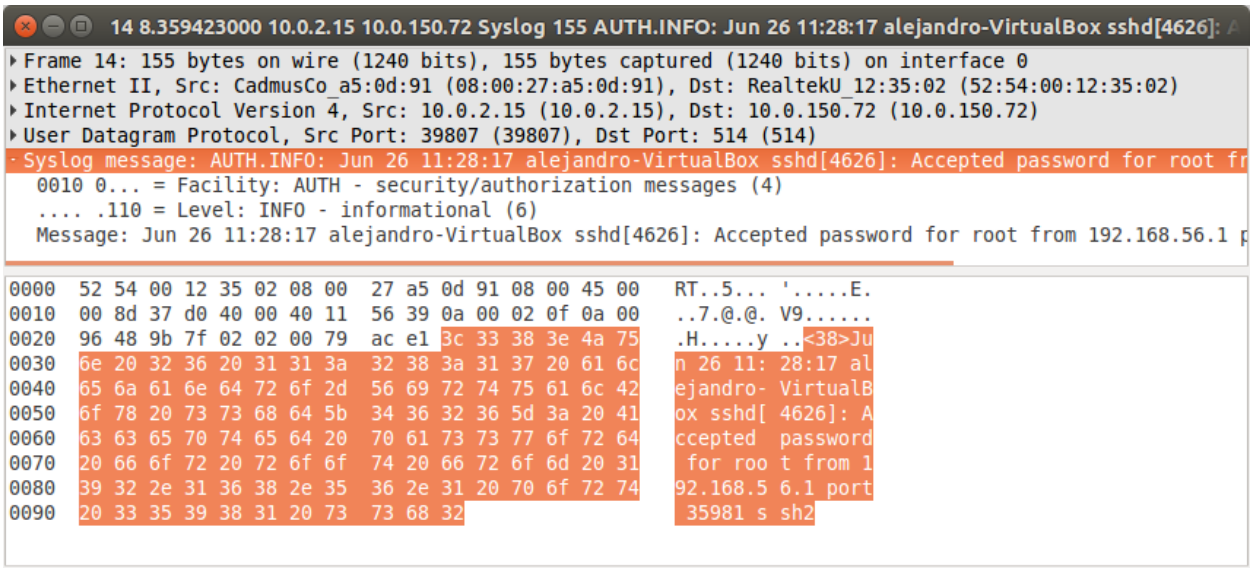


Figura 6-3. Captura Wireshark: Syslog sobre UDP.

6.1.2 TCP

En caso de querer enviar los mensajes Syslog sobre TCP tan sólo debemos especificar el protocolo TCP en la acción:

```
action(type="omfwd" Target="172.16.7.8" Port="514" Protocol="tcp")
```

En la figura 6-4 podemos ver la captura de un mensaje Syslog transportado sobre TCP:

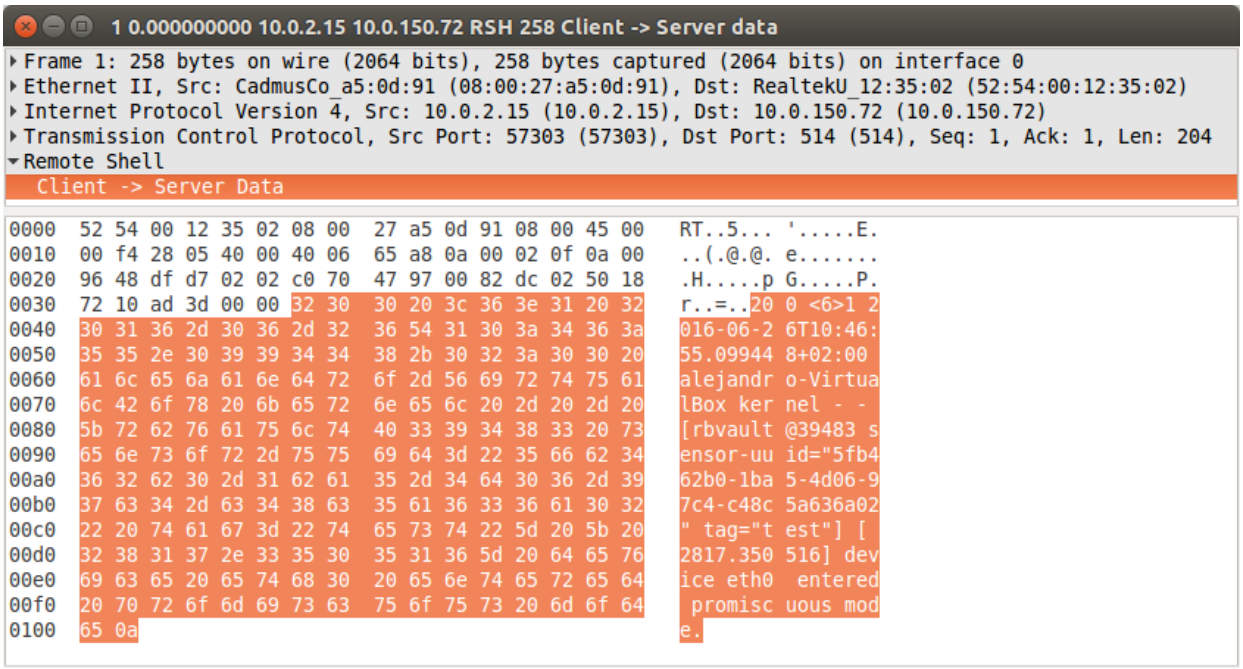


Figura 6-4. Captura Wireshark: Syslog sobre TCP.

6.1.3 TLS

Para el caso de usar la capa de transporte seguro TLS, la configuración es algo más compleja que en los casos anteriores. Antes de pasar a la configuración conviene explicar brevemente el escenario que propone Rsyslog para el uso de TLS [20].

En el protocolo TLS se usan certificados X.509 y por lo tanto criptografía asimétrica para autentificar a la contraparte con quien se están comunicando, y para intercambiar una llave simétrica. Esta sesión es luego usada para cifrar el flujo de datos entre las partes.

El método proporcionado por Rsyslog nos garantiza los siguientes beneficios:

- Los mensajes syslog son encriptados en su transmisión.
- El emisor de mensajes syslog se autentica al receptor.
- El receptor también se autentica al emisor.
- La autenticación mutua de ambos componentes previene de ataques "man-in-the-middle".

El funcionamiento es el siguiente: tenemos una instancia (la Autoridad Certificadora, CA) que se encarga de expedir todos los certificados de máquina. Cada certificado de máquina identifica un host particular.

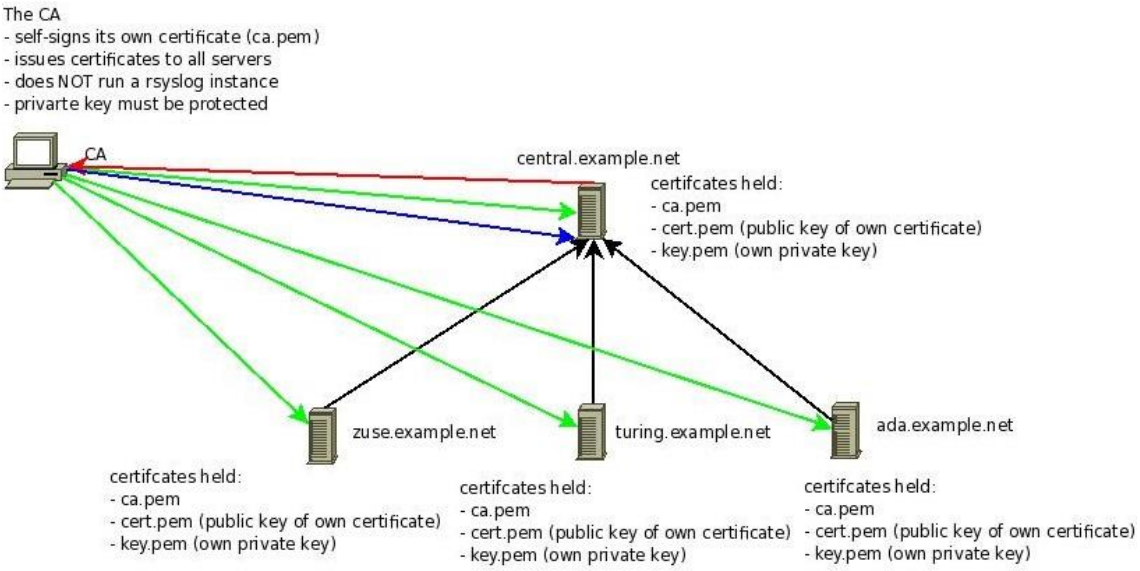


Figura 6-5. Proceso de creación de certificados.

En la figura 6-5 vemos un resumen del proceso de creación de certificados. El proceso se divide en dos fases: una primera fase en la que se configura la autoridad certificadora (CA) generando un certificado auto firmado a partir de una clave privada, y una segunda fase en la que la autoridad certificadora copia el certificado (la parte pública) que ha generado en cada host (flecha verde), cada host genera una petición de certificado a partir de su clave privada y la manda a la autoridad certificadora (flecha roja). Por último, la autoridad certificadora firma un certificado y lo manda a los hosts que lo han solicitado (flecha azul).

El procedimiento para la configuración de la autoridad certificadora y la generación de los certificados de máquina se describe paso a paso en el anexo C. Para ello se hace uso de la herramienta certtool.

Una vez tenemos una visión general del esquema usado para el protocolo TLS, podemos pasar a describir la configuración necesaria para la comunicación mediante este protocolo. Para ello será requisito previo tener instalado en nuestro sistema el paquete "rsyslog-gnutls" y "gnutls-utils", además de la última versión de Rsyslog. Además, debemos generar los certificados de máquina previamente.

En este caso debemos especificar también el protocolo TCP en la acción de envío, pero previamente a esta acción deberemos cambiar el controlador de flujo que se usa por defecto. Para ello deberemos usar la siguiente directiva:

```
$DefaultNetstreamDriver gtls
```

Además, debemos indicar a Rsyslog donde se encuentran la parte pública del certificado autofirmado que nos ha proporcionado la autoridad certificadora, la parte pública del certificado de máquina del cliente y, la clave privada del cliente. Para ello debemos usar las siguientes directivas:

```
$DefaultNetstreamDriverCAFile /etc/rsyslog.d/certificates/ca.pem
$DefaultNetstreamDriverCertFile /etc/rsyslog.d/certificates/cert.pem
$DefaultNetstreamDriverKeyFile /etc/rsyslog.d/certificates/key.pem
```

Antes de proceder a introducir la acción que enviará el mensaje Syslog haciendo uso de TLS es necesario introducir las siguientes directivas:

```
$ActionSendStreamDriverMode 1
$ActionSendStreamDriverAuthMode x509/name
$ActionSendStreamDriverPermittedPeer *.example.net
```

La primera de las directivas nos sirve para especificarle al controlador de flujo que envíe los mensajes Syslog usando el protocolo TLS. Con la segunda estamos configurando el modo de autenticación que, en este caso, es mediante validación de certificado X.509 (usando la clave pública del certificado autofirmado emitido por la autoridad certificadora) y el hostname del equipo con el que nos comunicamos. La última directiva puede ser usada cuantas veces queramos y nos sirve para configurar un equipo como permitido usando su hostname (debe coincidir con el hostname especificado en su certificado válido).

Para finalizar podemos hacer uso de la acción para enviar mensajes Syslog que coincide con el caso de TCP:

```
action(type="omfwd" Target="172.16.7.8" Port="514" Protocol="tcp")
```

Como en los casos anteriores, en la figura 6-6 se muestra la captura de un paquete con Wireshark cuando se ha usado este método. Como se puede observar, al estar el mensaje encriptado, se hace imposible su legibilidad a no ser que dispongamos de la clave adecuada para desencriptarlo, evitando de este modo que terceros no autorizados accedan a nuestra información.

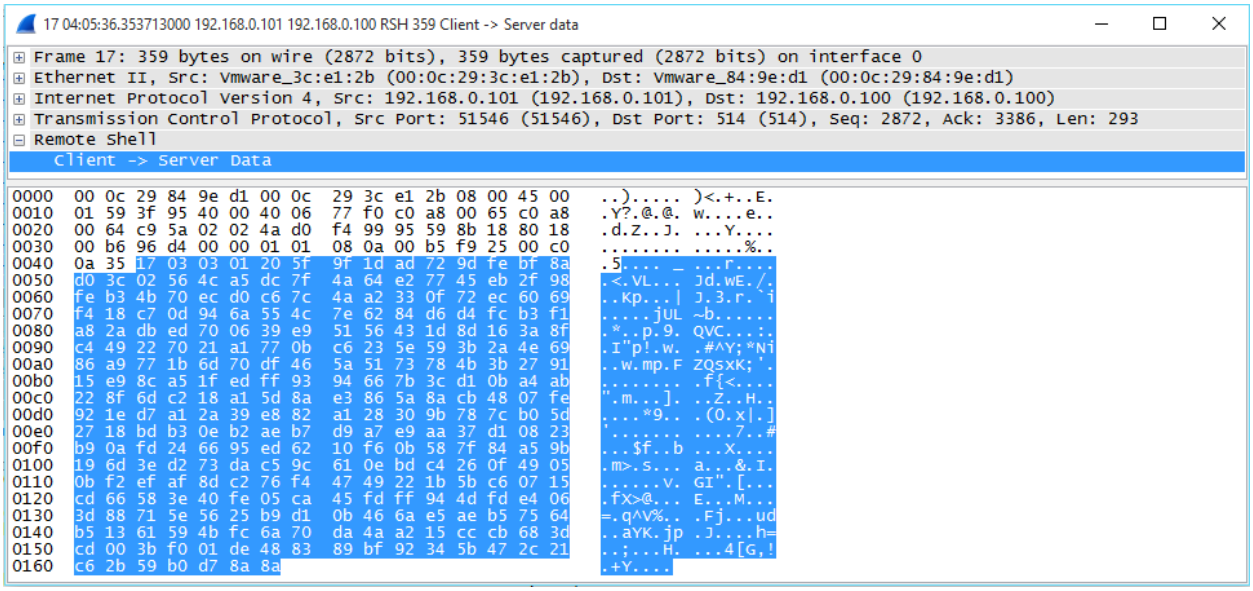


Figura 6-6. Captura Wireshark: Syslog sobre TLS.

6.2 Configuración del servidor

En esta sección desarrollaremos la configuración básica del servidor (la configuración completa puede encontrarse en el anexo D). Como paso previo, además de instalar Rsyslog (incluyendo módulos necesarios), Zookeeper y Kafka, deberemos habilitar el servidor para poder recibir mensajes Syslog en los puertos que se especifiquen en la configuración posteriormente. Para ello será necesario añadir reglas iptables que permitan paquetes entrantes en estos puertos. Esto también se automatizará usando Chef:

```
#Si queremos recibir datos mediante el protocolo TCP
-A INPUT -m state --state NEW,ESTABLISHED,RELATED -m tcp -p tcp --
dport 514 -j ACCEPT

#Si queremos recibir datos mediante el protocolo UDP
-A INPUT -m udp -p udp --dport 514 -j ACCEPT
```

En la figura 6-7 se muestra el diagrama de flujo que describe el proceso que sigue un mensaje Syslog que llega al servidor. En las subsecciones posteriores explicaremos la configuración que posibilita estas acciones.

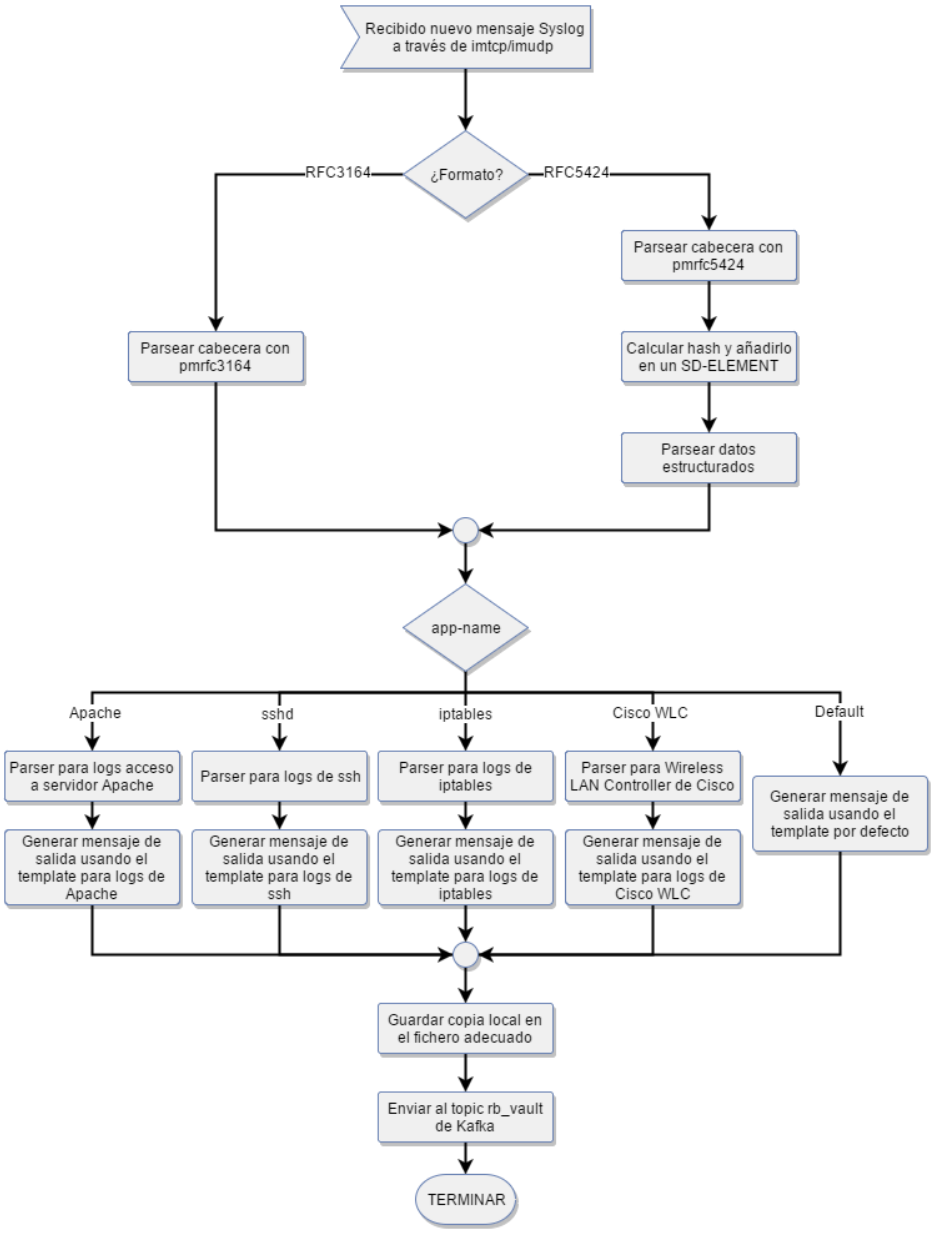


Figura 6-7. Diagrama de flujo que describe el proceso seguido en el servidor.

6.2.1 Recepción de mensajes Syslog

En esta subsección describiremos la configuración necesaria para habilitar el servidor para recibir mensajes Syslog a través de la red. Aunque esta subsección se divide en un apartado para cada protocolo (UDP, TCP y TLS), es posible configurar Rsyslog para recibir datos en varios puertos, que pueden usar distinto protocolo, al mismo tiempo.

En todo caso, podemos comprobar que los puertos se hayan configurado correctamente mediante el siguiente comando:

```
netstat -na | grep 514
```

6.2.1.1 UDP

Para recolectar logs usando el protocolo UDP, debemos cargar previamente el módulo "imudp":

```
module(load="imudp")
```

Además, debemos añadir una sentencia "input" para especificar el puerto UDP donde queremos escuchar los mensajes Syslog:

```
input(type="imudp" port="514")
```

6.2.1.2 TCP

El caso de recepción mediante el protocolo TCP es muy parecido al anterior, pero en lugar de cargar el módulo "imudp" y usarlo en la sentencia "input", debemos hacer lo propio con el módulo "imtcp":

```
module(load="imtcp")
input(type="imtcp" port="514")
```

6.2.1.3 TLS

En el caso de TLS, la configuración es similar a la usada en el cliente. Como requisito previo deberemos tener instalado en nuestro sistema el paquete "rsyslog-gnutls" y "gnutls-utils", además de haber generado previamente los certificados de maquina correspondiente.

El primer paso será cargar el módulo "imtcp" que será el encargado de recibir el flujo TLS haciendo uso de un controlador de flujo distinto al que se usa por defecto:

```
module(load="imtcp")
```

Al igual que en el cliente, deberemos cambiar el controlador de flujo que se usa por defecto. Para ello deberemos usar la siguiente directiva:

```
$DefaultNetstreamDriver gtls
```

También debemos indicar a Rsyslog donde se encuentran la parte pública del certificado autofirmado que nos ha proporcionado la autoridad certificadora, la parte pública del certificado de máquina del cliente y, la clave privada del cliente. Para ello debemos usar las siguientes directivas:

```
$DefaultNetstreamDriverCAFile /etc/rsyslog.d/certificates/ca.pem
$DefaultNetstreamDriverCertFile /etc/rsyslog.d/certificates/cert.pem
$DefaultNetstreamDriverKeyFile /etc/rsyslog.d/certificates/key.pem
```

Como último paso antes de introducir la directiva "input" correspondiente, tenemos que usar las mismas directivas que usamos en el cliente para especificarle al controlador de flujo que envíe los mensajes Syslog usando el protocolo TLS, configurar el modo de autenticación y, configurar los equipos que queremos permitir que envíen logs a esta máquina:

```
$ActionSendStreamDriverMode 1
$ActionSendStreamDriverAuthMode x509/name
$ActionSendStreamDriverPermittedPeer *.example.net
```

Para finalizar debemos introducir la directiva "input" que permitirá abrir el puerto especificado con la configuración que hemos especificado hasta ahora:

```
input(type="imtcp" port="514")
```

6.2.2 Parseado de cabecera Syslog

Una vez se ha recibido un mensaje Syslog en nuestro proceso Rsyslog, el primer paso será parsear su cabecera Syslog. Esto se realizará mediante el módulo "pmrfc3164" o "pmrfc5424" según corresponda.

Estos dos módulos están habilitados en la instalación por defecto de Rsyslog, por lo que no es necesario instalar ningún paquete extra para usarlos. Además, estos módulos son cargados y usados justos después de la recepción en la cadena de acciones por defecto de Rsyslog, por lo que no es necesario ningún tipo de configuración extra para cargar el módulo o ejecutar el parseo de la cabecera.

La información parseada se cargan en memoria en variables como las usadas para escribir el template usado en el cliente. A continuación, se muestra una lista de estas variables:

- *raw-message:* El log tal y como ha sido recibido en el socket.
- *message*: La parte del log correspondiente al mensaje. Contiene un mensaje con formato libre que proporciona información sobre el evento.
- *pri*: Entero que identifica la prioridad del log. Se calcula a partir de los campos syslogfacility y syslogseverity.
- *pri-text*: Forma textual del campo pri.
- *syslogfacility*: Entero que identifica el componente del sistema que ha generado el log.
- *syslogfacility-text*: Forma textual para el campo syslogfacility.
- *syslogseverity*: Entero que identifica el nivel de gravedad del log.
- *syslogseverity-text*: Forma textual para el campo syslogseverity.
- *protocol-version*: Versión del protocolo Syslog usado.
- *timestamp*: Especifica el tiempo en el que se generó el log.
- *hostname*: Identifica el equipo del cual proviene el log.
- *fromhost-ip*: Dirección IP del equipo que generó el log.
- *app-name*: Identifica la aplicación que generó el log.
- *procid*: Identificador de proceso asociado con un sistema syslog.
- *msgid*: Identificador del tipo de mensaje.

6.2.3 Cálculo del hash

Para que sea posible el cálculo y la inclusión del hash en el mensaje, este debe seguir el formato especificado en la RFC 5424, debido al funcionamiento del módulo "mmrfc5424addhmac". Este módulo permite calcular y añadir un hash en un SD-PARAM dentro de un SD-ELEMENT. Es por esto que no es posible usar este módulo para mensajes con el formato Syslog antiguo.

El hash es calculado sobre el mensaje original y se coloca en el SD-PARAM con SD-NAME "hash", dentro del SD-ELEMENT que especifiquemos en la configuración y que no debe existir previamente en el mensaje original.

Las funciones de hashing disponibles son aquellas ofrecidas por la herramienta openssl, ya que esta es usada por el módulo "mmrfc5424addhmac" para el cálculo del hash. Para este proyecto se ha usado la función SHA256.

Para comprobar la integridad del mensaje deberemos eliminar el SD-ELEMENT introducido por el módulo para así obtener el mensaje original, sin modificaciones. A partir del mensaje obtenido, tendremos que calcular el hash usando la misma función de hashing y la misma clave. Por último, comprobar si el hash calculado coincide con el transportado en el mensaje y si es así, la integridad del mensaje está comprobada.

Como paso previo a la configuración tenemos que instalar el módulo "mmrfc5424addhmac". Este módulo no viene habilitado por defecto en la instalación de Rsyslog y tampoco existe un paquete externo que nos permita instalarlo posteriormente. Debido a esto tendremos que habilitarlo en una instalación personalizada de Rsyslog (compilando el código) o compilar el módulo por separado y copiar el archivo de extensión "so" que se genera en la carpeta "/lib64/rsyslog". El proceso completo se detalla en el anexo B.

Una vez instalado el módulo, podemos pasar a cargarlo en la configuración como ya se ha hecho en otras ocasiones:

```
module(load="mmrfc5424addhmac")
```

Por último, tendremos que añadir una sentencia "action" para especificar que queremos usar este módulo:

```
action(type="mmrfc5424addhmac" key="yourenterprisekey"
hashFunction="sha256" sd_id="hash@39483")
```

Como se puede apreciar, la sentencia "action" tiene varias opciones:

- *key:* indica la clave que se usará para generar el hash a partir del mensaje.
- *hashFunction:* indica la función de hashing que se usará para generar el hash a partir del mensaje.
- *sd_id:* indica el SD-ID del SD-ELEMENT que se creará para contener el hash calculado.

6.2.4 Parseado de datos estructurados

El parseado de los datos estructurados se realizará sólo en el caso de que el mensaje siga el formato de la RFC 5424, ya que en el formato Syslog anterior no existía esta parte del mensaje.

Es estrictamente necesario colocar esta configuración después de la configuración mostrada en la subsección anterior, ya que al añadir el hash en un SD-ELEMENT, si se hace uso de este módulo previamente, no se parseará el SD-ELEMENT del hash y no tendremos acceso a este mediante variables cargadas en memoria.

Para esta fase será necesario hacer uso del módulo "mmpstrucdata" que está habilitado en la instalación por defecto de Rsyslog, por lo que no será necesaria la instalación de ningún paquete extra. Sí que será necesario cargar el módulo en la configuración:

```
module(load="mmpstrucdata")
```

Una vez cargado el módulo debemos indicar en la configuración una acción para usarlo. Esto se puede realizar sencillamente de la siguiente forma:

```
    action(type="mmpstrucdata")
```

Una vez ejecutada esta acción, la información parseada se carga en memoria en la variable "structured-data". Esta variable es accesible desde nuestra configuración, por lo que puede ser incluida en una plantilla. Contiene la información estructurada en un objeto JSON, de forma que cada SD-ELEMENT será un componente de este JSON. Los SD-IDs correspondientes serán los nombres que identifican al componente y el valor será otro objeto JSON que contendrá una pareja clave-valor por cada SD-PARAM presente en el SD-ELEMENT. A continuación, se muestra un ejemplo que ilustra lo que comúnmente se puede encontrar en la variable "structured-data":

Ejemplo 6-1. *Contenido de la variable "structured-data"*

```
{
      "rb_vault@39483" : {
            "tag" : "test",
            "sensor-uuid" : "fd6fc5e1-e30a-48f1-a109-d4af0ddcec01" },
      "hash@39483" : {
            "hash" :
"dbdb99f3b16e40c95b02f2f14e19031b760db11dfc19659e32f706071e5224ec" }
}
```

Si queremos acceder a un SD_ELEMENT o SD_PARAM concreto deberemos acceder al árbol de variables JSON. Para hacer esto, partimos de la raíz "$!rfc5424-sd" (esta variable nos mostraría los mismo que la variable structured-data) y vamos añadiendo niveles separados por el carácter '!'. De forma que si, por ejemplo, queremos acceder al parámetro tag del SD-ELEMENT "rb_vault@39483" del ejemplo anterior, tendremos que usar la variable "$!rfc5424-sd!rbvault@39483!tag".

6.2.5 Filtrado de mensajes

El filtrado de mensajes se usa para decidir que parseador usar para cada mensaje, aunque también podría usarse para no procesar logs que cumplan determinadas características. Además, también mejora el rendimiento del servidor Rsyslog ya que un mensaje sólo pasa por el parseador que tiene las reglas de parseo específicamente preparados para este tipo de mensajes. En caso de no usar esto, el mensaje debería pasar por todos los parseadores hasta encontrar una regla que coincida con este o, hasta acabar todas las reglas, lo que es mucho más ineficiente.

En este proyecto los mensajes se pasan por un parseador u otro (o por ninguno) dependiendo de la aplicación que generó el log. Para esto se utilizan las palabras reservadas "if" y "then". Entre estas dos palabras se coloca la expresión a evaluar y después de "then" se coloca un bloque de acciones agrupadas por los caracteres "{}". En el ejemplo 6-2 se muestra el filtro usado para los mensajes de ssh.

Ejemplo 6-2. *Filtro usado para los mensajes de ssh*

```
if $app-name == 'sshd' and $fromhost-ip != '127.0.0.1' then {
      #Acciones a ejecutar para los logs de ssh.
      stop
}
```

Si queremos que nuestro mensaje sólo pase por las acciones de un bloque controlado por una expresión "if…then", tendremos que colocar la sentencia "stop" al final de este bloque de acciones. Esto es especialmente útil si queremos que en caso de no existir un parseador para el tipo de mensaje, queremos que se ejecuten unas acciones por defecto.

6.2.6 Parseado del mensaje

Para parsear el mensaje, como ya se ha comentado en varias ocasiones, se hace uso de la herramienta "Liblognorm" mediante el módulo de Rsyslog cuyo nombre es "mmnormalize". Este módulo deberá ser previamente instalado ya que no está habilitado en la instalación por defecto de Rsyslog. Para ello deberemos instalar en nuestro sistema el paquete "rsyslog-mmnormalize" siguiendo la guía de instalación del anexo B.

Además, debemos cargar el módulo en la configuración de Rsyslog previamente a hacer uso de este:

```
module(load="mmnormalize")
```

En este proyecto, por las razones comentadas en la subsección anterior, el uso de este módulo se realiza siempre en un bloque de acciones controlado por una sentencia "if…then", que permite la ejecución del bloque sólo en el caso de que la aplicación que generó el log se corresponda con la aplicación para la que se ha escrito las reglas de parseo.

Para hacer uso de este módulo deberemos usar una sentencia "action" como la que se muestra a continuación:

```
action(type="mmnormalize"
ruleBase="/etc/rsyslog.d/rules/ssh.rules" useRawMsg="off")
```

Con el atributo "ruleBase" especificamos el fichero de reglas liblognorm a usar y mediante el atributo "useRawMsg" indicamos si queremos pasar a Liblognorm el log completo ("on") o el log sin cabecera ("off"). Esta última es la opción utilizada en este proyecto, ya que la cabecera ha sido procesada previamente.

Al ejecutar esta acción sobre un mensaje, se recorre el fichero de reglas especificado buscando una regla cuyo formato se corresponda con el mensaje. Al encontrar la primera coincidencia en el fichero de reglas, se para la búsqueda, de forma que las reglas restantes no se comparan con el mensaje.

Una regla Liblognorm es un string que describe el formato que debe tener el log mediante texto estático y variables (cada variable se identifica por un nombre y un tipo) cuyo valor será obtenido del mensaje. Si se produce alguna coincidencia, se cargan en memoria las variables especificadas en la regla en cuestión y pueden ser accedidas para su uso en plantillas o filtros mediante el nombre de variable indicado en la regla.

En caso de que el mensaje no coincida totalmente con ninguna de las reglas, se cargaría en la variable "unparsed" la parte del mensaje que no ha podido ser parseada o el mensaje entero en caso de que la coincidencia con alguna regla sea nula.

Como caso especial en que el mensaje contenga información en formato JSON podría usarse el módulo "mmjsonparse", lo que nos ahorraría la tarea de escribir reglas para Liblognorm. Este módulo es capaz de parsear un objeto JSON y cargar su contenido en memoria, de forma que tenemos la información parseada disponible haciendo referencia a ella mediante la clave del valor que queremos obtener precedida de los caracteres "$!". Si el valor al que queremos acceder es otro objeto JSON podemos descender niveles tan sólo usando los nombres de las claves separadas por el carácter '!'.

Para hacer uso de este módulo, además de instalar el paquete "rsyslog-mmjsonparse" previamente, debemos cargarlo en la configuración e introducir la sentencia "action" correspondiente:

```
module(load="mmjsonparse")
action(type="mmjsonparse")
```

6.2.7 Definición de la plantilla de salida

Antes de enviar el mensaje Syslog a un fichero o a Kafka, debemos definir una plantilla de salida que definirá el formato que tendrá el mensaje en el fichero o en Kafka. Esas plantillas nos servirán para normalizar nuestros logs en el formato JSON, lo cual era un requisito impuesto por redBorder para este proyecto.

En el ejemplo 6-3 se muestra la plantilla usada por defecto para los logs que no han pasado por ningún parseador (excluyendo los de cabecera):

Ejemplo 6-3. *Plantilla por defecto usada en este proyecto.*

```
template(name="norm_rfc5424" type="list"){
  constant(value="{")
  constant(value="\"raw-message\":\"") property(name="rawmsg"
format="json")
  constant(value="\",\"message\":\"") property(name="msg"
format="json")
  constant(value="\",\"pri\":\"") property(name="pri" format="json")
  constant(value="\",\"pri-text\":\"") property(name="pri-text"
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="\",\"protocol-version\":\"")
property(name="protocol-version" format="json")
  constant(value="\",\"timestamp\":\"") property(name="timestamp"
dateFormat="unixtimestamp")
  constant(value="\",\"hostname\":\"") property(name="hostname"
format="json")
  constant(value="\",\"fromhost-ip\":\"") property(name="fromhost-ip"
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="\",\"structured-data\":\"")
property(name="structured-data" format="json")
  constant(value="\",\"tag\":\"") property(name="$!rfc5424-
sd!rbvault@39483!tag" format="json")
  constant(value="\",\"sensor-uuid\":\"") property(name="$!rfc5424-
sd!rbvault@39483!sensor-uuid" format="json")
  constant(value="\",\"hash\":\"") property(name="$!rfc5424-
sd!hash@39483!hash" format="json")
  constant(value="\",\"inputname\":\"") property(name="inputname"
format="json")
  constant(value="\",\"proxy-uuid\":\"") constant(value="fd6fc5e1-
e30a-48f1-a109-d4af0ddcec01")
  constant(value="\"}")
}
```

Como se puede apreciar en el ejemplo anterior, el tipo de plantilla usado no se corresponde con el usado para la plantilla de salida en el cliente. En esta ocasión la plantilla es de tipo lista y consiste en una secuencia de elementos de tipo “constant” y “property”. Los elementos “constant” nos permiten introducir texto estático, por lo que los usaremos para introducir en la plantilla los elementos característicos de un objeto JSON (‘{’, ‘}’, ‘,’ ,…) y los nombres de cada una de las parejas clave-valor. Por otra parte, los elementos de tipo “property” nos permiten incluir en la plantilla un valor dinámico representado por una variable.

En los elementos de tipo “property” debemos indicar el nombre de la variable a usar, que será una de las

variables obtenidas al parsear la cabecera, datos estructurados o mensaje, y el formato, que nos permitirá entre otras cosas:

- Escapar comillas dobles presentes en el valor de una variable e imprimir el árbol de un objeto JSON almacenado en una variable. Esto se realiza usando el valor "json" para la propiedad "format", lo que nos permite construir un JSON sin errores de sintaxis.
- Normalizar la marca temporal de creación del log. Independientemente del formato de origen, usando el valor "unixtimestamp" para la propiedad "foramat", introduciremos en el mensaje de salida la marca temporal en un formato común. El formato elegido para este proyecto consiste en el número de segundos transcurridos desde 00:00:00 UTC de 01/01/1970.

6.2.8 Copia local en fichero

Como medida para aumentar la redundancia del sistema se decide mantener una copia en fichero local de los logs que lleguen al Client Proxy. Estos ficheros pueden ser gestionados por herramientas como Logrotate, para así poder rotar los archivos para que no se hagan demasiado grandes y eliminar los logs que tengas más de cierto tiempo.

Para mandar logs a ficheros locales se hace uso del módulo "omfile" que viene habilitado en la instalación por defecto de Rsyslog y además se carga en la configuración sin necesidad de especificarlo.

Para hacer uso de este módulo tendremos que usar una sentencia "action" en la que indiquemos el fichero en el que queremos guardar el mensaje y la plantilla a usar, que debe estar previamente definida en la configuración:

```
action( type="omfile"
        file="/var/log/remote/norm_rfc5424.log"
        template="norm_rfc5424" )
```

6.2.9 Enviar a Kafka

Como en el resto de procesos comenzamos enumerando los requisitos previos. En este caso debemos tener previamente instalados y configurados los servicios Zookeeper y Kafka (estos procesos se encuentran detallados en los anexos E y F respectivamente). Además, deberemos instalar el paquete "rsyslog-kafka" (instalación detallada en el anexo B), ya que es necesario el módulo "omkafka" de Rsyslog y no viene instalado por defecto.

Como primer paso en nuestra configuración, tenemos que cargar el módulo "omkafka":

```
module(load="omkafka")
```

A continuación, debemos incluir una sentencia "action" como la siguiente:

```
action( type="omkafka"
   topic="rb_vault"
   broker="localhost:9092"
   template="norm_rfc5424" )
```

Como se puede observar, además del parámetro "type" para indicar el módulo a usar, se pasan otros tres parámetros:

- *topic:* especifica el topic de Kafka a donde queremos enviar el mensaje. En este caso se trata del topic rb_vault que deberá estar creado previamente o configurar Kafka para que puedan añadirse topics dinámicamente en tiempo de ejecución.
- *broker:* especifica la dirección IP y el puerto donde escucha el broker de Kafka al que queremos enviar el mensaje.
- *template:* especifica la plantilla de salida a usar, que debe estar previamente definida.

7 INTEGRACIÓN EN REDBORDER

Information is the oil of the 21st century, and analytics is the combustion engine.

- Peter Sondergaard -

Durante este apartado hablaremos sobre la fase de integración del sistema en redBorder. Una vez hemos obtenido una configuración funcional en el entorno de pruebas del capítulo 6, tenemos que trasladar esta configuración al Client Proxy de redBorder. Esto no puede hacerse mediante la simple copia de los ficheros de configuración, ya que como se ha comentado en otra ocasión, la configuración de los servicios del Client Proxy es gestionado por la herramienta Chef.

Es por ello que debemos modificar el cookbook de nombre "redBorder-proxy" en el servidor Chef que se ejecuta en el Manager de redBorder. Inicialmente se creó un cookbook independiente que automatizara la configuración de Rsyslog y abriese los puertos necesarios en el servicio iptables. Una vez se obtuvo una versión funcional de este cookbook, se integró en el cookbook "redBorder-proxy" y los cambios introducidos en este cookbook se integraron en el SDK de redBorder para que estuviesen presentes en posteriores versiones del Client Proxy. A partir de este momento, se llevó a cabo un proceso de prueba y mejora continuo hasta llegar a la versión del cookbook presentada en esta memoria cuyos cambios respecto a la versión convencional de redBorder pueden encontrarse al completo en el anexo G.

Es importante destacar que el cookbook parte del supuesto de que todos los paquetes necesarios para Rsyslog y sus módulos, Zookeeper y Kafka, se encuentran ya instalados, ya que este no realiza ningún tipo de instalación de componentes. Esto es posible ya que la versión de Centos del Client Proxy está modificada para que instale los paquetes necesarios en el momento de la instalación del Client Proxy (el proceso de instalación del Client Proxy está detallado en el anexo A).

7.1 Modificaciones del cookbook para Rsyslog

En esta sección comentaremos de forma general cuales han sido los cambios introducidos en el cookbook del Client Proxy. Recordamos que todos los ficheros que han sido modificados o incluidos en el cookbook se encuentran completamente accesibles en el anexo G.

A continuación, se muestra la estructura de directorios del cookbook "redBorder-proxy", donde se ha marcado en color azul los archivos que ya existían, pero han necesitado modificaciones y en verdes los archivos nuevos creados para este proyecto:

```
redBorder-proxy
|-- attributes
|   |-- default.rb
|
|-- providers
|   |-- rsyslog.rb
|
|-- recipes
|   |-- default.rb
|
|-- resources
|   |-- rsyslog.rb
|
`-- templates
    |-- default
      |-- iptables.erb
      |-- rsyslog.conf.erb
      |-- rsyslog_01-server.conf.erb
      |-- rsyslog_02-general.conf.erb
      |-- rsyslog_03-parse_ssh.conf.erb
      |-- rsyslog_04-parse_apacheaccess.conf.erb
      |-- rsyslog_05-parse_iptables.conf.erb
      |-- rsyslog_06-parse_Cisco_WLC.conf.erb
      |-- rsyslog_99-parse_rfc5424.conf.erb
      |-- rsyslog_Cisco_WLC.rules.erb
      |-- rsyslog_iptables.rules.erb
      `-- rsyslog_ssh.rules.erb
```

Como se puede observar en el árbol de directorios, el cookbook se compone de 5 directorios que iremos comentado en las siguientes subsecciones.

7.1.1 attributes

En el directorio “attributes” se almacenan los ficheros de definición de atributos. Estos atributos nos sirven para personalizar algo la configuración dependiendo del nodo. Una vez esté disponible la interfaz web en el Manager de redBorder para Vault, los usuarios podrán acceder a través de ella a la configuración de Vault y dar valores a estos atributos para cada Client Proxy de su red.

Los tipos permitidos para estos atributos se corresponden con los tipos básicos de Ruby. En este proyecto se han usado atributos de tipo boolean, entero, string y, array de strings. A continuación, mostramos ejemplos de definición de atributos de cada uno de estos tipos:

```
default["redBorder"]["proxy"]["rsyslog"]["enable_tcp"] = true
default["redBorder"]["proxy"]["rsyslog"]["tcp_port"] = 10514
default["redBorder"]["proxy"]["rsyslog"]["config_dir"] =
  "/etc/rsyslog.d"
default["redBorder"]["proxy"]["rsyslog"]["permitted_peers_array"]
  = ['*.example.com','*.redborder.com']
```

Como podemos observar, todos estos atributos comienzan con la raíz “[‘redBorder’][‘proxy’][‘rsyslog’]” que ha sido asignada a atributos pertenecientes a este proyecto. Además, todos los atributos están precedidos de la palabra “default” que los convertirá en atributos por defecto y se usarán en caso de que no se use una configuración específica para el nodo.

Además, para definir un nuevo atributo, podemos hacer referencia a un atributo anteriormente creado, de modo que, si queremos definir un atributo de tipo string para el directorio de certificados a partir del directorio de configuración, debemos hacerlo de la siguiente forma:

```
default["redBorder"]["proxy"]["rsyslog"]["certificates_dir"] =
  "#{node["redBorder"]["proxy"]["rsyslog"]["config_dir"]}/certific
  ates"
```

En este proyecto ha sido necesario modificar el archivo "default.rb", que está pensado para definir atributos por defecto. Entre otros, se han definido atributos que permiten al usuario personalizar los protocolos de transporte a usar, el puerto donde recibir en cada protocolo, la función de hashing y la clave a usar para generar el hash, los directorios de Rsyslog, el usuario y grupo de Rsyslog, etc.

7.1.2 resources

El directorio "resources" está pensado para almacenar ficheros de definición de recursos. Un recurso representa una parte del sistema y sería el equivalente a una interfaz en los lenguajes de programación orientados a objetos. Al definir un recurso, estamos indicando qué acciones se pueden ejecutar sobre este, qué acción se ejecutará por defecto y, qué atributos se le pueden pasar para que los use durante la acción ejecutada. Es importante destacar que definimos las acciones que pueden ejecutarse sobre el recurso, pero no que hacen dichas acciones.

Una vez definido el recurso, podrá hacerse uso de este, mediante las acciones definidas y pasando los atributos indicados, en las recetas del cookbook.

Para este proyecto se ha creado el archivo "rsyslog.rb" que define el recuso con el mismo nombre y que tiene como única acción, y por tanto acción por defecto, la acción ":config". Para especificar esto debemos usar las siguientes directivas en el archivo anterior:

```
actions :config
default_action :config
```

Además, se han definido varios atributos como atributos de entrada para poder usarlos en la acción ":config". Para definir un atributo de entrada, debemos indicar un identificador, el tipo (usando los tipos de Ruby) y un valor por defecto para el caso de que se llame al recurso sin pasar dicho atributo. A continuación, mostramos los atributos definidos para el recuso rsyslog:

```
attribute :enable_tls, :kind_of => [TrueClass, FalseClass],
  :default => true
attribute :config_dir, :kind_of => String, :default =>
  "/etc/rsyslog.d"
attribute :certificates_dir, :kind_of => String, :default =>
  "/etc/rsyslog.d/certificates"
attribute :rules_dir, :kind_of => String, :default =>
  "/etc/rsyslog.d/rules"
attribute :remote_logs_dir, :kind_of => String, :default =>
  "/var/log/remote"
attribute :work_dir, :kind_of => String, :default =>
  "/var/spool/rsyslog"
attribute :user, :kind_of => String, :default => "syslog"
attribute :group, :kind_of => String, :default => "syslog"
```

7.1.3 providers

El directorio "resources" está pensado para almacenar ficheros de definición de proveedores. Mientras que un recurso representa una parte del sistema, un proveedor define los pasos a seguir para llevar esta parte del sistema al estado deseado. Volviendo a la analogía con los lenguajes de programación orientada a objetos, el proveedor sería el equivalente a la implementación de la interfaz, define el comportamiento de los métodos especificados en la interfaz (:config en este caso).

Para definir el comportamiento de una acción debemos incluir en el archivo de definición de proveedor una estructura como la siguiente:

```
action :config do
  begin
    ...
  rescue Exception => e
    Chef::Log.error(e.message)
  end
end
```

Tras la directiva “begin” debemos situar los pasos a seguir para ejecutar la acción que se está definiendo. Para comenzar debemos tomar los valores de los atributos que definen el estado al que se debe llegar y que se especifican en la llamada a la acción del recurso en la receta correspondiente. Para ello, estos atributos deben haber sido declarados en la definición del recurso (subsección anterior) y debe usarse una sentencia como la siguiente:

```
enable_tls = new_resource.enable_tls
```

En esta sentencia, damos un nombre local al atributo para poder referirnos a este dentro de la acción que se está definiendo y, tomamos el valor del atributo usando el nombre con el que se definió en el fichero de definición de recurso.

Si volvemos a fijarnos en la estructura de definición de una acción, las sentencias que se encuentran entre “rescue” y el primer “end”, sirven para capturar cualquier ejecución que ocurra durante la ejecución de la acción e imprimir un log de error en la terminal que esté ejecutando chef-client.

Para este proyecto se ha definido el archivo de definición de proveedor de nombre “rsyslog.rb” para definir el comportamiento de la acción “:config” que se declaró en el fichero de definición de recursos.

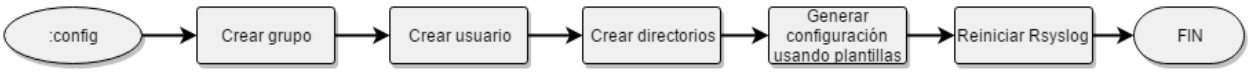


Figura 7-1. Diagrama de flujo acción “:config”.

La figura 7-1 muestra el diagrama de flujo que describe la metodología seguida en la acción “:config”. Para comenzar, una vez se han recibido los atributos y cargado en variables locales, se procede a la creación del grupo de Rsyslog, haciendo uso de la acción “:create” del recurso “group” y del valor del atributo “group”:

```
group group do
   action  :create
end
```

Una vez creado el grupo, se puede proceder a la creación del usuario que usará Rsyslog, especificando que el usuario pertenece al grupo anteriormente creado. Además, el usuario no debe poder logearse en el sistema, por lo que debemos especificar que use “/sbin/nologin” como shell. Usaremos la acción “:create” del recurso “user”:

```
user user do
   comment 'rsyslog user'
   shell   '/sbin/nologin'
   group group
   action  :create
end
```

El siguiente paso es crear los directorios necesarios para Rsyslog. Estos directorios son los siguientes:

- Directorio de configuración de Rsyslog. Por defecto: "/etc/rsyslog.d/".
- Directorio de certificados. Su existencia sólo es necesaria en caso de que se use el protocolo TLS. Por defecto: "/etc/rsyslog.d/certificates/".
- Directorio de reglas de liblognorm. Por defecto: "/etc/rsyslog.d/rules/".
- Directorio de trabajo de Rsyslog. En él se guardarán las colas de mensajes pendientes de procesado. Por defecto: "/var/spool/rsyslog/".
- Directorio de logs remotos. Almacenará las copias locales de los logs que reciba el Client Proxy. Por defecto: "/var/log/remote/".

Cada uno de estos directorios serán recibidos en atributos cuando se llame a la acción ":config". Para crear un directorio, hacemos uso de la acción ":create" del recurso "directory". Tenemos que especificar el usuario y grupo propietarios, los permisos y la ruta del propio directorio (atributo):

```
if enable_tls
   directory certificates_dir do
      owner   user
      group   group
      mode    '0755'
      action  :create
   end
end
```

Como se puede observar en el código anterior, es posible hacer uso de estructuras de control de Ruby siempre que sea necesario. Esto nos ayuda por ejemplo cuando tenemos elementos opcionales que dependen de la elección del usuario. Por ejemplo, si el usuario decide no usar TLS el directorio de certificados no es necesario y su creación puede omitirse.

Para terminar, tendremos que computar las plantillas atendiendo a los atributos del nodo donde se está ejecutando Chef para, de esta forma, generar los archivos de configuración apropiados. Para ello, tendremos que usar el recurso "template", indicándole la ruta del fichero de configuración que debe crear y la plantilla en la que debe basarse. Además, tendremos que especificar algunos datos del fichero a crear cómo el grupo y usuario propietario, y los permisos:

```
template "#{config_dir}/99-parse_rfc5424.conf" do
   source  'rsyslog_99-parse_rfc5424.conf.erb'
   owner 'root'
   group   'root'
   mode  '0644'
   retries 2
   notifies :restart, 'service[rsyslog]', :delayed
end
```

La última sentencia que aparece en el recurso "template", sirve para notificar al servicio Rsyslog (no confundir con el recurso de mismo nombre que hemos definido en este cookbook) que debe ejecutar la acción ":restart" al acabar la ejecución de chef-client (":delayed"). Esto es útil ya que, si cambiamos la configuración de Rsyslog, es necesario reiniciar el servicio para que los cambios surjan efecto.

En el árbol de directorios, presentado al principio de esta sección, podemos ver una lista de todas las plantillas (excluyendo la plantilla de nombre "iptables.erb") que se han creado exclusivamente para este proyecto y para cada una de las cuales se ha definido una estructura como la anterior en la acción ":config" del recurso "rsyslog" creado.

7.1.4 recipes

Como ya sabemos, una "recipe" o "receta" es el elemento fundamental de configuración en un CookBook de Chef. En una receta podemos describir el proceso a seguir para que nuestro sistema llegue al estado de configuración deseado. Esto se consigue por medio de código Ruby y el uso de recursos, definidos por Chef, o definidos por el usuario, como es nuestro caso.

Para este proyecto ha sido necesario modificar la receta por defecto del cookbook "redBorder-proxy" para hacer uso del recurso creado previamente y configurar Rsyslog. Para ello se pasan como atributos los atributos por defecto del nodo y ejecutamos la acción ":config" que definimos en la subsección anterior, como podemos ver a continuación:

```
redborder_proxy_rsyslog "config" do
  enable_tls node["redBorder"]["proxy"]["rsyslog"]["enable_tls"]
  config_dir node["redBorder"]["proxy"]["rsyslog"]["config_dir"]
  certificates_dir
    node["redBorder"]["proxy"]["rsyslog"]["certificates_dir"]
  rules_dir node["redBorder"]["proxy"]["rsyslog"]["rules_dir"]
  remote_logs_dir
    node["redBorder"]["proxy"]["rsyslog"]["remote_logs_dir"]
  work_dir node["redBorder"]["proxy"]["rsyslog"]["work_dir"]
  user node["redBorder"]["proxy"]["rsyslog"]["user"]
  group node["redBorder"]["proxy"]["rsyslog"]["group"]
  action :config
end
```

7.1.5 templates

Como ya hemos comentado, las plantillas que se han creado para este cookbook son las que aparecen en el árbol de directorios que se encuentra al principio de esta sección. Si el lector echa un vistazo a estas plantillas, observará que básicamente se componen de los archivos de configuración y ficheros de reglas de parseo obtenidos en el desarrollo comentado en el capítulo anterior, enriquecidos con elementos que ayudan a que la configuración de Rsyslog sea dinámica según los atributos del nodo.

A continuación, se muestran algunos ejemplos de los distintos elementos usados en las plantillas antes mencionadas:

- Valores de atributos de Chef. Si introducimos la referencia a un atributo de Chef entre las secuencias de caracteres "<%= " y " %>", esta referencia y las secuencias de caracteres anteriores serán sustituidas por el valor del atributo indicado.

```
input(type="imudp" port="<%=
  node["redBorder"]["proxy"]["rsyslog"]["udp_port"] %>")
```

- Código Ruby embebido. Podemos introducir código Ruby entre las secuencias de caracteres "<% " y " %>". El código ruby será ejecuta al tiempo que se procesa la plantilla por lo que esta funcionalidad es útil para introducir estructuras de control de Ruby. En el siguiente ejemplo de muestra una sentencia "if":

```
<% if node["redBorder"]["proxy"]["rsyslog"]["enable_udp"] %>
  module(load="imudp")
  input(type="imudp" port="<%=
    node["redBorder"]["proxy"]["rsyslog"]["udp_port"] %>")
<% end %>
```

También es posible introducir estructuras de tipo bucle como la estructura "each … do" que podemos ver en el siguiente ejemplo, el cual recorre todos los valores del atributo "permitted_peers_array" de tipo "array":

```
<%
  node["redBorder"]["proxy"]["rsyslog"]["permitted_peers_array"].e
  ach do |peer| %>
    $InputTCPServerStreamDriverPermittedPeer <%= peer %>
<% end %>
```

- Código Bash embebido. También es posible ejecutar código Bash cuya salida sustituirá a la sentencia en la plantilla y pasará a formar parte del archivo generado. Para ello deberemos colocar el código Bash entre las cadenas de caracteres "<%= `" y "` %>". En el siguiente ejemplo puede observase como esta característica es usada para obtener el UUID del Client Proxy e introducirlo en una constante de una plantilla de salida (dentro de la configuración de Rsyslog) para los mensajes Syslog:

```
constant(value="<%= `cat /opt/rb/etc/rb-uuid | tr -d "\n\r"` %>")
```

7.2 Reglas iptables

Además de modificar el cookbook del Client Proxy para introducir en los nodos la configuración necesaria de Rsyslog, también es necesario modificar este cookbook para habilitar en iptables los puertos necesarios para poder recibir logs en el servicio Rsyslog del Client Proxy.

Para ello, deberemos introducir las siguientes dos reglas en la plantilla de configuración de iptables (fichero "/redBorder-proxy/templates/iptables.erb"):

```
<% if node["redBorder"]["proxy"]["rsyslog"]["enable_tcp"] %>
  -A INPUT -m state --state NEW,ESTABLISHED,RELATED -m tcp -p tcp
    --dport <%= node["redBorder"]["proxy"]["rsyslog"]["tcp_port"]
    %> -j ACCEPT
<% end %>
<% if node["redBorder"]["proxy"]["rsyslog"]["enable_udp"] %>
  -A INPUT -m udp -p udp --dport <%=
    node["redBorder"]["proxy"]["rsyslog"]["udp_port"] %> -j
    ACCEPT
<% end %>
```

Como se puede observar, hacemos uso de dos estructuras de control "if" y de los atributos definidos para incluir la regla iptables sólo en caso de que sea necesaria, ya que abrir un puerto que no va a ser usado supone un riesgo de seguridad innecesario. Además, también se usan los parámetros definidos para abrir el puerto que dicte los atributos del nodo. De esta forma damos al usuario la libertad de usar el puerto que desee y no preocuparse si el puerto está ocupado por otro servicio.

7.3 Otras modificaciones

Para que la integración en el Client Proxy de redBorder sea completa, hemos tenido que hacer otras modificaciones de menor envergadura: cambiar la configuración de k2http y cambiar los topics creados por defecto en la configuración del Client Proxy.

Para la creación automática del topic "rb_vault" en la configuración del Client Proxy, hemos modificado el script que se encarga de crear estos topics ("rb_create_topics.sh"). Ha sido necesario modificar la variable "TOPICS", existente en este script, para añadir el topic "rb_vault".

```
TOPICS="$*"
[ "x$TOPICS" == "x" ] && TOPICS="rb_event rb_flow rb_monitor rb_loc
  rb_nmsp rb_social rb_hashtag rb_state rb_radius rb_vault"
```

Por otra parte, en la configuración de k2http, ha sido necesario especificar que escuche en el topic “rb_vault” para que envíe los mensajes al Manager de redBorder:

```
kafka:
  broker: "127.0.0.1:9092"  # kafka broker
  consumergroup: "k2http"   # Consumer group ID
  begining: false           # Reset offset
  topics:                   # Kafka topics to listen
    - rb_nmsp
    - rb_radius
    - rb_flow
    - rb_loc
    - rb_monitor
    - rb_state
    - rb_social
    - rb_vault
```

8 PARSEADORES Y RESULTADOS

Information is a source of learning. But unless it is organized, processed, and available to the right people in a format for decision making, it is a burden, not a benefit.

- William Pollard -

En este capítulo hablaremos sobre los parseadores que hemos realizado para este proyecto. Se han realizado parseadores para mensajes log de los servicios ssh, iptables y apache. Además, también se ha realizado un parseador para los mensajes log más comunes del Wireless LAN Controller de Cisco, en concreto el modelo 5500.

La metodología usada para la creación de parseadores consiste en la investigación previa de los logs generados por el servicio en cuestión. Esta investigación consiste tanto en buscar información en la documentación del servicio como en la recogida de gran cantidad de muestras de logs de este servicio. Con esto se pretende encontrar un formato común para los mensajes de ese servicio.

Tras esto, comienza el desarrollo de reglas de parseo que se ajusten al formato de los mensajes más comunes del servicio. Por último, entramos en la fase de prueba y error en la que probamos y corregimos nuestras reglas con muestras de logs del servicio, haciendo uso de la herramienta Lognormalizer. Esta herramienta nos permite pasarle un fichero de reglas, muestras de los logs a parsear mediante la entrada estándar y nos muestra por la salida estándar los logs parseados según las reglas de parseo del fichero pasado.

La figura 8-1 muestra una captura del consumidor Kafka de línea de comandos escuchando el topic "rb_vault". Como se puede apreciar, los mensajes parseados son correctamente enviados a Kafka en formato JSON, mostrando la información parseada tanto de la cabecera Syslog, como del mensaje en el caso de que exista un parseador para los mensajes del servicio en cuestión. El objeto JSON siempre tiene un campo con el mensaje original para casos en los que sea necesario consultarlo. Además, también podemos observar campos con información extra como el hash calculado, el UUID del Client Proxy, la etiqueta puesta por el usuario, etc.

La figura 8-1 es poco legible, por lo que, a partir de ahora, para presentar los resultados, se copiará uno de los mensajes recibidos y se copiará en esta memoria de una forma más legible.

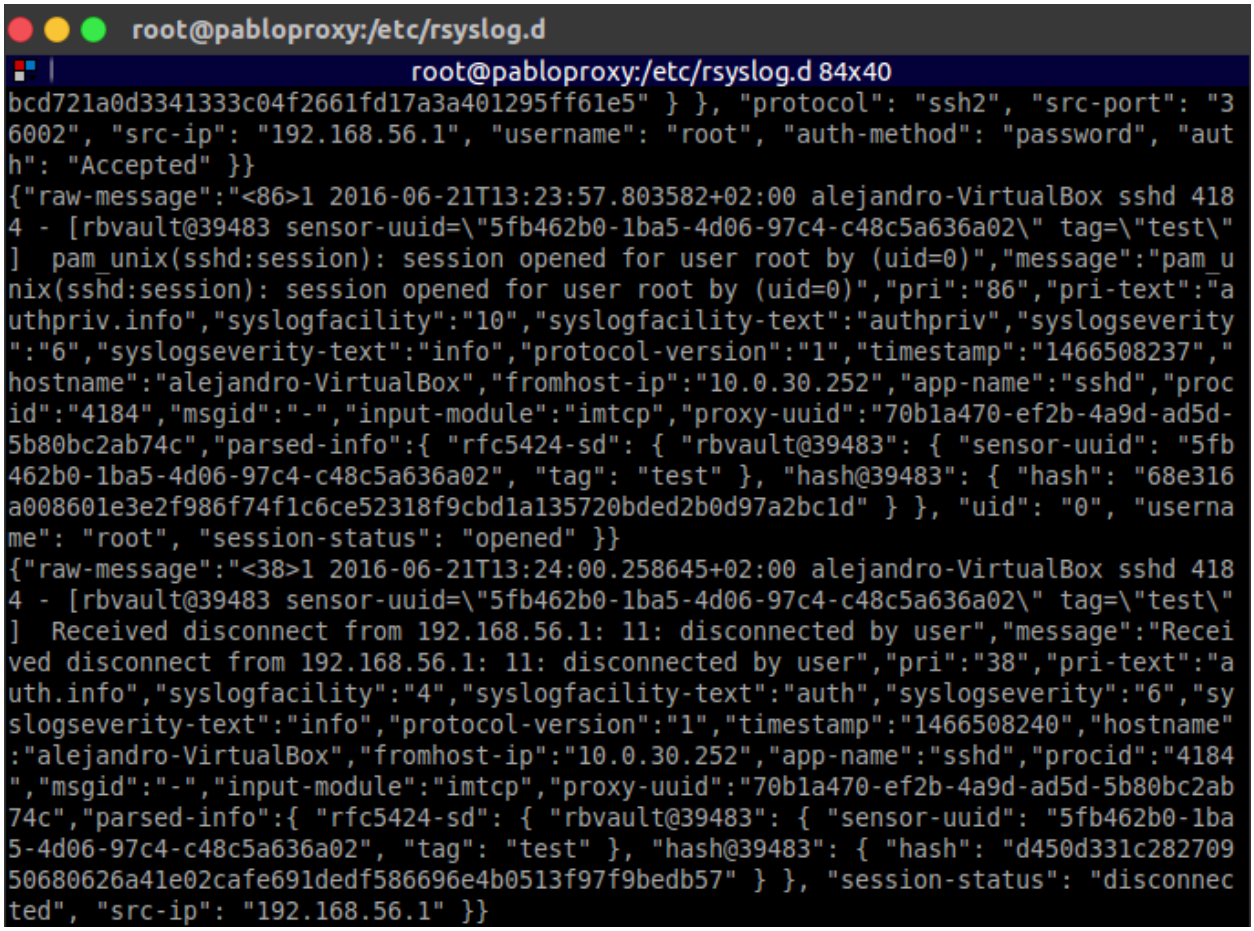


Figura 8-1. Consumidor Kafka ecuchando en el topic “rb_vault”.

8.1 Parseador ssh

El parseador ssh está basado en reglas Liblognorm. Se han escrito reglas de parseo para las situaciones más comunes que pueden ocurrir al usar este servicio: usuario logeado correctamente, clave incorrecta, usuario inexistente, etc.

A continuación, se muestra un ejemplo de log de este servicio una vez a pasado por nuestro sistema. El mensaje original puede verse en el campo “raw-message” del objeto JSON.


```
{
    "raw-message": "<38>1 2016-06-21T13:23:57.802572+02:00 alejandro-
 VirtualBox sshd 4184 - [rbvault@39483 sensor-uuid="5fb462b0-1ba5-4d06-
 97c4-c48c5a636a02" tag="test"] Accepted password for root from
 192.168.56.1 port 36002 ssh2",
    "message": "Accepted password for root from 192.168.56.1 port 36002
 ssh2",
    "pri": "38",
    "pri-text": "auth.info",
    "syslogfacility": "4",
    "syslogfacility-text": "auth",
    "syslogseverity": "6",
    "syslogseverity-text": "info",
    "protocol-version": "1",
    "timestamp": "1466508237",
    "hostname": "alejandro-VirtualBox",
   "fromhost-ip": "10.0.30.252",
```

```
      "app-name": "sshd",
      "procid": "4184",
      "msgid": "-",
      "input-module": "imtcp",
      "proxy-uuid": "70b1a470-ef2b-4a9d-ad5d-5b80bc2ab74c",
      "parsed-info": {
            "rfc5424-sd": {
                  "rbvault@39483": {
                        "sensor-uuid": "5fb462b0-1ba5-4d06-97c4-
          c48c5a636a02",
                        "tag": "test"
                  },
                  "hash@39483": {
                        "hash":
"19252bf05f3098d074da7bcd721a0d3341333c04f2661fd17a3a401295ff61e5"
                  }
            },
            "protocol": "ssh2",
            "src-port": "36002",
            "src-ip": "192.168.56.1",
            "username": "root",
            "auth-method": "password",
            "auth": "Accepted"
      }
}
```


8.2 Parseador iptables

El parseador iptables también está basado en reglas Liblognorm. En esta ocasion se han escrito reglas para los protocolos más comunes, por lo que abarcamos mensajes tanto entrantes como salientes de los protocolos UDP, TCP e ICMP.

Para posibilitar la generación de logs en el servicio iptables, el usuario debe incluir en sus tablas iptables reglas con el objetivo "LOG". Por ejemplo, la siguiente regla iptables, crea logs cuando se reciben paquetes TCP del servicio ssh. Además, añade como prefijo al log la cadena "iptables:" que nos servirá para diferenciar estos logs del resto de logs del kernel.

```
iptables -A INPUT -p tcp --dport 22 --syn -j LOG --log-prefix='iptables:'
--log-level 4
```

A continuación, se muestra un ejemplo de log de este servicio una vez a pasado por este parseador:


```
{
      "raw-message" : "<4>1 2016-06-21T13:33:41.678144+02:00 alejandro-
 VirtualBox kernel - - [rbvault@39483 sensor-uuid=" 5fb462b0-1ba5-4d06-
 97c4-c48c5a636a02 " tag=" test "] [ 7627.676350] iptables:IN=eth0 OUT=
 MAC=08:00:27:a5:0d:91:52:54:00:12:35:02:08:00 SRC=10.0.150.72
 DST=10.0.2.15 LEN=40 TOS=0x00 PREC=0x00 TTL=64 ID=3336 PROTO=TCP SPT=514
 DPT=53114 WINDOW=65535 RES=0x00 ACK URGP=0 ",
      "message" : "[ 7627.676350] iptables:IN=eth0 OUT=
 MAC=08:00:27:a5:0d:91:52:54:00:12:35:02:08:00 SRC=10.0.150.72
 DST=10.0.2.15 LEN=40 TOS=0x00 PREC=0x00 TTL=64 ID=3336 PROTO=TCP SPT=514
 DPT=53114 WINDOW=65535 RES=0x00 ACK URGP=0 ",
      "pri" : "4",
      "pri-text" : "kern.warning",
      "syslogfacility" : "0",
      "syslogfacility-text" : "kern",
      "syslogseverity" : "4",
```

```
      "syslogseverity-text" : "warning",
      "protocol-version" : "1",
      "timestamp" : "1466508821",
      "hostname" : "alejandro-VirtualBox",
      "fromhost-ip" : "10.0.30.252",
      "app-name" : "kernel",
      "procid" : "-",
      "msgid" : "-",
      "input-module" : "imtcp",
      "proxy-uuid" : "70b1a470-ef2b-4a9d-ad5d-5b80bc2ab74c",
      "parsed-info" : {
            "rfc5424-sd" : {
                  "rbvault@39483" : {
                        "sensor-uuid" : "5fb462b0-1ba5-4d06-97c4-
c48c5a636a02",
                        "tag" : "test"
                  },
                  "hash@39483" : {
                        "hash" :
"a3cc5667d84e1d90a6de8f41bed61a03085d7ff54b0824a0ba70c5d3607d15d3"
                  }
            },
            "tcp-urgent-pointer" : "0",
            "tcp-pkt-type" : "ACK",
            "tcp-reserved" : "0x00",
            "tcp-window-size" : "65535",
            "dst-port" : "53114",
            "src-port" : "514",
            "proto" : "TCP",
            "id" : "3336",
            "ttl" : "64",
            "precision" : "0x00",
            "type-of-service" : "0x00",
            "frame-length" : "40",
            "dst-ip" : "10.0.2.15",
            "src-ip" : "10.0.150.72",
            "ethertype" : "08:00",
            "src-mac" : "52:54:00:12:35:02",
            "dst-mac" : "08:00:27:a5:0d:91",
            "in-interface" : "eth0",
            "kernel-timestamp" : " 7627.676350"
      }
   }
```


8.3 Parseador Cisco Wireless LAN Controller

Como en los casos anteriores, este parseador también está basado en reglas Liblognorm. En este caso, la variedad de logs generados es mucho mayor que en el caso de los servicios anteriores. En este caso se han creado reglas para las situaciones en las que se modifica la tabla de estaciones conectadas (creación, modificación y borrado de entradas).

A continuación, al igual que en las secciones anteriores, mostramos un ejemplo del resultado obtenido por este parseador:

```
{
      "raw-message": "<134>WLC1: *sisfSwitcherTask: Jun 21 13:12:53.702:
#SISF-6-ENTRY_CREATED: sisf_shim_utils.c:486 Entry created
A=fe80::14db:754d:50f3:470e V=30 I=wireless:0 P=0005 M=",
      "message": " *sisfSwitcherTask: Jun 21 13:12:53.702: #SISF-6-
ENTRY_CREATED: sisf_shim_utils.c:486 Entry created
A=fe80::14db:754d:50f3:470e V=30 I=wireless:0 P=0005 M=",
      "pri": "134",
      "pri-text": "local0.info",
      "syslogfacility": "16",
      "syslogfacility-text": "local0",
      "syslogseverity": "6",
      "syslogseverity-text": "info",
      "protocol-version": "0",
      "timestamp": "1466507365",
      "hostname": "cisco-capwap-controller.redborder.lan",
      "fromhost-ip": "10.0.50.40",
      "app-name": "WLC1",
      "procid": "-",
      "msgid": "-",
      "input-module": "imudp",
      "proxy-uuid": "70b1a470-ef2b-4a9d-ad5d-5b80bc2ab74c",
      "parsed-info": {
            "P": "0005",
            "I": "wireless:0",
            "V": "30",
            "station_ip": "fe80::14db:754d:50f3:470e",
            "action": "created",
            "line": "486",
            "file": "sisf_shim_utils.c",
            "MNEMONIC": "SISF-6-ENTRY_CREATED",
            "process": "sisfSwitcherTask"
      }
  }
```


8.4 Parseador Apache-access

Este parseador se usa para los logs de acceso a páginas web alojadas en un servidor Apache. A diferencia de los anteriores, este parseador no está basado en reglas Liblognorm, sino que hace uso del módulo "mmjsonparse" de Rsyslog.

Este módulo parsea logs en formato JSON por lo que en usuario debe cambiar la configuración de Apache para que sus logs tengan dicho formato. Para ello basta con incluir las siguientes directivas en el archivo de configuración de Apache:

```
#Plantilla de logs para redBorder VAULT
LogFormat
   "@cee:{\"time\":\"%t\",\"remoteIP\":\"%a\",\"host\":\"%V\",\"request\":
   \"%U\",\"query\":\"%q\",\"method\":\"%m\",\"status\":\"%>s\",\"userAgen
   t\":\"%{User-agent}i\",\"referer\":\"%{Referer}i\",\"X-Forwarded-
   For\":\"%{X-Forwarded-For}i\"}" combined

#Guardar logs de acceso en el fichero /var/log/apache2/access.log con el
   formato JSON especificado.
CustomLog ${APACHE_LOG_DIR}/access.log combined
```

La secuencia de caracteres "@cee:" que se añade al comienzo de la plantilla de salida, es necesaria para que Rsyslog sepa que a continuación viene un mensaje en formato JSON.

Mostramos a continuación el correspondiente ejemplo de resultado para este parseador:


```
{
      "raw-message": "<190>1 2016-06-21T13:26:35.291208+02:00 alejandro-
VirtualBox apache.access - - [rbvault@39483 sensor-uuid=\"5fb462b0-1ba5-
4d06-97c4-c48c5a636a02\" tag=\"test\"]
@cee:{\"time\":\"[21\/Jun\/2016:13:26:35
+0200]\",\"remoteIP\":\"192.168.56.1\",\"host\":\"192.168.56.101\",\"reques
t\":\"\/index.html\",\"query\":\"\",\"method\":\"GET\",\"status\":\"200\",\
"userAgent\":\"Mozilla\/5.0 (X11; Linux x86_64) AppleWebKit\/537.36 (KHTML,
like Gecko) Chrome\/49.0.2623.110 Safari\/537.36\",\"referer\":\"-\",\"X-
Forwarded-For\":\"-\"}",
      "message": "@cee:{\"time\":\"[21\/Jun\/2016:13:26:35
+0200]\",\"remoteIP\":\"192.168.56.1\",\"host\":\"192.168.56.101\",\"reques
t\":\"\/index.html\",\"query\":\"\",\"method\":\"GET\",\"status\":\"200\",\
"userAgent\":\"Mozilla\/5.0 (X11; Linux x86_64) AppleWebKit\/537.36 (KHTML,
like Gecko) Chrome\/49.0.2623.110 Safari\/537.36\",\"referer\":\"-\",\"X-
Forwarded-For\":\"-\"}",
      "pri": "190",
      "pri-text": "local7.info",
      "syslogfacility": "23",
      "syslogfacility-text": "local7",
      "syslogseverity": "6",
      "syslogseverity-text": "info",
      "protocol-version": "1",
      "timestamp": "1466508395",
      "hostname": "alejandro-VirtualBox",
      "fromhost-ip": "10.0.30.252",
      "app-name": "apache.access",
      "procid": "-",
      "msgid": "-",
      "structured-data": "[rbvault@39483 sensor-uuid=\"5fb462b0-1ba5-4d06-
97c4-c48c5a636a02\" tag=\"test\"][hash@39483
hash=\"94a5796ab96eea9b68fc1d6aa40119d14c1441e94259be0c6b5bec9aa5e2cbf3\"]"
,
      "tag": "test",
      "sensor-uuid": "5fb462b0-1ba5-4d06-97c4-c48c5a636a02",
      "hash":
"94a5796ab96eea9b68fc1d6aa40119d14c1441e94259be0c6b5bec9aa5e2cbf3",
      "inputname": "imtcp",
      "proxy-uuid": "70b1a470-ef2b-4a9d-ad5d-5b80bc2ab74c",
      "access-time": "[21\/Jun\/2016:13:26:35 +0200]",
      "remoteIP": "",
      "host": "192.168.56.101",
      "request": "\/index.html",
      "query": "",
      "method": "GET",
      "status": "200",
      "userAgent": "",
      "referer": "-",
      "X-Forwarded-For": ""
   }
```

8.5 Otros casos

En caso de recibir logs de un servicio para el cual no hemos creado ningún parseador, se parseará la cabecera y los datos estructurados como el resto de mensajes, con la diferencia de que el JSON de salida no contendrá información parseada del mensaje.

A continuación, mostramos un ejemplo basado en un log del servicio "su", el cual no dispone actualmente de parseador en nuestro sistema:


```
{
      "raw-message": "<86>1 2016-06-21T13:36:47.799950+02:00 alejandro-
VirtualBox su 4584 - [rbvault@39483 sensor-uuid=\"5fb462b0-1ba5-4d06-97c4-
c48c5a636a02\" tag=\"test\"]  Successful su for alejandro by root",
      "message": "Successful su for alejandro by root",
      "pri": "86",
      "pri-text": "authpriv.info",
      "syslogfacility": "10",
      "syslogfacility-text": "authpriv",
      "syslogseverity": "6",
      "syslogseverity-text": "info",
      "protocol-version": "1",
      "timestamp": "1466509007",
      "hostname": "alejandro-VirtualBox",
      "fromhost-ip": "10.0.30.252",
      "app-name": "su",
      "procid": "4584",
      "msgid": "-",
      "structured-data": "[rbvault@39483 sensor-uuid=\"5fb462b0-1ba5-4d06-
97c4-c48c5a636a02\" tag=\"test\"][hash@39483
hash=\"98822d0f7ee1ea86c1e23de8edd6b0b3709c4ee7562fbb7b3b2dff7a8aaaeea8\"]"
,
      "tag": "test",
      "sensor-uuid": "5fb462b0-1ba5-4d06-97c4-c48c5a636a02",
      "hash":
"98822d0f7ee1ea86c1e23de8edd6b0b3709c4ee7562fbb7b3b2dff7a8aaaeea8",
      "inputname": "imtcp",
      "proxy-uuid": "70b1a470-ef2b-4a9d-ad5d-5b80bc2ab74c"
  }
```

9 PRESUPUESTO

A budget is telling your money where to go instead of wondering where it went

- Dave Ramsey -

En este capítulo haremos una estimación del coste que podría suponer reproducir este proyecto. Para ello dividimos los costes en costes asociados a recursos humanos y costes asociados a recursos materiales. Además, también dividiremos el coste en recursos humanos en las distintas fases que se han llevado a cabo durante el desarrollo de este proyecto. La tabla 9-1 muestra este presupuesto.

Tabla 9-1. Presupuesto del proyecto.

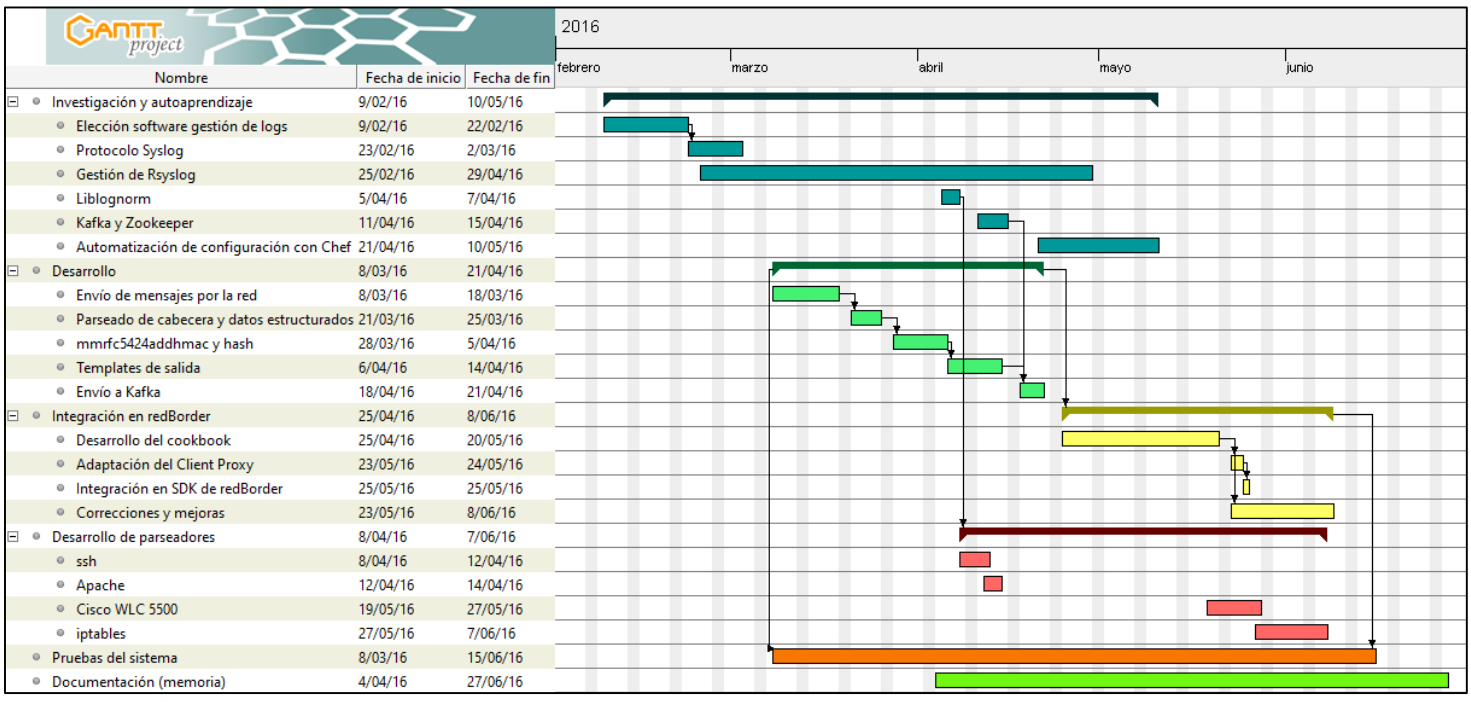
PRESUPUESTO			
RECURSOS HUMANOS			**4.875,00 €**
Concepto	**Coste/hora (€/h)**	**Total horas (h)**	**Coste total (€)**
Ingeniero de Telecomunicaciones Junior			
Investigación y documentación del proyecto	13,00	75	975,00 €
Desarrollo del sistema	13,00	150	1.950,00 €
Integración en redBorder	13,00	100	1.300,00 €
Mejora continua	13,00	50	650,00 €
RECURSOS MATERIALES			**6.749,00 €**
Concepto	**Coste unidad**	**Total unidades**	**Coste total (€)**
Ordenador Portátil HP Pavilion g6-2006es	749,00 €	1	749,00 €
Anfitrión de virtualización Cisco UCS C260 M2	6.000,00 €	1	6.000,00 €
TOTAL			**11.624,00 €**

10 PLANIFICACIÓN

A goal without a plan is just a wish.

- Antoine de Saint-Exupéry -

En este capítulo mostramos mediante la figura 10-1, un diagrama de Gantt en el que se describe, de forma general, la planificación seguida a lo largo de este proyecto. Como se puede observar, en el diagrama se muestran las tareas más generales que se han llevado a cabo clasificadas en varias secciones: investigación, desarrollo, integración, desarrollo de parseadores, pruebas y documentación.



Nombre	Fecha de inicio	Fecha de fin
Investigación y autoaprendizaje	9/02/16	10/05/16
Elección software gestión de logs	9/02/16	22/02/16
Protocolo Syslog	23/02/16	2/03/16
Gestión de Rsyslog	25/02/16	29/04/16
Liblognorm	5/04/16	7/04/16
Kafka y Zookeeper	11/04/16	15/04/16
Automatización de configuración con Chef	21/04/16	10/05/16
Desarrollo	8/03/16	21/04/16
Envío de mensajes por la red	8/03/16	18/03/16
Parseado de cabecera y datos estructurados	21/03/16	25/03/16
mmrfc5424addhmac y hash	28/03/16	5/04/16
Templates de salida	6/04/16	14/04/16
Envío a Kafka	18/04/16	21/04/16
Integración en redBorder	25/04/16	8/06/16
Desarrollo del cookbook	25/04/16	20/05/16
Adaptación del Client Proxy	23/05/16	24/05/16
Integración en SDK de redBorder	25/05/16	25/05/16
Correcciones y mejoras	23/05/16	8/06/16
Desarrollo de parseadores	8/04/16	7/06/16
ssh	8/04/16	12/04/16
Apache	12/04/16	14/04/16
Cisco WLC 5500	19/05/16	27/05/16
iptables	27/05/16	7/06/16
Pruebas del sistema	8/03/16	15/06/16
Documentación (memoria)	4/04/16	27/06/16

Figura 10-1. Diagrama de Gantt del proyecto.

11 CONCLUSIONES

Nuestra recompensa se encuentra en el esfuerzo y no en el resultado. Un esfuerzo total es una victoria completa.

- Mahatma Gandhi -

En esta memoria se ha intentado plasmar el resultado de varios meses de investigación, aprendizaje, superación personal, trabajo y en definitiva, esfuerzo. A lo largo de este periodo he podido aprender sobre tecnologías tan interesantes como Rsyslog, Kafka, Chef y otras que sin duda serán de gran ayuda en un futuro próximo.

No podemos olvidar la experiencia adquirida al haber trabajado durante este tiempo en el equipo de redBorder. Durante este periodo he trabajado como un componente más del equipo, asistiendo a reuniones semanales o quincenales de seguimiento y objetivos sobre mi proyecto. He podido aprender cómo una empresa como redBorder, que desarrolla un producto Open Source, se divide en distintos equipos de trabajo enfocados en distintas áreas de este producto y, además, cómo estos equipos se coordinan entre sí para obtener un producto consistente.

Una vez terminado el proyecto, podemos decir que se ha conseguido configurar e integrar en la plataforma redBorder las herramientas necesarias para recolectar logs de diferentes fuentes, parsear y normalizar estos logs, y hacerlos llegar al manager de redBorder en formato JSON. La sensación es que se han cumplido los objetivos inicialmente propuestos lo que produce en mí una gran satisfacción personal.

11.1 Líneas de desarrollo

La principal línea de desarrollo será abordar la programación de la interfaz web para redBorder Vault. Así mismo, también será necesario adaptar los mecanismos de indexado, búsqueda y filtrado del Manager de redBorder para que la interfaz web sea capaz de procesar los datos y presentarlos al usuario.

Otra posibilidad sería centrarse en el desarrollo de más parseadores para otros servicios interesantes como puede ser el servidor web Nginx, servidores de base de datos como MySQL o MongoDB, etc.

REFERENCIAS

[1] A. Chuvakin, K. Schmidt and C. Phillips, Logging and Log Management, Waltham, MA: Syngress, 2013.

[2] Wikipedia, «Syslog,» [En línea]. Available: https://en.wikipedia.org/wiki/Syslog.

[3] B. Schwartz, «Google: Log File Analysis Is So Underrated,» 12 Abril 2016. [En línea]. Available: https://www.seroundtable.com/google-log-file-analysis-21919.html.

[4] C. Schroder, «What the Heck is --Mark--? Learn Linux Logging,» 5 Julio 2005. [En línea]. Available: http://www.enterprisenetworkingplanet.com/netsysm/article.php/3517611/What-the-Heck-is-Mark--Learn-Linux-Logging.htm.

[5] A. Echeverri y S. Hussain, «Troubleshooting with Linux Logs,» [En línea]. Available: http://www.loggly.com/ultimate-guide/troubleshooting-with-linux-logs/.

[6] «RFC4765 - The Intrusion Detection Message Exchange Format (IDMEF),» Marzo 2007. [En línea]. Available: https://tools.ietf.org/html/rfc4765.

[7] C. Lonvick, «RFC 3164 - The BSD syslog Protocol,» Agosto 2001. [En línea]. Available: https://tools.ietf.org/html/rfc3164.

[8] R. Gerhards, «RFC 5424 - The Syslog protocol,» Marzo 2009. [En línea]. Available: https://tools.ietf.org/html/rfc5424.

[9] A. Okmianski, «RFC 5426 - Transmission of Syslog Messages over UDP,» Marzo 2009. [En línea]. Available: https://tools.ietf.org/html/rfc5426.

[10] R. Gerhards y C. Lonvick, «RFC 6587 - Transmission of Syslog Messages over TCP,» Abril 2012. [En línea]. Available: https://tools.ietf.org/html/rfc6587.

[11] F. Miao y J. Salowey, «RFC 5425 - Transport Layer Security (TLS) Transport Mapping for Syslog,» Marzo 2009. [En línea]. Available: https://tools.ietf.org/html/rfc5425.

[12] IANA, «Syslog parameters,» [En línea].

[13] G. Klyne y C. Newman, «RFC 3339 - Date and Time on the Internet: Timestamps,» Julio 2002. [En línea]. Available: https://www.ietf.org/rfc/rfc3339.txt.

[14] P. Mockapetris, «RFC 1034 - Domain Names - Concepts and Facilities,» Noviembre 1987. [En línea]. Available: https://tools.ietf.org/html/rfc1034.

[15] R. Gerhards, «Rsyslog Web,» [En línea]. Available: www.rsyslog.com.

[16] R. Gerhards, «Documentación Rsyslog v8,» [En línea]. Available: http://www.rsyslog.com/doc/v8-stable/index.html.

[17] «Documentación Kafka,» [En línea]. Available: http://kafka.apache.org/documentation.html.

[18] «Documentación Zookeeper,» [En línea]. Available: https://zookeeper.apache.org/doc/trunk/.

[19] «Documentación Chef,» [En línea]. Available: https://docs.chef.io/.

[20] R. Gerhards, «Encrypting Syslog Traffic with TLS - Rsyslog Doc,» 3 Julio 2008. [En línea]. Available: http://www.rsyslog.com/doc/v8-stable/tutorials/tls_cert_summary.html.

[21] A. M. Iljushkin, «How to install zookeeper as service on centos 6,» 2014. [En línea]. Available: http://positivealex.github.io/blog/posts/how-to-install-zookeeper-as-service-on-centos/.

[22] E. Nemeth, G. Snyder, T. R. Hein y B. Whaley, Unix and Linux system administration handbook, Michigan: Prentice Hall, 2010.

[23] «syslog(3) - Linux man page,» [En línea]. Available: http://linux.die.net/man/3/syslog.

ÍNDICE DE CONCEPTOS

Attributes 36
Chef 35
client proxy 40
Collector 17
cookbook 36
filtrado 14
Kafka 32
log 5
logging 2
manager 39
Normalización 14
Originator 17
Parsear 15
Recipes 36
Relay 17
Rsyslog 29
sensores 40
Templates 36
topic 32
Transport receiver 18
Transport sender 18
workstation 35
Zookeeper 34

GLOSARIO

ANSI: American National Standars Institute (Instituto Nacional Estadounidense de Estándares). 10

ICMP: Internet Control Message Protocol (Protocolo de Mensajes de Control de Internet). 7

IDS: Intrusion Detection System (Sistema de Detección de Intrusos). 7

IPS: Intrusion Prevention System (Sistema de Prevención de Intrusos). 7

ISO: International Organization for Standardization (Organización Internacional de Normalización) 10

ISP: Internet Service Provider (Proveedor de Servicios de Internet). 6

JSON: JavaScript Object Notation 15

RFC: Request For Comments 10

SNMP: Simple Network Management Protocol (Protocolo Simple de Gestión de Red). 13

TCP: Transmission Control Protocol (Protocolo de Control de Transmisión) 13

UDP: User Datagram Protocol (Protocolo de Datagrama de Usuario). 13

UUID: Universally Unique Identifier (Identificador Único Universal). 90

VPN: Virtual Private Network (Red Privada Virtual). 12

ANEXO A: REDBORDER CLIENT PROXY – INSTALACIÓN Y CONFIGURACIÓN INICIAL

En este anexo mostraremos paso por paso el proceso para instalar un Client Proxy de redBorder desde cero y como reclamarlo en nuestra cuenta de redBorder Live para que mande datos a nuestro manager. Para que sea más fácil de entender nos apoyaremos en múltiples capturas del proceso.

El Client Proxy es una versión modificada de la versión 6.5 de CentOS. El primer paso será descargar la imagen ISO del Client Proxy desde http://download.redborder.net/isos/redBorder-latest-proxy.iso . Una vez termine la descarga de la imagen ISO tendremos que grabarla en un DVD si vamos a instalar en un equipo con lector de DVDs o pasar la ISO a nuestro anfitrión de virtualización si vamos a hacer uso de este.

A continuación, encenderemos el equipo arrancando desde el lector de DVDs y obtendremos una pantalla en la que se nos muestra las opciones del instalador del Client Proxy. Podemos verlo en la siguiente captura:

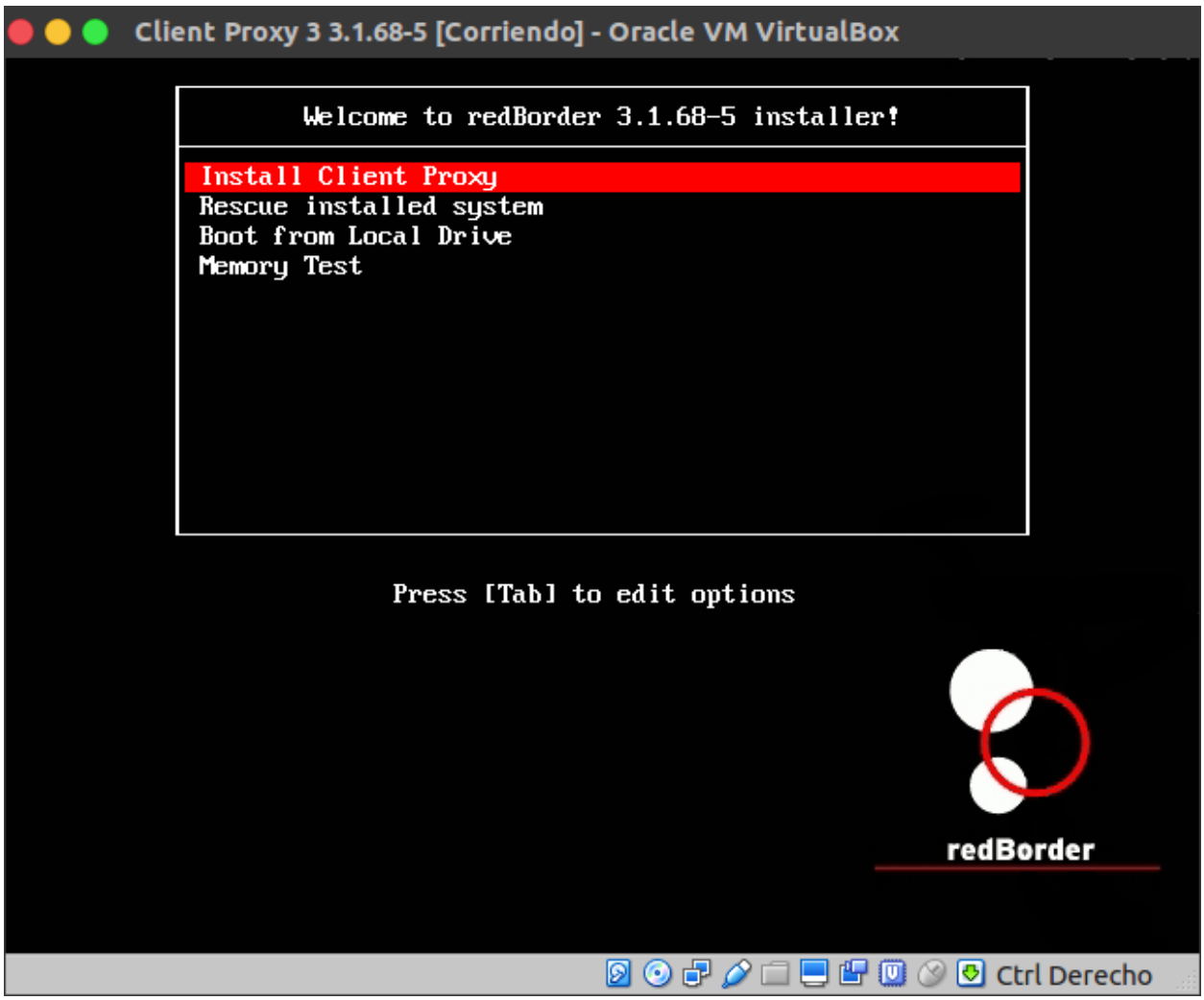


Figura A-1. Opciones del instalador de redBorder Client Proxy.

En esta pantalla deberemos seleccionar la opción “Install Client Proxy”, lo que nos llevará directamente a la pantalla mostrada en la figura A-2, donde deberemos configurar varias opciones:

- La partición donde se desea instalar el Client Proxy.
- La contraseña para el usuario root. Se nos pedirá por seguridad que la repitamos.
- La IP que tendrá el Client Proxy. Por defecto (sin introducir nada) se usará el protocolo DHCP para obtener esta IP.
- La URL de la cloud donde está alojado nuestro manager. Por defecto se usará redBorderLive.

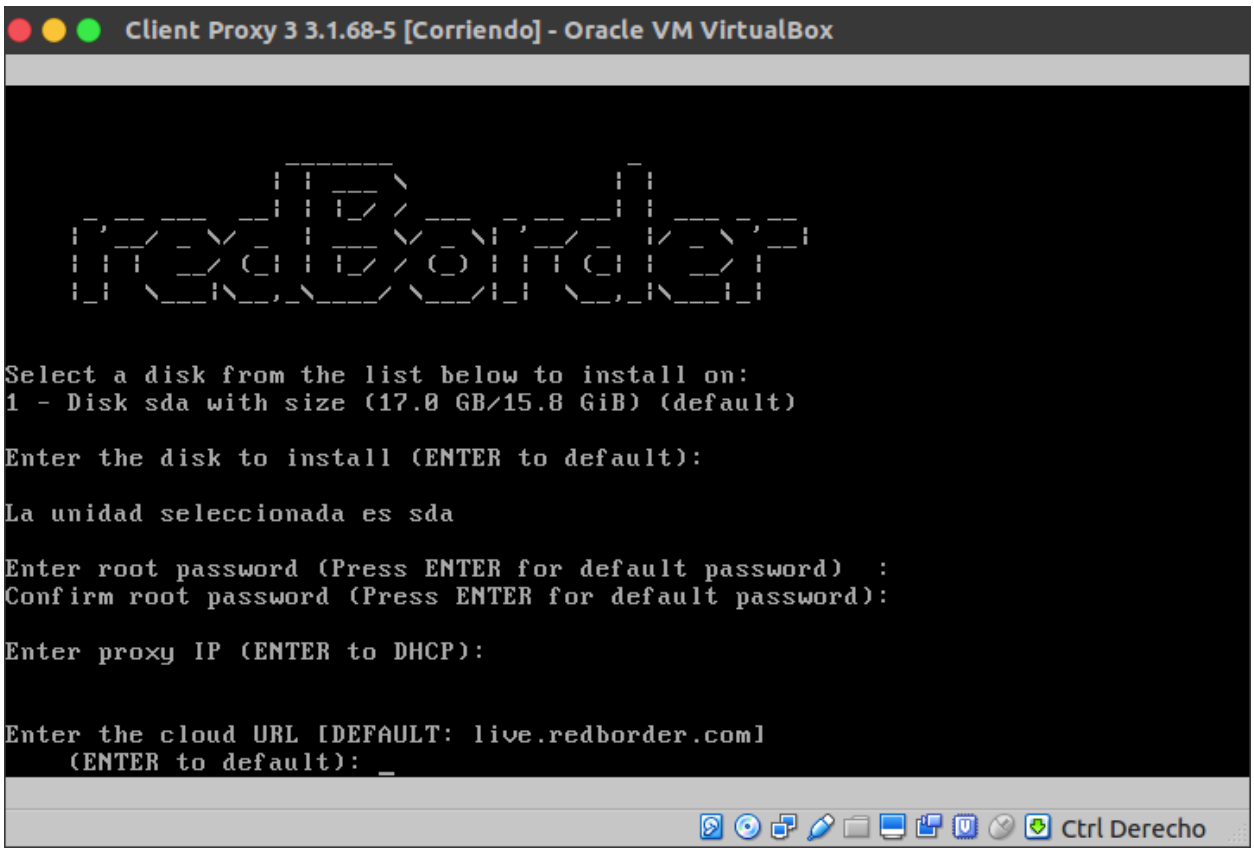


Figura A-2. Configuración de instalación de redBorder Client Proxy.

Una vez hayamos terminado de configurar las opciones, el resto de la instalación terminará de forma automática. Pasaremos por tres etapas antes de terminar por completo el proceso de instalación: verificación de dependencias de los paquetes a instalar, instalación de los paquetes y ejecución de scripts post-instalación. En las figuras A-3, A-4 y A-5, se muestran capturas de estas tres etapas respectivamente.

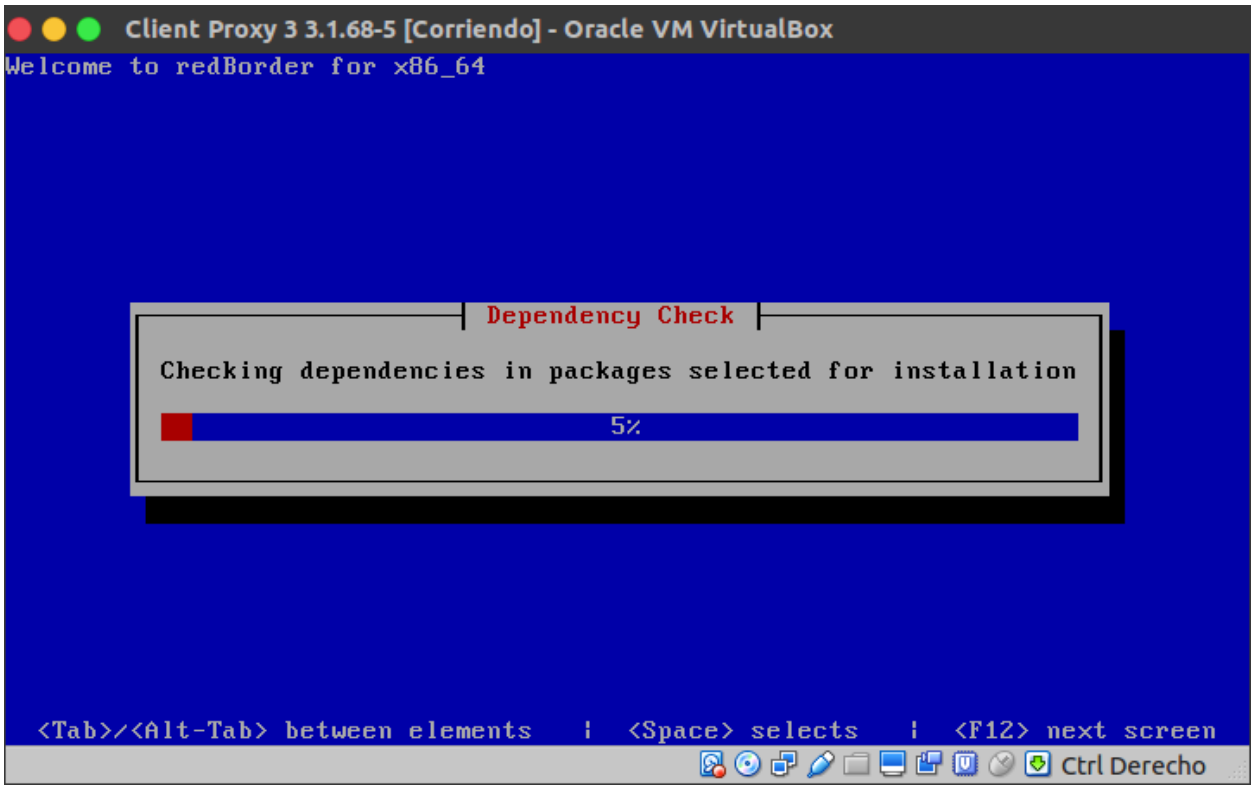


Figura A-3. Etapa de verificación de dependencias.

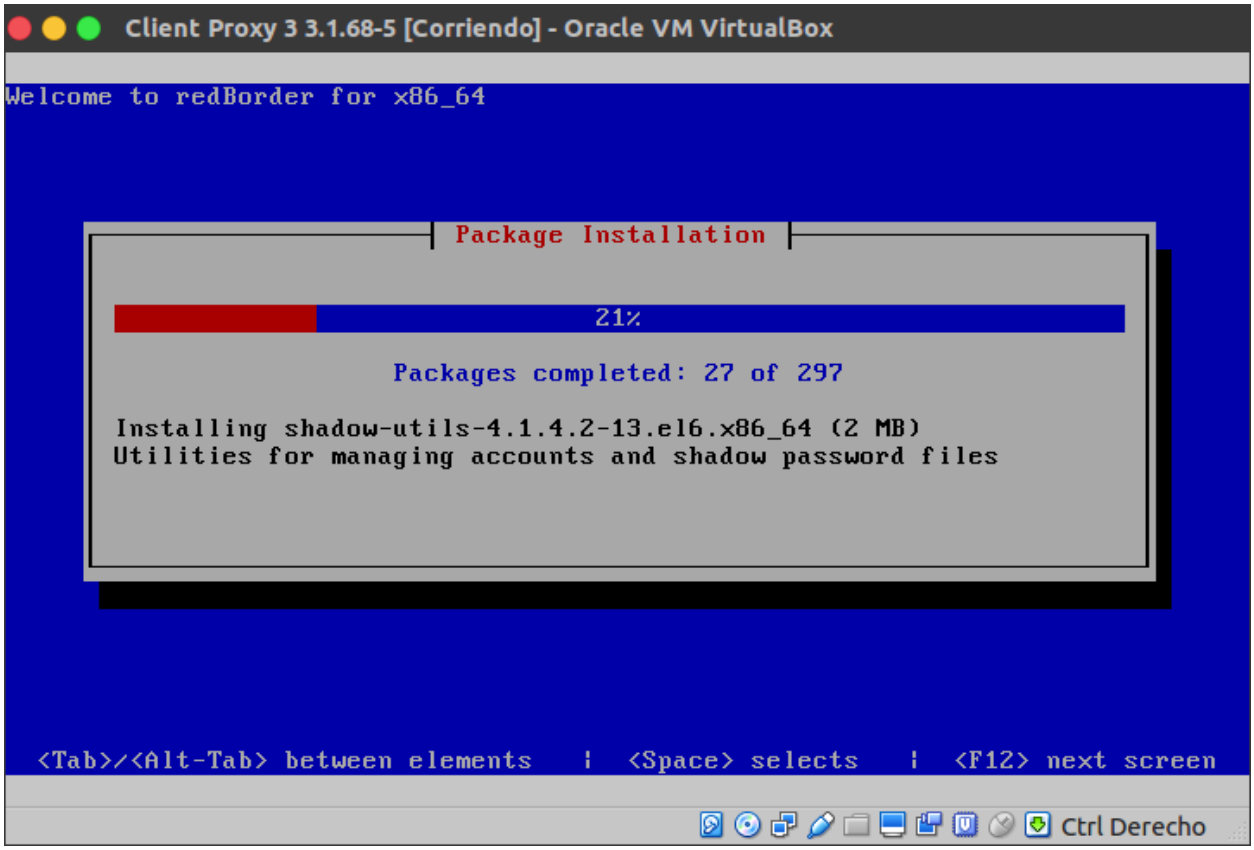


Figura A-4. Etapa de instalación de paquetes.

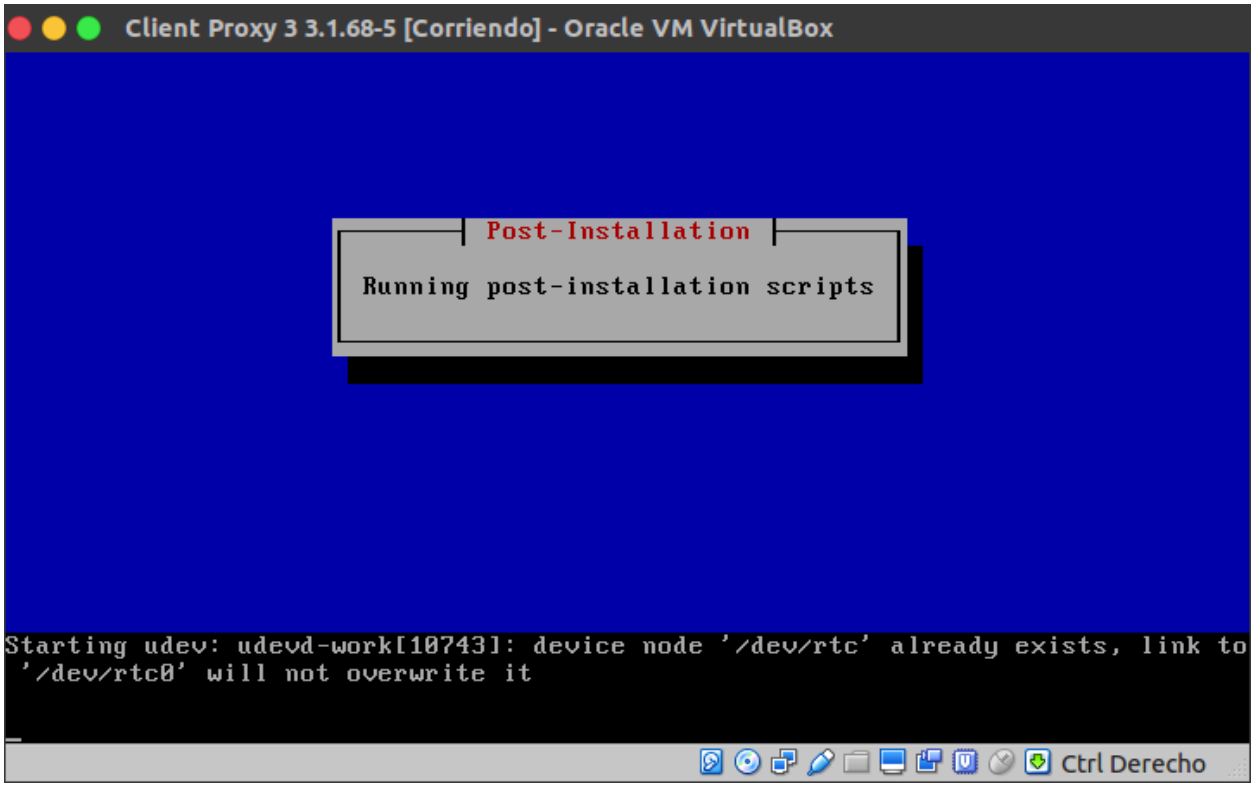


Figura A-5. Etapa de ejecución de scripts post-instalación.

Al terminar el proceso de instalación se reiniciará el equipo, momento en el cual podemos aprovechar para retirar la imagen ISO de este. Una vez el equipo se inicie de nuevo nos pedirá que insertemos un usuario y una contraseña (podemos usar la cuenta root con la contraseña que configuramos antes).

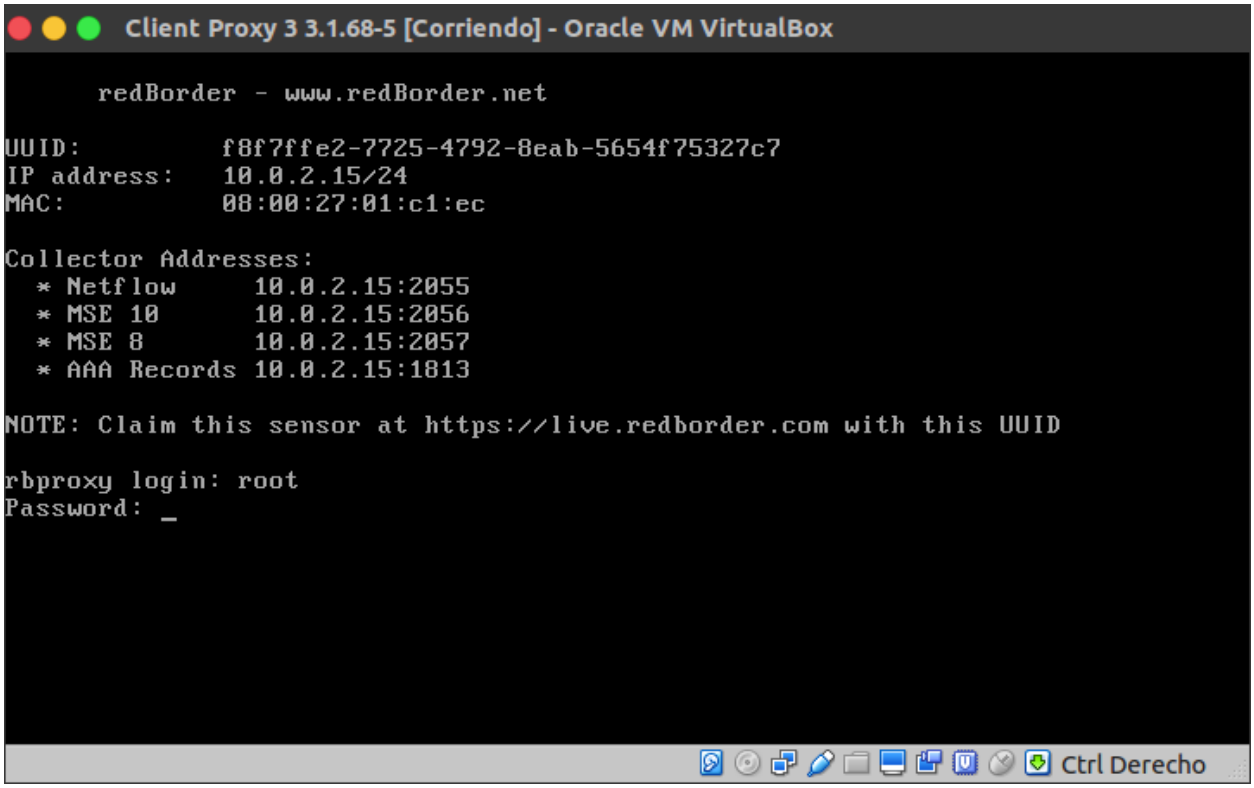


Figura A-6. Información mostrada al iniciar el Client Proxy.

Como se puede observar en la figura A-6, también nos muestra información que puede ser de utilidad como la IP del Client Proxy, la dirección MAC, algunos puertos de procesos colectores y, lo más importante, el UUID.

El UUID es una cadena generada de forma aleatoria y compuesta de 32 dígitos hexadecimales que nos permitirá identificar de forma unívoca. Como dice la nota que se muestra en la figura A-6, tendremos que usar este UUID para reclamar el Client Proxy en nuestro manager, que en este caso lo tenemos disponible en https://live.redborder.com .

Para reclamar nuestro Client Proxy deberemos iniciar sesión con nuestras credenciales en nuestro manager. Una vez hecho esto, en la barra superior donde se muestran las distintas vistas disponibles en el manager, deberemos seleccionar la opción "Sensors", lo que nos llevará a la siguiente pantalla:

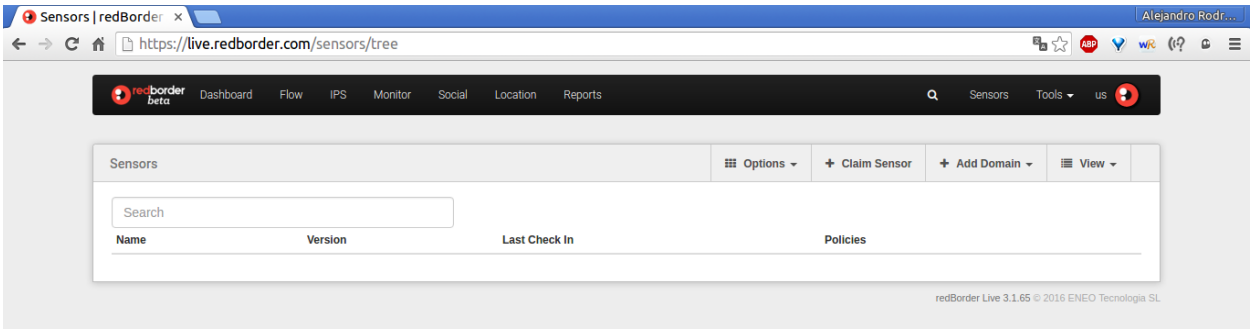


Figura A-7. Apartado "Sensors" del manager de redBorder.

Como se puede apreciar en la figura A-7, aún no hemos reclamado ningún sensor ni Client Proxy en este manager, por lo que la lista de sensores nos aparece vacía. Para reclamar nuestro Client Proxy será necesario pulsar en la opción "Claim Sensor" que aparece en la barra de título "Sensors". Al pulsarlo se nos mostrará el siguiente diálogo emergente:

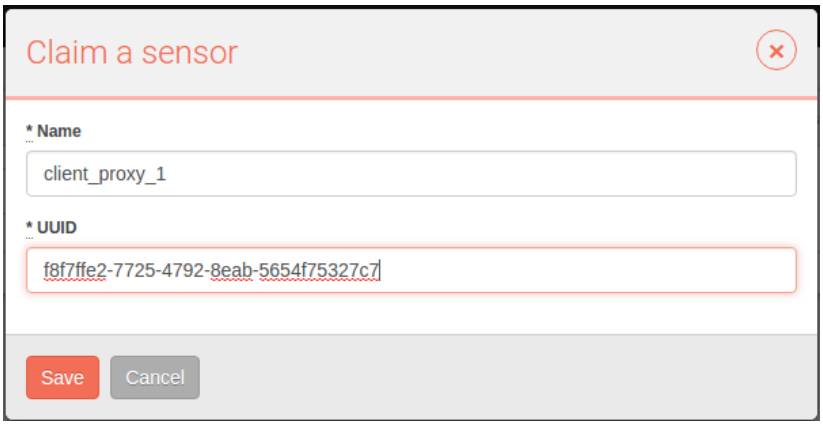


Figura A-8. Diálogo emergente "Claim a sensor".

Deberemos dar un nombre a nuestro Client Proxy y especificar el UUID que se nos facilitó tras la instalación del mismo. Cuando pulsemos el botón "Save" nos aparecerá nuestro Client Proxy en la lista de sensores que antes se encontraba vacía:

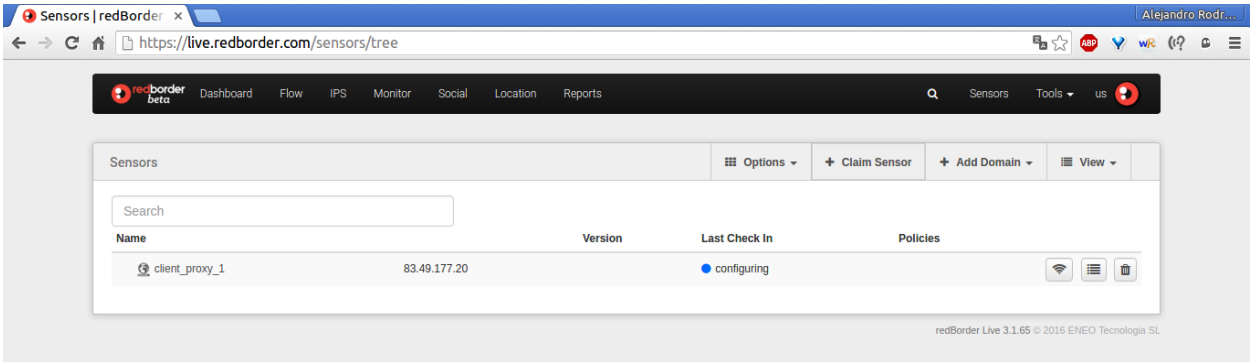


Figura A-9. Apartado "Sensors" del manager de redBorder (2).

Como podemos ver en la figura anterior, efectivamente la lista de sensores se ha actualizado y en la fila correspondiente a nuestro Client Proxy nos aparece un punto azul indicando el mensaje "configuring". Esto quiere decir que el Client Proxy se ha conectado correctamente al manager, pero como es la primera vez que se conecta, el Client Proxy tiene que descargar el cookbook correspondiente del servidor Chef que se ejecuta en el manager y configurarse conforme a lo indicado en este cookbook.

Este proceso puede durar varios minutos y podremos saber que ha terminado si al recargar la página en nuestro navegador el punto azul ha pasado a verde, cambiando también el mensaje. En la figura A-10 se muestra la sección "Sensors" una vez terminado el proceso de configuración del Client Proxy.

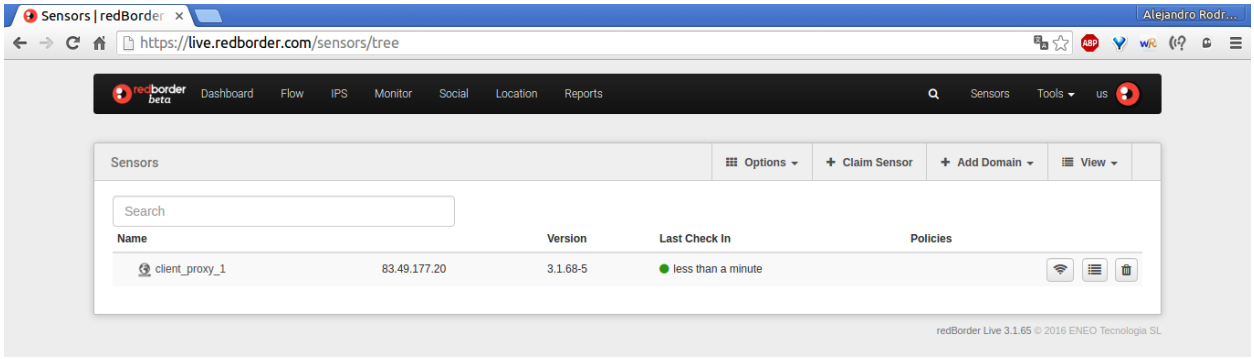


Figura A-10. Apartado "Sensors" del manager de redBorder (3).

Esto nos indica que nuestro Client Proxy ya está configurado y debería ser capaz de enviar datos al manager. Podemos comprobar que el proceso de configuración del Client Proxy ha sido correcto si nos conectamos a este mediante un terminal y ejecutamos el script "rb_get_services.sh". Si el proceso de configuración ha terminado sin problemas, no debería aparecernos el mensaje "not running and it should" en ninguno de los servicios mostrados en la salida de dicho script. La figura A-11 muestra una salida correcta de este script.

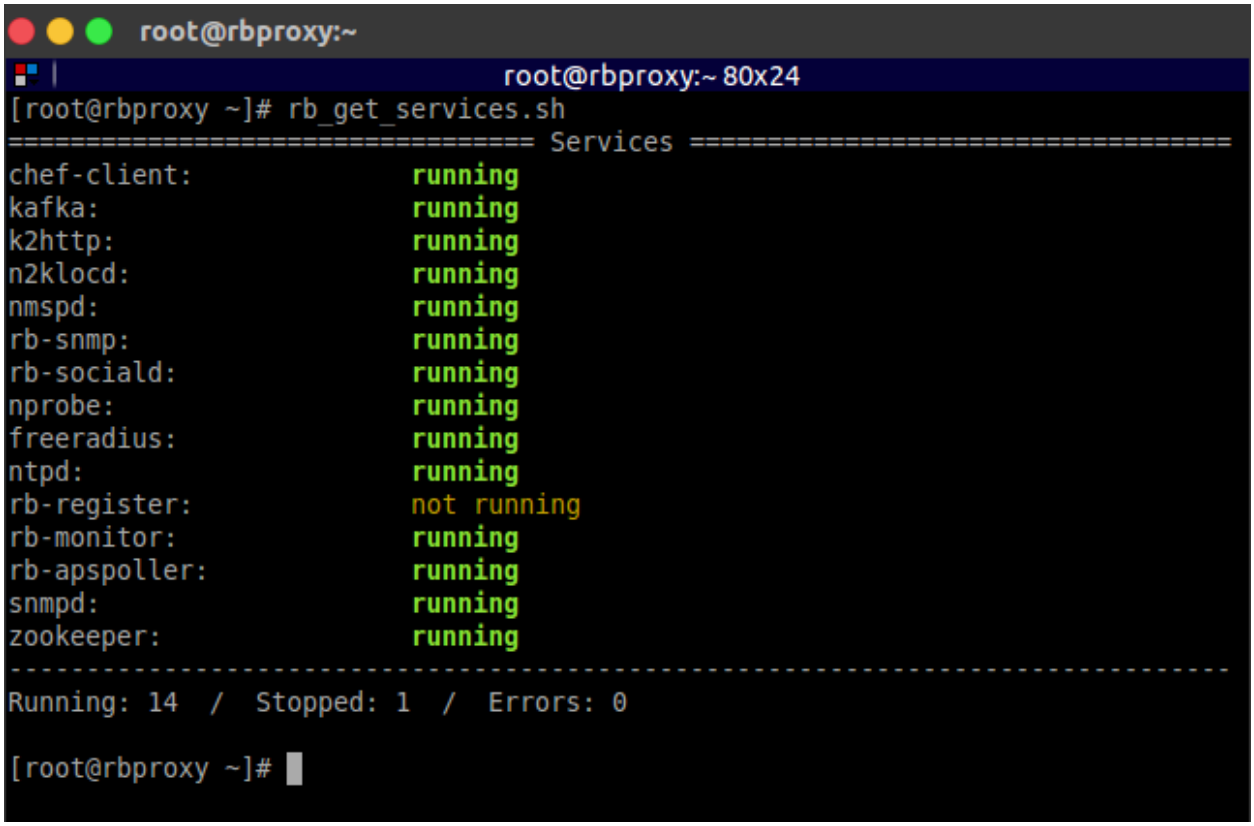


Figura A-11. Ejecución del script "rb_get_services.sh".

Si todo es correcto, nuestro Client Proxy está correctamente instalado y configurado, y preparado para enviar datos al manager. Si entramos en la vista "Monitor", podemos ver que ya está enviando métricas sobre consumo de recursos del equipo que ejecuta el Client Proxy. En la figura A-12 podemos ver una gráfica que representa el porcentaje de memoria usada en el tiempo (los datos válidos son a partir de las 21:05, momento en el que terminó la configuración del Client Proxy). Además de la gráfica, también se nos muestra los valores mínimo, máximo y promedio.

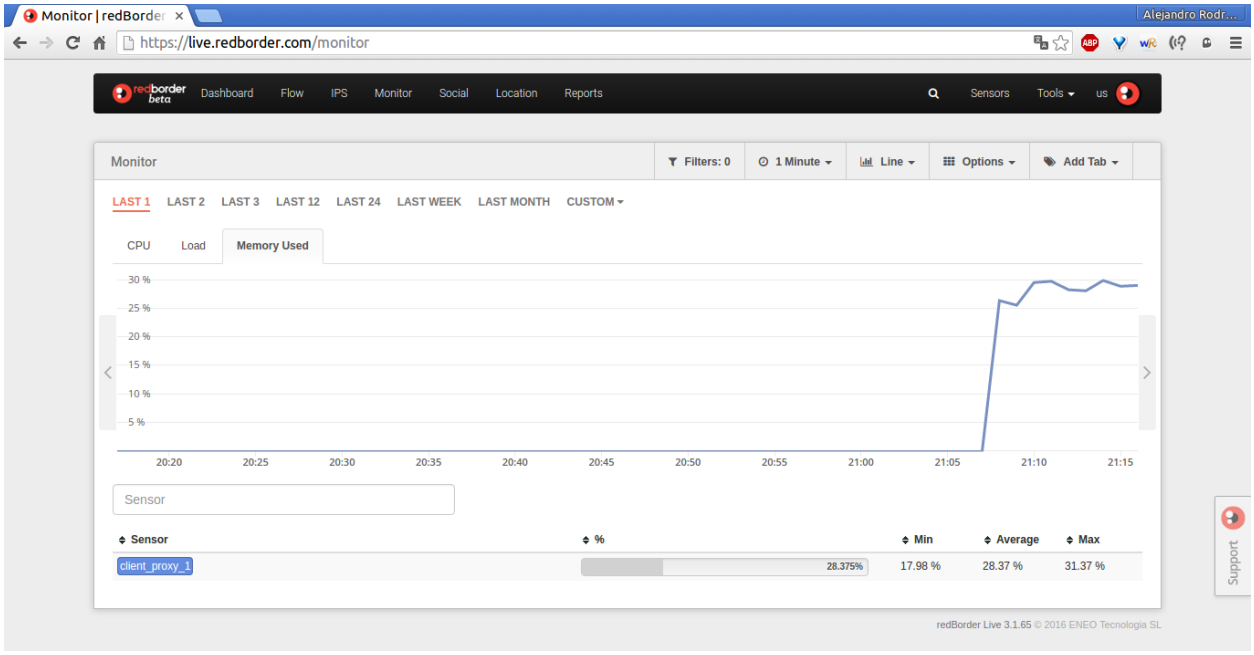


Figura A-12. Vista "Monitor" del manager de redBorder.

ANEXO B: RSYSLOG – INSTALACIÓN

En este Anexo mostraremos el procedimiento para instalar Rsyslog [16] en un sistema Centos. Se comentarán dos formas de instalación: usando el gestor de paquetes "yum" y compilando el código fuente de Rsyslog. La primera opción es mucho más sencilla, pero compilando el código tenemos la opción de personalizar la instalación de modo que sólo se instalen los módulos cuya funcionalidad necesitemos.

También hablaremos sobre como instalar los módulos que han sido necesarios para este proyecto y que no vienen en la instalación por defecto de Rsyslog.

Finalmente se hablará de cómo ejecutar Rsyslog en modo debug, opción que ha sido de gran ayuda en la resolución de problemas a lo largo de este proyecto.

Instalación mediante gestor de paquetes

La instalación mediante el gestor de paquetes "yum" es tan sencilla como ejecutar el siguiente comando con permisos de administrador:

```
yum install rsyslog
```

Esto debería funcionar sin dar problemas, pero en alguna ocasión puede ocurrir que el sistema en el que estamos intentado instalar o actualizar Rsyslog, no tenga los repositorios del mismo actualizados, por lo que no se instalaría la última versión. Este es el caso de la figura B-1, donde se intenta actualizar mediante el gestor de paquetes "yum" pero se nos informa de que el paquete de Rsyslog instalado (versión 7.4.7-12) está actualizado. Sin embargo, en el momento que se tomó esta captura, la versión más reciente de Rsyslog era la 8.19.0-1.

```
[root@localhost ~]# yum install rsyslog
Complementos cargados:fastestmirror
Loading mirror speeds from cached hostfile
 * base: centos.mirror.xtratelecom.es
 * extras: centos.mirror.xtratelecom.es
 * updates: centos.mirror.xtratelecom.es
El paquete rsyslog-7.4.7-12.el7.x86_64 ya se encuentra instalado con su versión
más reciente
Nada para hacer
[root@localhost ~]# _
```

Figura B-1. Actualización de Rsyslog fallida.

Para solucionar esto, debemos actualizar el repositorio de Rsyslog en nuestro sistema para que, de esta forma, el gestor de paquetes sea capaz de encontrar la última versión de Rsyslog. Todo lo que debemos hacer es descargar el archivo descriptor del repositorio de Rsyslog y ponerlo en la carpeta /etc/yum.repos/ para que el gestor "yum" tenga en cuenta este repositorio.

Dicho archivo está disponible en http://rpms.adiscon.com/rsyslogall.repo . En la figura B-2 se muestra la secuencia de comandos necesaria para realizar la descarga del archivo .repo mediante la herramienta "curl" y mover este archivo a la carpeta correspondiente.

```
[root@localhost ~]# curl -O http://rpms.adiscon.com/rsyslogall.repo
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1462  100  1462    0     0   4224      0 --:--:-- --:--:-- --:--:--  4237
[root@localhost ~]# ls -l
total 8
-rw-------. 1 root root 1158 jun  4 19:22 anaconda-ks.cfg
-rw-r--r--. 1 root root 1462 jun  4 19:58 rsyslogall.repo
[root@localhost ~]# mv rsyslogall.repo /etc/yum.repos.d/
[root@localhost ~]# _
```

Figura B-2. Descarga del descriptor de repositorio de Rsyslog.

Una vez realizadas estas acciones, el gestor de paquetes "yum" será capaz de actualizar a la última versión de Rsyslog disponible haciendo uso del mismo comando. En la figura B-3 se puede ver como "yum" nos da la opción de instalar la versión 8.19.0-1 y algunas dependencias extra.

```
Dependencias resueltas

================================================================================
 Package          Arquitectura
                            Versión              Repositorio            Tamaño
================================================================================
Actualizando:
 rsyslog          x86_64      8.19.0-1.el7        rsyslog-v8-stable      636 k
Instalando para las dependencias:
 libfastjson      x86_64      0.99.2-1.el7        rsyslog-v8-stable       61 k
 libgt            x86_64      0.3.11-1.el7        rsyslog-v7-stable       58 k
 liblogging       x86_64      1.0.5-1.el7         rsyslog-v8-stable       25 k

Resumen de la transacción
================================================================================
Instalar               ( 3 Paquetes dependientes)
Actualizar  1 Paquete
```

Figura B-3. Instalación de Rsyslog correcta.

Para verificar que se ha instalado la versión de Rsyslog deseada, podemos hacer uso del siguiente comando:

```
rsyslogd -v
```

La salida de este comando se puede ver en la figura B-4. Además de la versión instalada, nos muestra algunos datos extra sobre la compilación de Rsyslog.

```
[root@localhost ~]# rsyslogd -v
rsyslogd 8.19.0, compiled with:
        PLATFORM:                               x86_64-redhat-linux-gnu
        PLATFORM (lsb_release -d):
        FEATURE_REGEXP:                         Yes
        GSSAPI Kerberos 5 support:              No
        FEATURE_DEBUG (debug build, slow code): No
        32bit Atomic operations supported:      Yes
        64bit Atomic operations supported:      Yes
        memory allocator:                       system default
        Runtime Instrumentation (slow code):    No
        uuid support:                           Yes
        Number of Bits in RainerScript integers: 64

See http://www.rsyslog.com for more information.
[root@localhost ~]# _
```

Figura B-4. Verificación de la versión de Rsyslog.

Instalación mediante compilación de código

La instalación por compilación de código fuente es un poco más compleja debido a la gran cantidad de dependencias que estamos obligados a instalar, y que en la instalación con "yum" se instalan automáticamente. Sin embargo, la instalación mediante compilación de código nos da mayor libertad a la hora de elegir los componentes que serán instalados junto a Rsyslog.

El primer paso será obtener el código fuente de la última versión de Rsyslog (en este caso la versión más reciente era la 8.19.0). El código fuente comprimido está disponible en la dirección web http://www.rsyslog.com/files/download/rsyslog/rsyslog-8.19.0.tar.gz y puede ser descargado haciendo uso de la herramienta "curl". Para descomprimir el código podemos usar la herramienta "tar".

En la figura B-5 se muestra la secuencia de comandos necesaria para descargar el archivo comprimido con el código fuente y descomprimirlo.

```
[root@localhost ~]# curl -O http://www.rsyslog.com/files/download/rsyslog/rsyslo
g-8.19.0.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2226k  100 2226k    0     0  1739k      0  0:00:01  0:00:01 --:--:-- 1739k
[root@localhost ~]# ls -l
total 2232
-rw-------. 1 root root    1158 jun  4 19:22 anaconda-ks.cfg
-rw-r--r--. 1 root root 2279714 jun  4 23:58 rsyslog-8.19.0.tar.gz
[root@localhost ~]# tar -zxf rsyslog-8.19.0.tar.gz
[root@localhost ~]# ls -l
total 2236
-rw-------.  1 root        root           1158 jun  4 19:22 anaconda-ks.cfg
drwxrwxr-x. 11 arodriguezc arodriguezc    4096 may 30 12:41 rsyslog-8.19.0
-rw-r--r--.  1 root        root        2279714 jun  4 23:58 rsyslog-8.19.0.tar.g
z
[root@localhost ~]# _
```

Figura B-5. Descarga del código fuente de Rsyslog.

El siguiente paso es entrar en el directorio que se crea al descomprimir el código fuente y configurar la instalación de Rsyslog. Para configurar la instalación usaremos el siguiente comando:

```
./configure [options]
```

Las opciones de este comando es lo que nos permite especificar que módulos vamos a añadir a la instalación de Rsyslog. Para comprobar cuáles son las opciones disponibles podemos hacer uso del siguiente comando:

```
./configure --help
```

Pero como paso previo a configurar la instalación, tendremos que instalar las dependencias necesarias para compilar e instalar Rsyslog. A continuación, se muestra la lista de los paquetes necesarias para el caso de una instalación las opciones por defecto. Para instalar estas dependencias se puede hacer uso del comando "yum install nombre_paquete".

```
gcc.x86_64                      libgcrypt.x86_64
make.x86_64                     libgcrypt-devel.x86_64
libestr.x86_64                  openssl.x86_64
libestr-devel.x86_64            openssl-devel.x86_64
libfastjson.x86_64              liblogging.x86_64
libfastjson-devel.x86_64        liblogging-devel.x86_64
zlib.x86_64
zlib-devel.x86_64
libuuid-devel.x86_64
```

En el caso de que no usemos las opciones por defecto al configurar la instalación, existe la posibilidad de que sea necesaria alguna otra dependencia además de las que aparecen en la lista anterior. En este caso, el comando "configure" nos mostrará un mensaje de error como el mostrado en la figura B-6, señalando la dependencia que falta.

```
checking for LIBLOGGING_STDLOG... no
configure: error: Package requirements (liblogging-stdlog >= 1.0.3) were not met
:

No package 'liblogging-stdlog' found

Consider adjusting the PKG_CONFIG_PATH environment variable if you
installed software in a non-standard prefix.

Alternatively, you may set the environment variables LIBLOGGING_STDLOG_CFLAGS
and LIBLOGGING_STDLOG_LIBS to avoid the need to call pkg-config.
See the pkg-config man page for more details.
[root@localhost rsyslog-8.19.0]# _
```

Figura B-6. Mensaje de dependencia no encontrada.

Una vez tengamos todas las dependencias necesarias instaladas, sabremos que se ha configurado correctamente la instalación porque el comando "configure" nos mostrará un resumen de los módulos que han sido habilitados.

En la figura B-7 podemos ver parte de este resumen al ejecutar el comando "configure" con las siguientes opciones:

```
./configure --enable-mmjsonparse --enable-mmpstrucdata
```

```
---{ message modification modules }---
    mmnormalize module will be compiled:      no
    mmjsonparse module will be compiled:      yes
    mmgrok module will be compiled:           no
    mmjaduit module will be compiled:         no
    mmsnmptrapd module will be compiled:      no
    mmutf8fix enabled:                        no
    mmrfc5424addhmac enabled:                 no
    mmpstrucdata enabled:                     yes
    mmsequence enabled:                       no

---{ database support }---
    MySql support enabled:                    no
    libdbi support enabled:                   no
    PostgreSQL support enabled:               no
    mongodb support enabled:                  no
    hiredis support enabled:                  no
```

Figura B-7. Módulos habilitados en la ejecución de "configure".

Si nos fijamos en la sección "message modification modules" podemos ver que los dos módulos (mmjsonparse y mmpstrucdata) que se han especificado en las opciones del comando "configure" han sido habilitados correctamente y por lo tanto serán compilados e instalados junto a Rsyslog.

Una vez tenemos la instalación configurada, el siguiente paso es compilar el código e instalar Rsyslog. Esto se realizará mediante los siguientes comandos:

```
make
make install
```

Instalación de módulos

Cuando realizamos una instalación haciendo uso del gestor de paquetes o compilando el código con las opciones de instalación por defecto, es posible que queramos añadir un módulo de los que ofrece Rsyslog para aprovechar su funcionalidad, como es el caso de este proyecto.

Para instalar estos módulos posteriormente a la instalación de Rsyslog, existen paquetes RPM que nos ofrecen los módulos por separado. Estos paquetes pueden ser instalados fácilmente haciendo uso del gestor de paquetes "yum". A continuación, se muestra una lista de los paquetes de modulos de Rsyslog instalados en este proyecto:

```
gnutls-utilsx86_64
rsyslog-gnutls.x86_64
rsyslog-mmjsonparse.x86_64
rsyslog-mmnormalize.x86_64
rsyslog-kafka.x86_64
```

El resto de módulos que han sido usados en este proyecto y no se encuentran en la lista anterior, están habilitado en la instalación por defecto de Rsyslog (a excepción de mmrfc5424addhmac) y por lo tanto se instalan al mismo tiempo que Rsyslog.

El problema viene cuando queremos hacer uso de un módulo que no se instala por defecto y no cuenta con paquete RPM propio que nos facilite su instalación. En este caso debemos compilarlo para generar un archivo de extensión "so" que deberemos copiar en la carpeta de librerías de nuestra instalación.

El módulo mmrfc5424addhmac es uno de los que cumple esta peculiaridad y que además hemos usado en nuestro proyecto, por lo que a continuación procederemos a explicar cómo compilarlo a partir del código fuente de Rsyslog, donde también se encuentra el código fuente de este módulo.

El primer paso, al igual que en la instalación de Rsyslog mediante compilación del código fuente, será descargar la última versión del código fuente Rsyslog, descomprimirlo y entrar en la carpeta que se crea al descomprimirlo.

Una vez hecho esto deberemos configurar la instalación de Rsyslog para habilitar el módulo mmrfc5424addhmac mediante el siguiente comando:

```
./configure --libdir="/lib64/" --enable-mmrfc5424addhmac
```

La primera de las opciones sirve para indicar el directorio donde se guardan las librerías de las aplicaciones. Al compilar el módulo se creará (en caso de no existir) un directorio de nombre rsyslog dentro del directorio especificado en esta primera opción y, se copiará el módulo mmrfc5424addhmac compilado en este directorio. La segunda opción sirve para habilitar el módulo mmrfc5424addhmac.

Como no queremos compilar e instalar Rsyslog al completo, sino que sólo queremos el módulo mmrfc5424addhmac, debemos cambiar al directorio "contrib/mmrfc5424addhmac" y dentro de este directorio proceder a compilar e instalar el módulo. La secuencia de comandos necesaria es la siguiente:

```
cd contrib/mmrfc5424addhmac
make install
```

Esto compilará el módulo y lo copiará en la carpeta "/lib64/rsyslog/" dejándolo listo para ser usado por Rsyslog.

Ejecución en modo debug

El modo debug de Rsyslog nos permite ejecutar el demonio Rsyslog mostrando los mensajes de depuración en la salida estándar. Gracias a estos mensajes podemos encontrar fácilmente errores en la sintaxis de la configuración de Rsyslog, fallos en la autenticación de certificados que impiden el intercambio de mensajes, etc.

Para ejecutar Rsyslog en modo debug basta con parar el proceso de Rsyslog que se está ejecutando actualmente y ejecutar uno nuevo con la opción "-nd". Para ello será necesario ejecutar los siguientes comandos:

```
service rsyslog stop
rsyslogd -nd
```

ANEXO C: RSYSLOG – GENERACIÓN DE CERTIFICADOS

En este Anexo mostraremos el procedimiento para generar la autoridad certificadora y, a partir de ella, generar los certificados de maquina usando una petición creada por el usuario y que contiene la información necesaria para expedir el certificado en cuestión.

Configuración de la autoridad certificadora

El primer paso es configurar la autoridad certificadora (CA). Esta será la encargada de aprobar la identidad de los distintos hosts. Para ello usará los certificados.

La CA crea un certificado llamado certificado auto firmado, el cual aprueba su propia autenticidad. Esto puede sonar inútil, pero la clave para entenderlo es que cada host recibirá una copia de este certificado, que es una prueba de confianza. Para configurar el certificado proporcionado por la CA, el administrador le dice a rsyslog en que certificados confiar. Es por ello que la clave privada de la autoridad certificadora es tan importante, ya que con esta se pueden generar certificados de confianza para nuestro entorno rsyslog.

Para crear el certificado auto firmado, tenemos que usar los siguientes comandos de la herramienta GnuTLS (es necesario instalar dicha herramienta ya que no viene por defecto en la mayoría de plataformas):

- Generar la clave privada:

```
certtool --generate-privkey --outfile ca-key.pem
```

- Crear el certificado auto firmado:

```
certtool --generate-self-signed --load-privkey ca-key.pem
  --outfile ca.pem
```

Este comando hace una serie de preguntas que tienen que ser respondidas apropiadamente. El periodo de expiración debe ser un valor alto, ya que una vez expire este certificado se deberán expedir todos los certificados de nuevo. Es necesario especificar que el certificado pertenece a una autoridad. El certificado se usa para firmar otros certificados.

A continuación, se muestra un ejemplo de ejecución (las respuestas en blanco significan que se acepta la respuesta por defecto):

```
[root@rgf9dev sample]# certtool --generate-privkey --outfile
  ca-key.pem --bits 2048
Generating a 2048 bit RSA private key...
[root@rgf9dev sample]# certtool --generate-self-signed
  --load-privkey ca-key.pem --outfile ca.pem
Generating a self signed certificate...
Please enter the details of the certificate's distinguished name.
  Just press enter to ignore a field.
```

```
Country name (2 chars): SP
Organization name: redBorder
Organizational unit name: Vault
Locality name: Sevilla
State or province name: Sevilla
Common name: redborder.cluster
UID:
This field should not be used in new certificates.
E-mail:
Enter the certificate's serial number (decimal):

Activation/Expiration time.
The certificate will expire in (days): 3650

Extensions.
Does the certificate belong to an authority? (Y/N): y
Path length constraint (decimal, -1 for no constraint):
Is this a TLS web client certificate? (Y/N):
Is this also a TLS web server certificate? (Y/N):
Enter the e-mail of the subject of the certificate:
  arodriguezc.ext@redborder.com
Will the certificate be used to sign other certificates? (Y/N): y
Will the certificate be used to sign CRLs? (Y/N):
Will the certificate be used to sign code? (Y/N):
Will the certificate be used to sign OCSP requests? (Y/N):
Will the certificate be used for time stamping? (Y/N):
Enter the URI of the CRL distribution point:
X.509 Certificate Information:
    Version: 3
    Serial Number (hex): 485a365e
    Validity:
        Not Before: Thu Jun 19 10:35:12 UTC 2008
        Not After: Sun Jun 17 10:35:25 UTC 2018
    Subject:
  C=SP,O=redBorder,OU=Vault,L=Sevilla,ST=Sevilla,CN=redborder.clust
  er
    Subject Public Key Algorithm: RSA
        Modulus (bits 2048):
            d9:9c:82:46:24:7f:34:8f:60:cf:05:77:71:82:61:66
            05:13:28:06:7a:70:41:bf:32:85:12:5c:25:a7:1a:5a
            28:11:02:1a:78:c1:da:34:ee:b4:7e:12:9b:81:24:70
            ff:e4:89:88:ca:05:30:0a:3f:d7:58:0b:38:24:a9:b7
            2e:a2:b6:8a:1d:60:53:2f:ec:e9:38:36:3b:9b:77:93
            5d:64:76:31:07:30:a5:31:0c:e2:ec:e3:8d:5d:13:01
            11:3d:0b:5e:3c:4a:32:d8:f3:b3:56:22:32:cb:de:7d
            64:9a:2b:91:d9:f0:0b:82:c1:29:d4:15:2c:41:0b:97
        Exponent:
            01:00:01
    Extensions:
        Basic Constraints (critical):
            Certificate Authority (CA): TRUE
        Subject Alternative Name (not critical):
            RFC822name: arodriguezc.ext@redborder.com
        Key Usage (critical):
            Certificate signing.
```

```
      Subject Key Identifier (not critical):
            fbfe968d10a73ae5b70d7b434886c8f872997b89
Other Information:
    Public Key Id:
        fbfe968d10a73ae5b70d7b434886c8f872997b89

Is the above information ok? (Y/N): y
Signing certificate...
[root@rgf9dev sample]# chmod 400 ca-key.pem
[root@rgf9dev sample]# ls -l
total 8
-r-------- 1 root root  887 2008-06-19 12:33 ca-key.pem
-rw-r--r-- 1 root root 1029 2008-06-19 12:36 ca.pem
[root@rgf9dev sample]#
```

Es muy importante mantener la clave privada (ca-key.pem) segura. Nadie excepto la autoridad certificadora necesita tenerla. Si alguien más consiguiese esta clave, la seguridad de nuestro entorno rsyslog estaría rota.

Expedición de certificados de máquina

En este paso generamos los certificados para cada uno de los hosts. Estos certificados identificarán cada host al host remoto con el que se comunica. El nombre DNS especificado en el certificado puede ser a su vez especificado en la configuración de rsyslog como un host permitido.

El administrador del host para el cual se va a generar el certificado debe generar una clave privada la cual solo puede ser accedida por dicho host. Con esta clave privada el administrador del host generará una petición de certificado que se enviará a la autoridad certificadora, que será la encargada de generar el certificado (contiene la clave pública). Por último, la CA enviará el certificado de vuelta al host y este lo instalará.

El procedimiento con la herramienta GnuTLS es el siguiente:

- El administrador del host genera una clave privada y a partir de esta genera una petición de certificado que posteriormente enviará a la autoridad certificadora:

```
[root@rgf9dev sample]# certtool --generate-privkey --outfile
  key.pem --bits 2048
Generating a 2048 bit RSA private key...
[root@rgf9dev sample]# certtool --generate-request --load-privkey
  key.pem --outfile request.pem
Generating a PKCS #10 certificate request...
Country name (2 chars): SP
Organization name: redBorder
Organizational unit name: Vault
Locality name: Sevilla
State or province name: Sevilla
Common name: machine.example.net
UID:
Enter a dnsName of the subject of the certificate:
Enter the IP address of the subject of the certificate:
Enter the e-mail of the subject of the certificate:
Enter a challange password:
Does the certificate belong to an authority? (y/N): n
Will the certificate be used for signing (DHE and RSA-EXPORT
  ciphersuites)? (y/N):
Will the certificate be used for encryption (RSA ciphersuites)?
  (y/N):
```

```
Is this a TLS web client certificate? (y/N): y
Is this also a TLS web server certificate? (y/N): y
```

- La entidad certificadora genera el certificado haciendo uso de su clave privada y lo envía de vuelta al host:

```
[root@rgf9dev sample]# certtool --generate-certificate
  --load-request request.pem --outfile cert.pem
  --load-ca-certificate ca.pem --load-ca-privkey ca-key.pem
Generating a signed certificate...
Enter the certificate's serial number (decimal):


Activation/Expiration time.
The certificate will expire in (days): 1000


Extensions.
Do you want to honour the extensions from the request? (y/N):
Does the certificate belong to an authority? (Y/N): n
Is this a TLS web client certificate? (Y/N): y
Is this also a TLS web server certificate? (Y/N): y
Enter the dnsName of the subject of the certificate:
  machine.example.net
Enter the IP address of the subject of certificate:
Will the certificate be used for signing (DHE and RSA-EXPORT
  ciphersuites)? (Y/N):
Will the certificate be used for encryption (RSA ciphersuites)?
  (Y/N):
X.509 Certificate Information:
    Version: 3
    Serial Number (hex): 485a3819
    Validity:
        Not Before: Thu Jun 19 10:42:54 UTC 2008
        Not After: Wed Mar 16 10:42:57 UTC 2011
    Subject:
  C=SP,O=redBorder,OU=Vault,L=Sevilla,ST=Sevilla,CN=machine.example
  .net
    Subject Public Key Algorithm: RSA
        Modulus (bits 2048):
            b2:4e:5b:a9:48:1e:ff:2e:73:a1:33:ee:d8:a2:af:ae
            2f:23:76:91:b8:39:94:00:23:f2:6f:25:ad:c9:6a:ab
            2d:e6:f3:62:d8:3e:6e:8a:d6:1e:3f:72:e5:d8:b9:e0
            d0:79:c2:94:21:65:0b:10:53:66:b0:36:a6:a7:cd:46
            1e:2c:6a:9b:79:c6:ee:c6:e2:ed:b0:a9:59:e2:49:da
            c7:e3:f0:1c:e0:53:98:87:0d:d5:28:db:a4:82:36:ed
            3a:1e:d1:5c:07:13:95:5d:b3:28:05:17:2a:2b:b6:8e
            8e:78:d2:cf:ac:87:13:15:fc:17:43:6b:15:c3:7d:b9
        Exponent:
            01:00:01
    Extensions:
        Basic Constraints (critical):
            Certificate Authority (CA): FALSE
```

```
        Key Purpose (not critical):
            TLS WWW Client.
            TLS WWW Server.
        Subject Alternative Name (not critical):
            DNSname: machine.example.net
        Subject Key Identifier (not critical):
            0ce1c3dbd19d31fa035b07afe2e0ef22d90b28ac
        Authority Key Identifier (not critical):
            fbfe968d10a73ae5b70d7b434886c8f872997b89
Other Information:
    Public Key Id:
        0ce1c3dbd19d31fa035b07afe2e0ef22d90b28ac

Is the above information ok? (Y/N): y


Signing certificate...
[root@rgf9dev sample]#
```

- El host ya dispone del certificado que lo identifica listo para ser usado en rsyslog.

Estos pasos hay que repetirlos para cada host, ya sea Client Proxy o un sensor que envíe logs.

ANEXO D: RSYSLOG – CONFIGURACIÓN DEL CLIENT PROXY

En este anexo mostraremos los archivos completos que forman parte de la configuración de Rsyslog desarrollada para el Client Proxy de redBorder y cuyos aspectos más interesantes fueron explicados en el capítulo 6. También se mostrarán los archivos de definición de reglas de parseo de Liblognorm, usados para parsear los mensajes Syslog que nos llegan por la red.

Previamente a mostrar dichos archivos, se muestra un esquema en forma de árbol de directorio para facilitar la localización de cada uno de estos archivos:

```
/
|-- etc
    |-- rsyslog.conf
    `-- rsyslog.d
        |-- 01-server.conf
        |-- 02-general.conf
        |-- 03-parse_ssh.conf
        |-- 04-parse_apacheaccess.conf
        |-- 05-parse_iptables.conf
        |-- 06-parse_Cisco_WLC.conf
        |-- 99-parse_rfc5424.conf
        `-- rules
            |-- Cisco_WLC.rules
            |-- iptables.rules
            `-- ssh.rules
```

Ficheros de configuración de Rsyslog

/etc/rsyslog.conf

```
#################
#### MÓDULOS ####
#################

$ModLoad imuxsock # proporciona soporte a logging local

###############################
#### DIRECTIVAS GLOBALES ####
###############################

# Usar el formato de timestamp tradicional por defecto
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat
```

```
# Filtrar mensajes duplicados
$RepeatedMsgReduction on


# Permisos por defectos para los archivos donde se guardarán los
# logs.
$FileOwner syslog
$FileGroup syslog
$FileCreateMode 0640
$DirCreateMode 0755
$Umask 0022
$PrivDropToUser syslog
$PrivDropToGroup syslog

#
# Directorio de trabajo donde se almacenarán las colas que
# continen los logs que van a ser procesados.
$WorkDirectory /var/spool/rsyslog

#
# Incluir todos los archivos de configuración que se encuentren en
# el directorio /etc/rsyslog.d/
$IncludeConfig /etc/rsyslog.d/*.conf
```

/etc/rsyslog.d/01-server.conf

```
#Configuración TCP
module(load="imtcp")

#Descomentar estas líneas para usar TLS
#$DefaultNetstreamDriver gtls

#$DefaultNetstreamDriverCAFile /etc/rsyslog.d/certificates/ca.pem
#$DefaultNetstreamDriverCertFile
  /etc/rsyslog.d/certificates/cert.pem
#$DefaultNetstreamDriverKeyFile
  /etc/rsyslog.d/certificates/key.pem

#$ActionSendStreamDriverMode 1
#$ActionSendStreamDriverAuthMode x509/name
#$ActionSendStreamDriverPermittedPeer *.example.net

input(type="imtcp" port="514")

#Configuración UDP
module(load="imudp")
```

/etc/rsyslog.d/02-general.conf

```
#Cargar los módulos necesarios.
module(load="omkafka")
module(load="mmjsonparse")
module(load="mmpstrucdata")
module(load="mmnormalize")
module(load="mmrfc5424addhmac")

#Acciones comunes a todos los mensajes.
action(type="mmrfc5424addhmac" key="yourenterprisekey"
  hashFunction="sha256" sd_id="hash@39483")
action(type="mmpstrucdata")
```

/etc/rsyslog.d/03-parse_ssh.conf

```
# 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.

####Plantilla para los logs del servicio ssh####
template(name="norm_ssh"
    type="list"){
      constant(value="{")
      constant(value="\"raw-message\":\"")
         property(name="rawmsg" format="json")
      constant(value="\",\"message\":\"")
         property(name="msg" format="json")

      constant(value="\",\"pri\":\"")
         property(name="pri" format="json")
      constant(value="\",\"pri-text\":\"")
         property(name="pri-text" 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="\",\"protocol-version\":\"")
         property(name="protocol-version" format="json")
      constant(value="\",\"timestamp\":\"")
         property(name="timestamp" dateFormat="unixtimestamp")
      constant(value="\",\"hostname\":\"")
         property(name="hostname" format="json")
      constant(value="\",\"fromhost-ip\":\"")
         property(name="fromhost-ip" 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="\",\"input-module\":\"")
               property(name="inputname" format="json")
            constant(value="\",\"proxy-uuid\":\"")
               constant(value="fd6fc5e1-e30a-48f1-a109-d4af0ddcec01")
            constant(value="\",\"parsed-info\":")
               property(name="$!")
                    constant(value="}")
      }

   if $app-name == 'sshd' and $fromhost-ip != '127.0.0.1' then
   {
     #Extracción campos mediante reglas, usando el módulo liblognorm
     action(type="mmnormalize"
     ruleBase="/etc/rsyslog.d/rules/ssh.rules")

     #Copia local en un fichero.
   & action(     type="omfile"
            file="/var/log/remote/norm_ssh.log"
            template="norm_ssh"
            )

     #Acción para enviar los logs a Apache Kafka
   & action(type="omkafka" topic="rb_vault" broker="localhost:9092"
            template="norm_ssh" )
     stop
   }
```

/etc/rsyslog.d/04-parse_apacheaccess.conf

```
# En este fichero encontramos la configuración necesaria para
# recolectar logs de Apache y enviarlos a Kafka en formato JSON.

####Plantillas para logs de Apache con formato JSON####
template(name="norm_apacheaccess" type="list"){
        constant(value="{")
        constant(value="\"raw-message\":\"")
           property(name="rawmsg" format="json")
        constant(value="\",\"message\":\"")
           property(name="msg" format="json")

        constant(value="\",\"pri\":\"")
           property(name="pri" format="json")
        constant(value="\",\"pri-text\":\"")
           property(name="pri-text" 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="\",\"protocol-version\":\"")
        property(name="protocol-version" format="json")
     constant(value="\",\"timestamp\":\"")
        property(name="timestamp" dateFormat="unixtimestamp")
     constant(value="\",\"hostname\":\"")
        property(name="hostname" format="json")
     constant(value="\",\"fromhost-ip\":\"")
        property(name="fromhost-ip" 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="\",\"structured-data\":\"")
        property(name="structured-data" format="json")
     constant(value="\",\"tag\":\"")
        property(name="$!rfc5424-sd!rbvault@39483!tag"
        format="json")
     constant(value="\",\"sensor-uuid\":\"")
        property(name="$!rfc5424-sd!rbvault@39483!sensor-uuid"
        format="json")
     constant(value="\",\"hash\":\"")
        property(name="$!rfc5424-sd!hash@39483!hash" format="json")

     constant(value="\",\"inputname\":\"")
        property(name="inputname" format="json")

     constant(value="\",\"proxy-uuid\":\"")
        constant(value="fd6fc5e1-e30a-48f1-a109-d4af0ddcec01")


     constant(value="\",\"access-time\":\"")
        property(name="$!time" format="json")
     constant(value="\",\"remoteIP\":\"")
        property(name="$!remoteIP" format="json")
     constant(value="\",\"host\":\"")
        property(name="$!host" format="json")
     constant(value="\",\"request\":\"")
        property(name="$!request" format="json")
     constant(value="\",\"query\":\"")
        property(name="$!query" format="json")
     constant(value="\",\"method\":\"")
        property(name="$!method" format="json")
     constant(value="\",\"status\":\"")
        property(name="$!status" format="json")
     constant(value="\",\"userAgent\":\"")
        property(name="$!userAgent" format="json")
     constant(value="\",\"referer\":\"")
        property(name="$!referer" format="json")
     constant(value="\",\"X-Forwarded-For\":\"")
        property(name="$!X-Forwarded-For" format="json")
   constant(value="\"}")
}
```

```
if $app-name == 'apache.access' and $fromhost-ip != '127.0.0.1'
then{
      #mmjsonparse se usa para obtener los datos en formato JSON que
trasnporta el mensaje
      action(type="mmjsonparse")

      #Copia local en un fichero
&     action( type="omfile"
              file="/var/log/remote/norm_apacheaccess.log"
              template="norm_apacheaccess"
              )

      #Acción para enviar logs a Kafka
&     action(type="omkafka" topic="rb_vault" broker="localhost:9092"
              template="norm_apacheaccess"
             )
    stop
}
```

/etc/rsyslog.d/05-parse_iptables.conf

```
  # Este fichero de configuración se encarga de obtener los logs
  # de iptables 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.

  template(name="norm_iptables" type="list"){
     constant(value="{")
     constant(value="\"raw-message\":\"")
        property(name="rawmsg" format="json")
     constant(value="\",\"message\":\"")
        property(name="msg" format="json")

     constant(value="\",\"pri\":\"")
        property(name="pri" format="json")
     constant(value="\",\"pri-text\":\"")
        property(name="pri-text" 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="\",\"protocol-version\":\"")
        property(name="protocol-version" format="json")
     constant(value="\",\"timestamp\":\"")
        property(name="timestamp" dateFormat="unixtimestamp")
```

```
    constant(value="\",\"hostname\":\"")
       property(name="hostname" format="json")
    constant(value="\",\"fromhost-ip\":\"")
       property(name="fromhost-ip" 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="\",\"input-module\":\"")
       property(name="inputname" format="json")
    constant(value="\",\"proxy-uuid\":\"")
       constant(value="fd6fc5e1-e30a-48f1-a109-d4af0ddcec01")
    constant(value="\",\"parsed-info\":")
       property(name="$!")
    constant(value="}")
}

if $app-name == 'kernel' and $msg contains 'iptables' and $fromhost-
  ip != '127.0.0.1' then
{
#Extracción campos mediante reglas, usando el modulo liblognorm
  action(type="mmnormalize"
  ruleBase="/etc/rsyslog.d/rules/iptables.rules")

#Copia local en un fichero.
& action(  type="omfile"
           file="/var/log/remote/norm_iptables.log"
           template="norm_iptables"
           )

#Acción para enviar los logs a Apache Kafka
&   action(type="omkafka" topic="rb_vault" broker="localhost:9092"
            template="norm_iptables"
                      )
  stop
}
```

/etc/rsyslog.d/06-parse_Cisco_WLC.conf

```
# Este fichero de configuración se encarga de obtener los logs
# de los wireless controller de Cisco 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.

template(name="norm_Cisco_WLC" type="list"){
   constant(value="{")
   constant(value="\"raw-message\":\"")
      property(name="rawmsg" format="json")
   constant(value="\",\"message\":\"") property(name="msg"
      format="json")
```

```
    constant(value="\",\"pri\":\"")
       property(name="pri" format="json")
    constant(value="\",\"pri-text\":\"")
       property(name="pri-text" 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="\",\"protocol-version\":\"")
       property(name="protocol-version" format="json")
    constant(value="\",\"timestamp\":\"")
       property(name="timestamp" dateFormat="unixtimestamp")
    constant(value="\",\"hostname\":\"")
       property(name="hostname" format="json")
    constant(value="\",\"fromhost-ip\":\"")
       property(name="fromhost-ip" 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="\",\"input-module\":\"")
       property(name="inputname" format="json")
    constant(value="\",\"proxy-uuid\":\"")
       constant(value="fd6fc5e1-e30a-48f1-a109-d4af0ddcec01")
    constant(value="\",\"parsed-info\":")
       property(name="$!")
              constant(value="}")
}

if $app-name == 'WLC1' and $fromhost-ip != '127.0.0.1' then
{
#Extracción campos mediante reglas, usando el modulo liblognorm
     action(type="mmnormalize"
        ruleBase="/etc/rsyslog.d/rules/Cisco_WLC.rules")

#Copia local en un fichero.
&    action(    type="omfile"
                file="/var/log/remote/norm_Cisco_WLC.log"
                template="norm_Cisco_WLC"
                )

#Acción para enviar los logs a Apache Kafka
&   action(type="omkafka" topic="rb_vault" broker="localhost:9092"
           template="norm_Cisco_WLC"
                     )
     stop
}
```

/etc/rsyslog.d/99-parse_rfc5424.conf

```
#Plantilla de salida para logs con formato rfc5424
template(name="norm_rfc5424" type="list"){
    constant(value="{")
    constant(value="\"raw-message\":\"")
       property(name="rawmsg" format="json")
    constant(value="\",\"message\":\"")
       property(name="msg" format="json")

    constant(value="\",\"pri\":\"")
       property(name="pri" format="json")
    constant(value="\",\"pri-text\":\"")
       property(name="pri-text" 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="\",\"protocol-version\":\"")
       property(name="protocol-version" format="json")
    constant(value="\",\"timestamp\":\"")
       property(name="timestamp" dateFormat="unixtimestamp")
    constant(value="\",\"hostname\":\"")
       property(name="hostname" format="json")
    constant(value="\",\"fromhost-ip\":\"")
       property(name="fromhost-ip" 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="\",\"structured-data\":\"")
       property(name="structured-data" format="json")
    constant(value="\",\"tag\":\"")
       property(name="$!rfc5424-sd!rbvault@39483!tag"
       format="json")
    constant(value="\",\"sensor-uuid\":\"")
       property(name="$!rfc5424-sd!rbvault@39483!sensor-uuid"
       format="json")
    constant(value="\",\"hash\":\"")
       property(name="$!rfc5424-sd!hash@39483!hash" format="json")

    constant(value="\",\"inputname\":\"")
       property(name="inputname" format="json")
    constant(value="\",\"proxy-uuid\":\"")
       constant(value="fd6fc5e1-e30a-48f1-a109-d4af0ddcec01")
    constant(value="\"}")
}
```

```
if $fromhost-ip != '127.0.0.1' then
{
 #Copia local en un fichero.
   action(    type="omfile"
              file="/var/log/remote/norm_rfc5424.log"
              template="norm_rfc5424"
              )
#Acción para enviar los logs a Apache Kafka
&    action(type="omkafka" topic="rb_vault" broker="localhost:9092"
            template="norm_rfc5424"
                       )
   Stop
}
```

Ficheros de definición de reglas de parseo

/etc/rsyslog.d/rules/ssh.rules

```
#Este fichero contiene las reglas de parseo para liblognorm,
#que se aplicarán a los logs de la aplicación ssh. Se contemplan
#los logs que aparecen con más frecuencia.

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%
rule=:message repeated %-:number% times: [ %auth:word% %auth-method:word%
  for %username:word% from %src-ip:ipv4% port %src-port:number%
  %protocol:char-to:\x5d%\x5d
rule=:pam_unix(sshd:session): session %session-status:word% for user
  %username:word% by (uid=%uid:number%)
rule=:pam_unix(sshd:session): session %session-status:word% for user
  %username:word%
rule=:pam_unix(sshd:auth): %auth-method:word% %auth:char-to:\x3b%\x3b
  logname= uid=%uid:number% euid=%euid:number% tty=%tty:word% ruser=
  rhost=%src-ip:ipv4%%-:whitespace%user=%username:word%
rule=:pam_unix(sshd:auth): %auth-method:word% %auth:char-to:\x3b%\x3b
  logname= uid=%uid:number% euid=%euid:number% tty=%tty:word% ruser=
  rhost=%src-ip:ipv4%
rule=:pam_unix(sshd:auth): check pass; user %username:word%
rule=:Invalid user %username:word% from %src-ip:ipv4%
rule=:input_userauth_request: invalid user %username:word% [preauth]
rule=:Received disconnect from %src-ip:ipv4%: %-:number%: %session-
  status:word% by user
rule=:Received signal %-:number%; %session-status:char-to:\x2e%\x2e
rule=:Received %signal:char-to:\x3b%\x3b %status:char-to:\x2e%\x2e
rule=:PAM %-:number% more %auth-method:word% %auth:char-to:\x3b%\x3b
  logname= uid=%uid:number% euid=%euid:number% tty=%tty:word% ruser=
  rhost=%src-ip:ipv4%%-:whitespace%user=%username:word%
rule=:Connection %session-status:word% by %src-ip:ipv4% [preauth]
rule=:Server %server-status:word% on %server-ip:ipv4% port %server-
  port:number%\x2e
rule=:Server %server-status:word% on %server-ip:ipv6% port %server-
  port:number%\x2e
```

/etc/rsyslog.d/rules/iptables.rules

```
prefix=[%kernel-timestamp:char-to:]%] iptables\x3a

#Ejemplo: [31628.772760] iptables:IN=eth3 OUT=
  MAC=ff:ff:ff:ff:ff:ff:0a:00:27:00:00:00:08:00 SRC=192.168.56.1
  DST=192.168.56.255 LEN=336 TOS=0x00 PREC=0x00 TTL=64 ID=10020 DF
  PROTO=UDP SPT=17500 DPT=17500 LEN=316
rule=:IN=%in-interface:word% OUT= MAC=%dst-mac:mac48%\x3a%src-
  mac:mac48%\x3a%ethertype:char-to:\x20% SRC=%src-ip:ipv4% DST=%dst-
  ip:ipv4% LEN=%frame-length:number% TOS=%type-of-service:hexnumber%
  PREC=%precision:hexnumber% TTL=%ttl:number% ID=%id:number% DF
  PROTO=%proto:word% SPT=%src-port:number% DPT=%dst-port:number%
  LEN=%packet-length:number%

#Ejemplo: [31628.772760] iptables:IN=eth3 OUT=
  MAC=ff:ff:ff:ff:ff:ff:0a:00:27:00:00:00:08:00 SRC=192.168.56.1
  DST=192.168.56.255 LEN=336 TOS=0x00 PREC=0x00 TTL=64 ID=10020 PROTO=UDP
  SPT=17500 DPT=17500 LEN=316
rule=:IN=%in-interface:word% OUT= MAC=%dst-mac:mac48%\x3a%src-
  mac:mac48%\x3a%ethertype:char-to:\x20% SRC=%src-ip:ipv4% DST=%dst-
  ip:ipv4% LEN=%frame-length:number% TOS=%type-of-service:hexnumber%
  PREC=%precision:hexnumber% TTL=%ttl:number% ID=%id:number%
  PROTO=%proto:word% SPT=%src-port:number% DPT=%dst-port:number%
  LEN=%packet-length:number%

#Ejemplo: [31599.573324] iptables:IN=eth2 OUT= MAC= SRC=172.16.7.6
  DST=224.0.0.251 LEN=73 TOS=0x00 PREC=0x00 TTL=255 ID=50425 DF PROTO=UDP
  SPT=5353 DPT=5353 LEN=53
rule=:IN=%in-interface:word% OUT= MAC= SRC=%src-ip:ipv4% DST=%dst-ip:ipv4%
  LEN=%frame-length:number% TOS=%type-of-service:hexnumber%
  PREC=%precision:hexnumber% TTL=%ttl:number% ID=%id:number% DF
  PROTO=%proto:word% SPT=%src-port:number% DPT=%dst-port:number%
  LEN=%packet-length:number%

#Ejemplo: [31599.573324] iptables:IN=eth2 OUT= MAC= SRC=172.16.7.6
  DST=224.0.0.251 LEN=73 TOS=0x00 PREC=0x00 TTL=255 ID=50425 PROTO=UDP
  SPT=5353 DPT=5353 LEN=53
rule=:IN=%in-interface:word% OUT= MAC= SRC=%src-ip:ipv4% DST=%dst-ip:ipv4%
  LEN=%frame-length:number% TOS=%type-of-service:hexnumber%
  PREC=%precision:hexnumber% TTL=%ttl:number% ID=%id:number%
  PROTO=%proto:word% SPT=%src-port:number% DPT=%dst-port:number%
  LEN=%packet-length:number%

#Ejemplo: [30819.806655] iptables:IN=eth0 OUT=
  MAC=08:00:27:a5:0d:91:52:54:00:12:35:02:08:00 SRC=10.0.150.72
  DST=10.0.2.15 LEN=40 TOS=0x00 PREC=0x00 TTL=64 ID=398 PROTO=TCP SPT=514
  DPT=47125 WINDOW=65535 RES=0x00 ACK URGP=0
rule=:IN=%in-interface:word% OUT= MAC=%dst-mac:mac48%\x3a%src-
  mac:mac48%\x3a%ethertype:char-to:\x20% SRC=%src-ip:ipv4% DST=%dst-
  ip:ipv4% LEN=%frame-length:number% TOS=%type-of-service:hexnumber%
  PREC=%precision:hexnumber% TTL=%ttl:number% ID=%id:number%
  PROTO=%proto:word% SPT=%src-port:number% DPT=%dst-port:number%
  WINDOW=%tcp-window-size:number% RES=%tcp-reserved:hexnumber% %tcp-pkt-
  type:word% URGP=%tcp-urgent-pointer:number%
```

```
#Ejemplo: [28886.863205] iptables:IN=eth3 OUT=
    MAC=08:00:27:ce:3d:67:0a:00:27:00:00:00:08:00 SRC=192.168.56.1
    DST=192.168.56.101 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=23349 DF
    PROTO=TCP SPT=50616 DPT=22 WINDOW=29200 RES=0x00 SYN URGP=0
rule=:IN=%in-interface:word% OUT= MAC=%dst-mac:mac48%\x3a%src-
    mac:mac48%\x3a%ethertype:char-to:\x20% SRC=%src-ip:ipv4% DST=%dst-
    ip:ipv4% LEN=%frame-length:number% TOS=%type-of-service:hexnumber%
    PREC=%precision:hexnumber% TTL=%ttl:number% ID=%id:number% DF
    PROTO=%proto:word% SPT=%src-port:number% DPT=%dst-port:number%
    WINDOW=%tcp-window-size:number% RES=%tcp-reserved:hexnumber% %tcp-pkt-
    type:word% URGP=%tcp-urgent-pointer:number%

#Ejemplo:  [30820.695881] iptables:IN=eth3 OUT=
    MAC=08:00:27:ce:3d:67:0a:00:27:00:00:00:08:00 SRC=192.168.56.1
    DST=192.168.56.101 LEN=108 TOS=0x10 PREC=0x00 TTL=64 ID=36954 DF
    PROTO=TCP SPT=50720 DPT=22 WINDOW=351 RES=0x00 ACK PSH URGP=0
rule=:IN=%in-interface:word% OUT= MAC=%dst-mac:mac48%\x3a%src-
    mac:mac48%\x3a%ethertype:char-to:\x20% SRC=%src-ip:ipv4% DST=%dst-
    ip:ipv4% LEN=%frame-length:number% TOS=%type-of-service:hexnumber%
    PREC=%precision:hexnumber% TTL=%ttl:number% ID=%id:number% DF
    PROTO=%proto:word% SPT=%src-port:number% DPT=%dst-port:number%
    WINDOW=%tcp-window-size:number% RES=%tcp-reserved:hexnumber% %tcp-pkt-
    type:word% %tcp-flag:word% URGP=%tcp-urgent-pointer:number%

#Ejemplo: [38199.519558] iptables:IN=eth0 OUT=
    MAC=08:00:27:a5:0d:91:52:54:00:12:35:02:08:00 SRC=10.0.2.2
    DST=10.0.2.15 LEN=60 TOS=0x00 PREC=0xC0 TTL=255 ID=10 PROTO=ICMP
    TYPE=11 CODE=0 [SRC=10.0.2.15 DST=224.0.0.22 LEN=40 TOS=0x00 PREC=0xC0
    TTL=1 ID=0 DF PROTO=2 ]
rule=:IN=%in-interface:word% OUT= MAC=%dst-mac:mac48%\x3a%src-
    mac:mac48%\x3a%ethertype:char-to:\x20% SRC=%src-ip:ipv4% DST=%dst-
    ip:ipv4% LEN=%frame-length:number% TOS=%type-of-service:hexnumber%
    PREC=%precision:hexnumber% TTL=%ttl:number% ID=%id:number%
    PROTO=%proto:word% TYPE=%icmp-type:number% CODE=%icmp-code:number%
    [SRC=%icmp-src-ip:ipv4% DST=%icmp-dst-ip:ipv4% LEN=%icmp-
    lenght:number% TOS=%icmp-type-of-service:hexnumber% PREC=%icmp-
    precision:hexnumber% TTL=%icmp-ttl:number% ID=%icmp-id:number% DF
    PROTO=%icmp-proto:number% ]

#Ejemplo: [ 8291.747312] iptables:IN=lo OUT=
    MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00 SRC=192.168.0.6
    DST=192.168.0.6 LEN=88 TOS=0x00 PREC=0xC0 TTL=64 ID=20970 PROTO=ICMP
    TYPE=3 CODE=1 [SRC=192.168.0.6 DST=10.0.150.72 LEN=60 TOS=0x00
    PREC=0x00 TTL=64 ID=41026 DF PROTO=TCP SPT=58436 DPT=514 WINDOW=29200
    RES=0x00 SYN URGP=0 ]
rule=:IN=%in-interface:word% OUT= MAC=%dst-mac:mac48%\x3a%src-
    mac:mac48%\x3a%ethertype:char-to:\x20% SRC=%src-ip:ipv4% DST=%dst-
    ip:ipv4% LEN=%frame-length:number% TOS=%type-of-service:hexnumber%
    PREC=%precision:hexnumber% TTL=%ttl:number% ID=%id:number%
    PROTO=%proto:word% TYPE=%icmp-type:number% CODE=%icmp-code:number%
    [SRC=%icmp-src-ip:ipv4% DST=%icmp-dst-ip:ipv4% LEN=%icmp-
    lenght:number% TOS=%icmp-type-of-service:hexnumber% PREC=%icmp-
    precision:hexnumber% TTL=%icmp-ttl:number% ID=%icmp-id:number% DF
    PROTO=%icmp-proto:word% SPT=%icmp-src-port:number% DPT=%icmp-dst-
    port:number% WINDOW=%tcp-window-size:number% RES=%tcp-
    reserved:hexnumber% %tcp-pkt-type:word% URGP=%tcp-urgent-
    pointer:number% ]
```

```
#Ejemplo: [ 8291.747312] iptables:IN=lo OUT=
  MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00 SRC=192.168.0.6
  DST=192.168.0.6 LEN=88 TOS=0x00 PREC=0xC0 TTL=64 ID=20970 PROTO=ICMP
  TYPE=3 CODE=1 [SRC=192.168.0.6 DST=10.0.150.72 LEN=60 TOS=0x00
  PREC=0x00 TTL=64 ID=41026 DF PROTO=TCP SPT=58436 DPT=514 WINDOW=29200
  RES=0x00 SYN URGP=0 ]
rule=:IN=%in-interface:word% OUT= MAC=%dst-mac:mac48%\x3a%src-
  mac:mac48%\x3a%ethertype:char-to:\x20% SRC=%src-ip:ipv4% DST=%dst-
  ip:ipv4% LEN=%frame-length:number% TOS=%type-of-service:hexnumber%
  PREC=%precision:hexnumber% TTL=%ttl:number% ID=%id:number%
  PROTO=%proto:word% TYPE=%icmp-type:number% CODE=%icmp-code:number%
  [SRC=%icmp-src-ip:ipv4% DST=%icmp-dst-ip:ipv4% LEN=%icmp-lenght:number%
  TOS=%icmp-type-of-service:hexnumber% PREC=%icmp-precision:hexnumber%
  TTL=%icmp-ttl:number% ID=%icmp-id:number% DF PROTO=%icmp-proto:word%
  SPT=%icmp-src-port:number% DPT=%icmp-dst-port:number% WINDOW=%tcp-
  window-size:number% RES=%tcp-reserved:hexnumber% %tcp-pkt-type:word%
  URGP=%tcp-urgent-pointer:number% ]

#Ejemplo: [ 9613.461813] iptables:IN=lo OUT=
  MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00 SRC=192.168.0.6
  DST=192.168.0.6 LEN=104 TOS=0x10 PREC=0xC0 TTL=64 ID=65113 PROTO=ICMP
  TYPE=3 CODE=1 [SRC=192.168.0.6 DST=193.145.15.15 LEN=76 TOS=0x10
  PREC=0x00 TTL=64 ID=63107 DF PROTO=UDP SPT=42753 DPT=123 LEN=56 ]
rule=:IN=%in-interface:word% OUT= MAC=%dst-mac:mac48%\x3a%src-
  mac:mac48%\x3a%ethertype:char-to:\x20% SRC=%src-ip:ipv4% DST=%dst-
  ip:ipv4% LEN=%frame-length:number% TOS=%type-of-service:hexnumber%
  PREC=%precision:hexnumber% TTL=%ttl:number% ID=%id:number%
  PROTO=%proto:word% TYPE=%icmp-type:number% CODE=%icmp-code:number%
  [SRC=%icmp-src-ip:ipv4% DST=%icmp-dst-ip:ipv4% LEN=%icmp-lenght:number%
  TOS=%icmp-type-of-service:hexnumber% PREC=%icmp-precision:hexnumber%
  TTL=%icmp-ttl:number% ID=%icmp-id:number% DF PROTO=%icmp-proto:word%
  SPT=%icmp-src-port:number% DPT=%icmp-dst-port:number% LEN=%udp-
  length:number% ]

#Repetimos las mismas reglas para el caso de que los logs se generen en la
  cadena OUTPUT.
#...
```

/etc/rsyslog.d/rules/Cisco_WLC.rules

```
prefix= *%process:char-to:\x3a%\x3a %-:date-rfc3164%.%-:number%\x3a
  %MNEMONIC:char-to:\x3a%\x3a %file:char-to:\x3a%\x3a%line:number%

#Ejemplo: *sisfSwitcherTask: May 03 10:48:16.083: #SISF-6-ENTRY_CREATED:
  sisf_shim_utils.c:486 Entry created A=fe80::820:d412:94de:840 V=30
  I=wireless:0 P=0005 M=90:60:f1:d4:c4:cb
rule=:Entry %action:word% A=%station_ip:ipv6% V=%V:number% I=%I:word%
  P=%P:word% M=%station_mac:mac48%

#Ejemplo: *osapiBsnTimer: May 03 10:32:40.462: #SISF-6-ENTRY_DELETED:
  sisf_shim_utils.c:483 Entry deleted A=fe80::ae72:89ff:fecf:3cee V=50
  I=wireless:0 P=0005 M=ac:72:89:cf:3c:ee
rule=:Entry %action:word% A=%station_ip:ipv6% V=%V:number% I=%I:word%
  P=%P:word% M=

#Ejemplo: *SISF BT Process: May 03 10:21:31.829: #LOG-6-Q_IND:
  sisf_shim_utils.c:489 Entry changed A=fe80::7952:f9fc:38b2:5693 V=30
  I=wireless:0 P=0000 M=[...It occurred %-:number% times.!]
rule=:Entry %action:word% A=%station_ip:ipv6% V=%V:number% I=%I:word%
  P=%P:word% M=%-:rest%
```

```
#Reglas anteriores repetidas para ipv4
rule=:Entry %action:word% A=%station_ip:ipv4% V=%V:number% I=%I:word%
  P=%P:word% M=%station_mac:mac48%
rule=:Entry %action:word% A=%station_ip:ipv4% V=%V:number% I=%I:word%
  P=%P:word% M=
rule=:Entry %action:word% A=%station_ip:ipv4% V=%V:number% I=%I:word%
  P=%P:word% M=%-:rest%
```

ANEXO E: ZOOKEEPER – INSTALACIÓN Y CONFIGURACIÓN

En este Anexo mostraremos el procedimiento para instalar Zookeeper [18] en Centos y la configuración que se ha usado para el desarrollo del proyecto. Además, se mostrará los cambios necesarios para hacer que Zookeeper se comporte como un servicio del sistema.

Como paso previo a la instalación de Zookeeper, será necesario tener instalado en nuestro sistema la máquina virtual de Java (JRE).

Instalación de Zookeeper

El primer paso será descargar la versión actual de Zookeeper, que se puede descargar desde el siguiente enlace: http://apache.rediris.es/zookeeper/ . En el momento de escribir este anexo, la versión actual de Zookeeper es la 3.4.8.

Una vez descargado, será necesario descomprimirlo en la carpeta "/opt" (usar esta carpeta es opcional). Para ello se hará uso de la herramienta "tar". En la figura E-1 se muestra este procedimiento, usando para la descarga de Zookeeper la herramienta "curl".

```
[root@localhost ~]# cd /opt/
[root@localhost opt]# curl -O http://apache.rediris.es/zookeeper/zookeeper-3.4.8
/zookeeper-3.4.8.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 21.2M  100 21.2M    0     0  4192k      0  0:00:05  0:00:05 --:--:-- 4394k
[root@localhost opt]# tar -zxf zookeeper-3.4.8.tar.gz
[root@localhost opt]# ls -l
total 21744
drwxr-xr-x. 10 arodriguezc arodriguezc     4096 feb  6 04:50 zookeeper-3.4.8
-rw-r--r--.  1 root        root        22261552 jun  5 13:34 zookeeper-3.4.8.tar
.gz
[root@localhost opt]# _
```

Figura E-1. Descarga de Zookeeper.

Como se puede observar en la captura anterior, al descomprimir se nos crea un directorio con el mismo nombre que el archivo comprimido. Dentro de este encontramos varios directorios, entre los que cabe destacar los siguientes:

- El directorio "bin" contiene varios scripts entre los que destaca "zkServer.sh", que nos servirá para arrancar y parar el proceso servidor de Zookeeper.
- El directorio "conf" contiene las configuraciones útiles para Zookeeper, como por ejemplo la configuración de la librería log4j (para crear logs desde una aplicación escrita en java) y la configuración del propio servidor de Zookeeper. Cuando iniciemos el servidor Zookeeper, se buscará por defecto en esta carpeta el archivo "zoo.cfg" que debe contener la configuración de Zookeeper.

Tras crear el archivo de configuración "conf/zoo.cgf", ya podemos iniciar el servidor Zookeeper haciendo uso del script "bin/zkServer" de la siguiente forma:

```
./bin/zkServer start
```

Si todo ha salido correctamente, debería iniciarse el servidor Zookeeper y mostrar la salida de la figura E-2:

Figura E-2. Iniciar el servidor Zookeeper.

Configuración de Zookeeper

A continuación, mostramos la configuración de Zookeeper usada para el desarrollo de este proyecto.

/opt/zookeeper/conf/zoo.cfg

```
# The number of milliseconds of each tick
tickTime=2000

# The number of ticks that the initial
# synchronization phase can take
initLimit=5

# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=2

forceSync=no

# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/tmp/zookeeper

# the port at which the clients will connect
clientPort=2181

# the IP address at which the clients will connect
clientPortAddress=127.0.0.1

# the maximum number of client connections.
# increase this if you need to handle more clients
maxClientCnxns=0

# The number of snapshots to retain in dataDir
autopurge.snapRetainCount=5

# Purge task interval in hours
# Set to "0" to disable auto purge feature
autopurge.purgeInterval=24
```

Zookeeper como servicio

Para configurar Zookeeper como servicio [20] vamos a usar el script "bin/zkServer.sh" ligeramente modificado. Las modificaciones están localizadas al principio de este archivo por lo que se muestra a continuación sólo la parte que ha sido modificada.

/opt/zookeeper/conf/zoo.cfg

```
#!/usr/bin/env bash
# description: Zookeeper Start Stop Restart
# processname: zookeeper
# chkconfig: 234 30 80

# Licensed to the Apache Software Foundation (ASF) under one or more
...

# use POSTIX interface, symlink is followed automatically
ZOOSH=`readlink $0`
ZOOBIN=`dirname $ZOOSH`
ZOOBINDIR=`cd $ZOOBIN; pwd`
ZOO_LOG_DIR=`echo $ZOOBIN`

if [ -e "$ZOOBIN/../libexec/zkEnv.sh" ]; then
  . "$ZOOBINDIR/../libexec/zkEnv.sh"
else
  . "$ZOOBINDIR/zkEnv.sh"
fi
```

Una vez realizadas las modificaciones indicadas, el siguiente paso es crear un enlace simbólico de este script en la carpeta "/etc/init.d/" y tras esto, configurar nuestro servicio para que se ejecute automáticamente al iniciar nuestro equipo. No podemos olvidar dar al directorio de Zookeeper los permisos necesarios. Para ello debemos ejecutar los siguientes comandos:

```
chmod -R 750 zookeeper-3.4.5
ln -s /opt/zookeeper-3.4.5/bin/zkServer.sh /etc/init.d/zookeeper
chkconfig zookeeper on
```

Tras realizar estas acciones ya podremos hacer uso de Zookeeper como si de cualquier otro servicio se tratase. Al reiniciar nuestro equipo, Zookeeper se ejecutará de forma automática, como se puede comprobar en la figura E-3.

```
[root@localhost ~]# service zookeeper status
ZooKeeper JMX enabled by default
Using config: /opt/zookeeper-3.4.8/bin/../conf/zoo.cfg
Mode: standalone
[root@localhost ~]# _
```

Figura E-3. Estado del servicio Zookeeper.

ANEXO F: KAFKA – INSTALACIÓN Y CONFIGURACIÓN

En este Anexo mostraremos el procedimiento para instalar Kafka [17] en Centos y la configuración que se ha usado para el desarrollo del proyecto. Además, se mostrará los cambios necesarios para hacer que Kafka se comporte como un servicio del sistema.

Como paso previo a la instalación de Kafka, será necesario tener instalado en nuestro sistema la máquina virtual de Java (JRE), además de Zookeeper (ver anexo C).

Instalación de Kafka

El primer paso será descargar la versión actual de Kafka, que se puede descargar desde el siguiente enlace: http://apache.rediris.es/kafka/ . En el momento de escribir este anexo, la versión actual de Kafka es la 0.10.0.0.

Una vez descargado, será necesario descomprimirlo en la carpeta "/opt" (usar esta carpeta es opcional). Para ello se hará uso de la herramienta "tar". En la figura F-1 se muestra este procedimiento, usando para la descarga de Kafka la herramienta "curl".

```
[root@localhost ~]# cd /opt
[root@localhost opt]# curl -O http://apache.rediris.es/kafka/0.10.0.0/kafka_2.11
-0.10.0.0.tgz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 30.1M  100 30.1M    0     0  4021k      0  0:00:07  0:00:07 --:--:-- 4552k
[root@localhost opt]# tar -zxf kafka_2.11-0.10.0.0.tgz
[root@localhost opt]# ls -l
total 30844
drwxr-xr-x.  6 root        root             83 may 18 06:32 kafka_2.11-0.10.0.0
-rw-r--r--.  1 root        root       31579147 jun  5 18:03 kafka_2.11-0.10.0.0
.tgz
drwxr-x---. 10 arodriguezc arodriguezc     4096 jun  5 14:24 zookeeper-3.4.8
[root@localhost opt]# _
```

Figura F-1. Descarga de Kafka.

Como se puede observar en la captura anterior, al descomprimir se nos crea un directorio con el mismo nombre que el archivo comprimido. Dentro de este encontramos varios directorios, entre los que cabe destacar los siguientes:

- El directorio "bin" contiene varios scripts entre los que podemos destacar los siguientes:
    - *kafka-server-start.sh:* nos permite ejecutar el servidor Kafka. Debemos pasarle como parámetro el archivo que contiene la configuración del servidor Kafka.
    - *kafka-server-stop.sh:* nos permite parar la ejecución del servidor Kafka.
    - *kafka-topics.sh:* nos permite gestionar los topics del servidor Kafka (crear, eliminar, cambiar propiedades, listar, etc.).
    - *kafka-consolo-producer.sh:* productor en línea de comandos que envía al topic de Kafka especificado lo que recibe por entrada estándar.
    - *kafka-console-consumer.sh:* consumidor en línea de comandos que consume los mensajes del topic de Kafka especificado y lo muestra por la salida estándar.
    - *kafka-consumer-groups.sh:* gestión de grupos de consumidores.

- El directorio "config" contiene las configuraciones útiles para Kafka, como por ejemplo la configuración de la librería log4j (para crear logs desde una aplicación escrita en java), la configuración del productor y consumidor por línea de comandos comentado anteriormente, y la configuración del propio servidor de Kafka (fichero "server.properties").

Tras crear el archivo de configuración "config/server.properties" y haber iniciado el servidor Zookeeper (ver anexo C), ya podemos iniciar el servidor Kafka haciendo uso del script "bin/kafka-server-start.sh" de la siguiente forma:

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

Si todo ha salido correctamente, debería iniciarse el servidor Kafka y mostrar el mensaje de la figura F-2:

Figura F-2. Iniciar el servidor Kafka.

Configuración de Kafka

A continuación, mostramos la configuración de Kafka usada para el desarrollo de este proyecto:

/opt/kafka/config/server.properties

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# see kafka.server.KafkaConfig for additional details and defaults

############################# Server Basics #############################

# The id of the broker. This must be set to a unique integer for each
broker.
broker.id=0

listen=127.0.0.1
advertised.host.name=127.0.0.1
port=9092

default.replication.factor=1
auto.leader.rebalance.enable=true
controlled.shutdown.enable=true
auto.create.topics.enable=true

#############################Socket Server Settings
#############################
# The address the socket server listens on. It will get the value returned
from java.net.InetAddress.getCanonicalHostName() if not configured.
```

```
#   FORMAT:
#     listeners = security_protocol://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
#listeners=PLAINTEXT://:9092

# Hostname and port the broker will advertise to producers and consumers.
If not set,
# it uses the value for "listeners" if configured.  Otherwise, it will use
the value
# returned from java.net.InetAddress.getCanonicalHostName().
#advertised.listeners=PLAINTEXT://your.host.name:9092

# The number of threads handling network requests
num.network.threads=3

# The number of threads doing disk I/O
num.io.threads=8

# The send buffer (SO_SNDBUF) used by the socket server
socket.send.buffer.bytes=102400

# The receive buffer (SO_RCVBUF) used by the socket server
socket.receive.buffer.bytes=102400

# The maximum size of a request that the socket server will accept
(protection against OOM)
socket.request.max.bytes=104857600

max.socket.request.bytes=104857600


############################# Log Basics #############################

# A comma seperated list of directories under which to store log files
log.dirs=/tmp/kafka

# The default number of log partitions per topic. More partitions allow
greater
# parallelism for consumption, but this will also result in more files
across
# the brokers.
num.partitions=2

# The number of threads per data directory to be used for log recovery at
startup and flushing at shutdown.
# This value is recommended to be increased for installations with data
dirs located in RAID array.
num.recovery.threads.per.data.dir=1

############################# Log Flush Policy
#############################

# Messages are immediately written to the filesystem but by default we only
fsync() to sync
# the OS cache lazily. The following configurations control the flush of
data to disk.
# There are a few important trade-offs here:
#    1. Durability: Unflushed data may be lost if you are not using
replication.
#    2. Latency: Very large flush intervals may lead to latency spikes when
the flush does occur as there will be a lot of data to flush.
```

```
#    3. Throughput: The flush is generally the most expensive operation,
and a small flush interval may lead to exceessive seeks.
# The settings below allow one to configure the flush policy to flush data
after a period of time or
# every N messages (or both). This can be done globally and overridden on a
per-topic basis.

# The number of messages to accept before forcing a flush of data to disk
#log.flush.interval.messages=10000

# The maximum amount of time a message can sit in a log before we force a
flush
log.flush.interval.ms=1000

############################# Log Retention Policy
#############################

# The following configurations control the disposal of log segments. The
policy can
# be set to delete segments after a period of time, or after a given size
has accumulated.
# A segment will be deleted whenever *either* of these criteria are met.
Deletion always happens
# from the end of the log.

# The minimum age of a log file to be eligible for deletion
log.retention.minutes=90

# A size-based retention policy for logs. Segments are pruned from the log
as long as the remaining
# segments don't drop below log.retention.bytes.
log.retention.bytes=615037383

# The maximum size of a log segment file. When this size is reached a new
log segment will be created.
log.segment.bytes=553533644

log.segment.delete.delay.ms=10000

# The interval at which log segments are checked to see if they can be
deleted according
# to the retention policies
log.retention.check.interval.ms=30000

log.cleanup.interval.mins=1

log.cleaner.enable=true

############################# Zookeeper #############################
# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=localhost:2181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=12000

zookeeper.session.timeout.ms=12000
```

Comandos útiles

A continuación, mostraremos una lista de comandos útiles a la hora de usar Kafka. Estos comandos se basan en los scripts comentados anteriormente y abarcan las funcionalidades más básicas de Kafka. Estos comandos deben invocarse desde la carpeta "/opt/Kafka/", una vez ejecutados tanto el servidor Zookeeper como Kafka.

- El primer comando nos permite crear un topic, aunque se pueden configurar los brokers para generar los topics automáticamente de hecho, en la configuración usada para este proyecto se hace.

```
  ./bin/kafka-topics.sh --create --zookeeper localhost:2181
--replication-factor 1 --partitions 2 --topic test
```

  Como se puede apreciar, se hace uso del script "kafka-topics.sh" indicándole la opción "create" y siendo necesario especificar donde se encuentra el servidor Zookeeper (localhost:2181). La opción "replication-factor" nos permite especificar el número de copias que se crearán para este topic. La opción "partitions" se usa para asignar un número de particiones al topic. Por último, la opción "topic" se usa para darle nombre al topic.

- El siguiente comando nos permite listar los topics creados haciendo uso del script "kafka-topics.sh" con la opción "list":

```
  ./bin/kafka-topics.sh --list --zookeeper localhost:2181
```

- Por último, la siguiente pareja de comandos nos permite lanzar el productor y consumidor por línea de comandos respectivamente. Para el caso del productor, debemos indicarle el broker o brokers al que queremos enviar los mensajes mediante la opción "broker-list". En el caso del consumidor, se le debe pasar el host y puerto donde se ejecuta el servidor Zookeeper. En ambos casos debemos indicar mediante la opción "topic" el topic al donde queremos publicar o consumir.

```
  ./bin/kafka-console-producer.sh --broker-list localhost:9092
--topic test
  ./bin/kafka-console-consumer.sh --zookeeper localhost:2181
--topic test
```

ANEXO G: CHEF – COOKBOOK DEL CLIENT PROXY

En este anexo mostraremos los archivos completos que forman parte del cookbook de Chef para el Client Proxy de redBorder y cuyos aspectos más interesantes fueron explicados en el capítulo 7. Los archivos de este cookbook que sólo hayan sido modificados y no creados desde cero, sólo será incluida la parte donde se ha introducido la modificación, ya que el resto no es importante para este proyecto.

Se omiten las plantillas que generarán los archivos de definición de reglas de parseo, ya que estas plantillas no presentan ningún cambio respecto de los archivos mostrados en el anexo D.

/redBorder-proxy/attributes/default.rb

```
default["redBorder"]["proxy"]["rsyslog"]["enable_tcp"] = true
default["redBorder"]["proxy"]["rsyslog"]["enable_tls"] = false
default["redBorder"]["proxy"]["rsyslog"]["enable_udp"] = true
default["redBorder"]["proxy"]["rsyslog"]["enable_client_proxy_logs"] =
  true
default["redBorder"]["proxy"]["rsyslog"]["udp_port"] = 10514
default["redBorder"]["proxy"]["rsyslog"]["tcp_port"] = 10514

default["redBorder"]["proxy"]["rsyslog"]["enable_hash"] = true
default["redBorder"]["proxy"]["rsyslog"]["hash_key"] =
  "yourenterprisekey"
default["redBorder"]["proxy"]["rsyslog"]["hash_function"] = "sha256"

default["redBorder"]["proxy"]["rsyslog"]["config_dir"] =
  "/etc/rsyslog.d"
default["redBorder"]["proxy"]["rsyslog"]["certificates_dir"] =
  "#{node["redBorder"]["proxy"]["rsyslog"]["config_dir"]}/certificates"
default["redBorder"]["proxy"]["rsyslog"]["rules_dir"] =
  "#{node["redBorder"]["proxy"]["rsyslog"]["config_dir"]}/rules"
default["redBorder"]["proxy"]["rsyslog"]["remote_logs_dir"] =
  "/var/log/remote"
default["redBorder"]["proxy"]["rsyslog"]["work_dir"] =
  "/var/spool/rsyslog"

default["redBorder"]["proxy"]["rsyslog"]["permitted_peers_array"] =
  ['*.example.com','*.redborder.com']

default["redBorder"]["proxy"]["rsyslog"]["user"] = "syslog"
default["redBorder"]["proxy"]["rsyslog"]["group"] = "syslog"
```

/redBorder-proxy/resources/rsyslog.rb

```
# Cookbook Name:: redBorder-proxy
# Resource:: rsyslog
#

actions :config
default_action :config

attribute :enable_tls, :kind_of => [TrueClass, FalseClass], :default =>
  true

attribute :config_dir, :kind_of => String, :default => "/etc/rsyslog.d"
attribute :certificates_dir, :kind_of => String, :default =>
  "/etc/rsyslog.d/certificates"
attribute :rules_dir, :kind_of => String, :default =>
  "/etc/rsyslog.d/rules"
attribute :remote_logs_dir, :kind_of => String, :default =>
  "/var/log/remote"
attribute :work_dir, :kind_of => String, :default =>
  "/var/spool/rsyslog"

attribute :user, :kind_of => String, :default => "syslog"
attribute :group, :kind_of => String, :default => "syslog"
```

/redBorder-proxy/providers/rsyslog.rb

```
# Cookbook Name:: redBorder-proxy
# Provider:: rsyslog
#

action :config do
  begin
    enable_tls = new_resource.enable_tls
    config_dir = new_resource.config_dir
    certificates_dir = new_resource.certificates_dir
    rules_dir = new_resource.rules_dir
    remote_logs_dir = new_resource.remote_logs_dir
    work_dir = new_resource.work_dir
    user = new_resource.user
    group = new_resource.group


    group group do
      action  :create
    end

    user user do
      comment 'rsyslog user'
      shell   '/sbin/nologin'
      group group
      action  :create
    end

    directory config_dir do
      owner   'root'
      group   'root'
      mode    '0755'
      action  :create
    end
```

```
      if enable_tls
        directory certificates_dir do
          owner   'root'
          group   'root'
          mode    '0755'
          action  :create
        end
      end

      directory rules_dir do
        owner   'root'
        group   'root'
        mode    '0755'
        action  :create
      end

      directory work_dir do
        owner user
        group group
        mode  '0755'
        action  :create
      end

      directory remote_logs_dir do
        owner user
        group group
        mode  '0755'
        action  :create
      end

      template '/etc/rsyslog.conf' do
        source  'rsyslog.conf.erb'
        owner 'root'
        group   'root'
        mode  '0644'
        retries 2
        notifies :restart, 'service[rsyslog]', :delayed
      end

      template "#{config_dir}/01-server.conf" do
        source  'rsyslog_01-server.conf.erb'
        owner 'root'
        group   'root'
        mode  '0644'
        retries 2
        notifies :restart, 'service[rsyslog]', :delayed
      end

      template "#{config_dir}/02-general.conf" do
        source  'rsyslog_02-general.conf.erb'
        owner 'root'
        group   'root'
        mode  '0644'
        retries 2
        notifies :restart, 'service[rsyslog]', :delayed
      end
```

```
template "#{config_dir}/03-parse_ssh.conf" do
  source  'rsyslog_03-parse_ssh.conf.erb'
  owner 'root'
  group   'root'
  mode  '0644'
  retries 2
  notifies :restart, 'service[rsyslog]', :delayed
end

template "#{rules_dir}/ssh.rules" do
  source  'rsyslog_ssh.rules.erb'
  owner 'root'
  group   'root'
  mode  '0644'
  retries 2
  notifies :restart, 'service[rsyslog]', :delayed
end

template "#{config_dir}/04-parse_apacheaccess.conf" do
  source  'rsyslog_04-parse_apacheaccess.conf.erb'
  owner 'root'
  group   'root'
  mode  '0644'
  retries 2
  notifies :restart, 'service[rsyslog]', :delayed
end

template "#{config_dir}/05-parse_iptables.conf" do
  source  'rsyslog_05-parse_iptables.conf.erb'
  owner 'root'
  group   'root'
  mode  '0644'
  retries 2
  notifies :restart, 'service[rsyslog]', :delayed
end

template "#{rules_dir}/iptables.rules" do
  source  'rsyslog_iptables.rules.erb'
  owner 'root'
  group   'root'
  mode  '0644'
  retries 2
  notifies :restart, 'service[rsyslog]', :delayed
end

template "#{config_dir}/06-parse_Cisco_WLC.conf" do
  source  'rsyslog_06-parse_Cisco_WLC.conf.erb'
  owner 'root'
  group   'root'
  mode  '0644'
  retries 2
  notifies :restart, 'service[rsyslog]', :delayed
end
template "#{rules_dir}/Cisco_WLC.rules" do
  source  'rsyslog_Cisco_WLC.rules.erb'
  owner 'root'
  group   'root'
  mode  '0644'
  retries 2
  notifies :restart, 'service[rsyslog]', :delayed
 end
```

```
      template "#{config_dir}/99-parse_rfc5424.conf" do
        source  'rsyslog_99-parse_rfc5424.conf.erb'
        owner 'root'
        group   'root'
        mode  '0644'
        retries 2
        notifies :restart, 'service[rsyslog]', :delayed
      end

      service "rsyslog" do
        provider Chef::Provider::Service::Init::Redhat
        supports :status => true, :start => true, :restart => true
        ignore_failure true
        action([:start, :enable])
      end

      Chef::Log.info("rsyslog has been configured correctly.")

    rescue Exception => e
      Chef::Log.error(e.message)
    end
  end
```

/redBorder-proxy/recipes/default.rb

```
  redborder_proxy_rsyslog "config" do
    enable_tls node["redBorder"]["proxy"]["rsyslog"]["enable_tls"]
    config_dir node["redBorder"]["proxy"]["rsyslog"]["config_dir"]
    certificates_dir
       node["redBorder"]["proxy"]["rsyslog"]["certificates_dir"]
    rules_dir node["redBorder"]["proxy"]["rsyslog"]["rules_dir"]
    remote_logs_dir
       node["redBorder"]["proxy"]["rsyslog"]["remote_logs_dir"]
    work_dir node["redBorder"]["proxy"]["rsyslog"]["work_dir"]
    user node["redBorder"]["proxy"]["rsyslog"]["user"]
    group node["redBorder"]["proxy"]["rsyslog"]["group"]
    action :config
  end
```

/redBorder-proxy/templates/default/rsyslog.conf.erb

```
  #################
  #### MÓDULOS ####
  #################

  $ModLoad imuxsock # provides support for local system logging

 ###############################
 #### DIRECTIVAS GLOBALES ####
 ###############################

 #
 # Usar el formato de timestamp tradicional por defecto
 $ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat


  # Filtrar mensajes duplicados
  $RepeatedMsgReduction on
```

```
# Permisos por defectos para los archivos donde se guardarán los
# logs.
$FileOwner <%= node["redBorder"]["proxy"]["rsyslog"]["user"] %>
$FileGroup <%= node["redBorder"]["proxy"]["rsyslog"]["group"] %>
$FileCreateMode 0640
$DirCreateMode 0755
$Umask 0022
$PrivDropToUser <%= node["redBorder"]["proxy"]["rsyslog"]["user"] %>
$PrivDropToGroup <%= node["redBorder"]["proxy"]["rsyslog"]["group"] %>

# Directorio de trabajo donde se almacenarán las colas que
# continen los logs que van a ser procesados.
$WorkDirectory <%= node["redBorder"]["proxy"]["rsyslog"]["work_dir"] %>

# Incluir todos los archivos de configuración que se encuentren en
# el directorio /etc/rsyslog.d/
$IncludeConfig <%= node["redBorder"]["proxy"]["rsyslog"]["config_dir"]
  %>/*.conf
```

/redBorder-proxy/templates/default/rsyslog_01-server.conf.erb

```
<% if node["redBorder"]["proxy"]["rsyslog"]["enable_tcp"] %>
#Configuración TCP
$ModLoad imtcp

<% if node["redBorder"]["proxy"]["rsyslog"]["enable_tls"] %>
$DefaultNetstreamDriver gtls

$DefaultNetstreamDriverCAFile <%=
  node["redBorder"]["proxy"]["rsyslog"]["certificates_dir"] %>/ca.pem
$DefaultNetstreamDriverCertFile <%=
  node["redBorder"]["proxy"]["rsyslog"]["certificates_dir"] %>/cert.pem
$DefaultNetstreamDriverKeyFile <%=
  node["redBorder"]["proxy"]["rsyslog"]["certificates_dir"] %>/key.pem

$InputTCPServerStreamDriverMode 1
$InputTCPServerStreamDriverAuthMode x509/name
<% node["redBorder"]["proxy"]["rsyslog"]["permitted_peers_array"].each
  do |peer| %>
$InputTCPServerStreamDriverPermittedPeer <%= peer %>
<% end %>
<% end %>

$InputTCPServerRun <%= node["redBorder"]["proxy"]["rsyslog"]["tcp_port"]
  %>
<% end %>

<% if node["redBorder"]["proxy"]["rsyslog"]["enable_udp"] %>
#Configuración UDP
module(load="imudp")

input(type="imudp" port="<%=
  node["redBorder"]["proxy"]["rsyslog"]["udp_port"] %>")
<% end %>
```

/redBorder-proxy/templates/default/rsyslog_02-general.conf.erb

```
#Cargar los módulos necesarios.
module(load="omkafka")
module(load="mmjsonparse")
module(load="mmpstrucdata")
module(load="mmnormalize")
<% if node["redBorder"]["proxy"]["rsyslog"]["enable_hash"] %>
module(load="mmrfc5424addhmac")
<% end %>

#Acciones comunes a todos los mensajes.
<% if node["redBorder"]["proxy"]["rsyslog"]["enable_hash"] %>
action(type="mmrfc5424addhmac" key="<%=
  node["redBorder"]["proxy"]["rsyslog"]["hash_key"] %>"
  hashFunction="<%=
  node["redBorder"]["proxy"]["rsyslog"]["hash_function"] %>"
  sd_id="hash@39483")
<% end %>
action(type="mmpstrucdata")
```

/redBorder-proxy/templates/default/rsyslog_03-parse_ssh.conf.erb

```
####Plantilla para los logs del servicio ssh####
template(name="norm_ssh" type="list"){
      constant(value="{")
      constant(value="\"raw-message\":\"")
          property(name="rawmsg" format="json")
      constant(value="\",\"message\":\"")
          property(name="msg" format="json")

      constant(value="\",\"pri\":\"")
          property(name="pri" format="json")
      constant(value="\",\"pri-text\":\"")
          property(name="pri-text" 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="\",\"protocol-version\":\"")
          property(name="protocol-version" format="json")
      constant(value="\",\"timestamp\":\"")
          property(name="timestamp" dateFormat="unixtimestamp")
      constant(value="\",\"hostname\":\"")
          property(name="hostname" format="json")
      constant(value="\",\"fromhost-ip\":\"")
          property(name="fromhost-ip" 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="\",\"input-module\":\"")
          property(name="inputname" format="json")
      constant(value="\",\"proxy-uuid\":\"")
          constant(value="<%= `cat /opt/rb/etc/rb-uuid | tr -d "\n\r"`
          %>")
      constant(value="\",\"parsed-info\":")
          property(name="$!")
```

```
        constant(value="}")

}


if $app-name == 'sshd' <% if
   node["redBorder"]["proxy"]["rsyslog"]["enable_client_proxy_logs"] %>
   and $fromhost-ip != '127.0.0.1' <% end %> then
{
#Extracción campos mediante reglas, usando el modulo liblognorm
   action(type="mmnormalize" ruleBase="<%=
   node["redBorder"]["proxy"]["rsyslog"]["rules_dir"] %>/ssh.rules")

#Copia local en un fichero.
&  action(type="omfile"
          file="<%=
   node["redBorder"]["proxy"]["rsyslog"]["remote_logs_dir"]
   %>/norm_ssh.log"
          template="norm_ssh"
          )

#Acción para enviar los logs a Apache Kafka
&    action(type="omkafka" topic="rb_vault" broker="localhost:9092"
             template="norm_ssh"
                       )
   stop
}
```

/redBorder-proxy/templates/default/rsyslog_04-parse_apacheaccess.conf.erb

```
template(name="norm_apacheaccess" type="list"){
    constant(value="{")
     constant(value="\"raw-message\":\"")
        property(name="rawmsg" format="json")
     constant(value="\",\"message\":\"")
        property(name="msg" format="json")

     constant(value="\",\"pri\":\"")
        property(name="pri" format="json")
     constant(value="\",\"pri-text\":\"")
        property(name="pri-text" 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="\",\"protocol-version\":\"")
        property(name="protocol-version" format="json")
     constant(value="\",\"timestamp\":\"")
        property(name="timestamp" dateFormat="unixtimestamp")
     constant(value="\",\"hostname\":\"")
        property(name="hostname" format="json")
     constant(value="\",\"fromhost-ip\":\"")
        property(name="fromhost-ip" 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="\",\"structured-data\":\"")
          property(name="structured-data" format="json")
      constant(value="\",\"tag\":\"")
          property(name="$!rfc5424-sd!rbvault@39483!tag" format="json")
      constant(value="\",\"sensor-uuid\":\"")
          property(name="$!rfc5424-sd!rbvault@39483!sensor-uuid"
          format="json")
      constant(value="\",\"hash\":\"")
          property(name="$!rfc5424-sd!hash@39483!hash" format="json")

      constant(value="\",\"inputname\":\"")
          property(name="inputname" format="json")

      constant(value="\",\"proxy-uuid\":\"")
          constant(value="<%= `cat /opt/rb/etc/rb-uuid | tr -d "\n\r"` %>")

      constant(value="\",\"access-time\":\"")
          property(name="$!time" format="json")
      constant(value="\",\"remoteIP\":\"")
          property(name="$!remoteIP" format="json")
      constant(value="\",\"host\":\"")
          property(name="$!host" format="json")
      constant(value="\",\"request\":\"")
          property(name="$!request" format="json")
      constant(value="\",\"query\":\"")
          property(name="$!query" format="json")
      constant(value="\",\"method\":\"")
          property(name="$!method" format="json")
      constant(value="\",\"status\":\"")
          property(name="$!status" format="json")
      constant(value="\",\"userAgent\":\"")
          property(name="$!userAgent" format="json")
      constant(value="\",\"referer\":\"")
          property(name="$!referer" format="json")
      constant(value="\",\"X-Forwarded-For\":\"")
          property(name="$!X-Forwarded-For" format="json")
    constant(value="\"}")
 }

  if $app-name == 'apache.access' <% if
    node["redBorder"]["proxy"]["rsyslog"]["enable_client_proxy_logs"] %>
    and $fromhost-ip != '127.0.0.1' <% end %> then{
     #mmjsonparse se usa para obtener los datos en formato JSON que
trasnporta el mensaje
    action(type="mmjsonparse")

     #Copia local en un fichero
  &  action( type="omfile" file="<%=
    node["redBorder"]["proxy"]["rsyslog"]["remote_logs_dir"]
    %>/norm_apacheaccess.log"
          template="norm_apacheaccess" )

     #Acción para enviar logs a Kafka
  &  action(type="omkafka" topic="rb_vault" broker="localhost:9092"
              template="norm_apacheaccess" )
     stop
  }
```

/redBorder-proxy/templates/default/rsyslog_05-parse_iptables.conf.erb

```
template(name="norm_iptables" type="list"){
    constant(value="{")
    constant(value="\"raw-message\":\"")
        property(name="rawmsg" format="json")
    constant(value="\",\"message\":\"")
        property(name="msg" format="json")

    constant(value="\",\"pri\":\"")
        property(name="pri" format="json")
    constant(value="\",\"pri-text\":\"")
        property(name="pri-text" 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="\",\"protocol-version\":\"")
        property(name="protocol-version" format="json")
    constant(value="\",\"timestamp\":\"")
        property(name="timestamp" dateFormat="unixtimestamp")
    constant(value="\",\"hostname\":\"")
        property(name="hostname" format="json")
    constant(value="\",\"fromhost-ip\":\"")
        property(name="fromhost-ip" 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="\",\"input-module\":\"")
        property(name="inputname" format="json")
    constant(value="\",\"proxy-uuid\":\"")
        constant(value="<%= `cat /opt/rb/etc/rb-uuid | tr -d "\n\r"` %>")
    constant(value="\",\"parsed-info\":")
        property(name="$!")
    constant(value="}")
  }

if $app-name == 'kernel' and $msg contains 'iptables' <% if
  node["redBorder"]["proxy"]["rsyslog"]["enable_client_proxy_logs"] %>
  and $fromhost-ip != '127.0.0.1' <% end %> then
{
#Extracción campos mediante reglas, usando el modulo liblognorm
  action(type="mmnormalize" ruleBase="<%=
  node["redBorder"]["proxy"]["rsyslog"]["rules_dir"] %>/iptables.rules")

#Copia local en un fichero.
& action(type="omfile"
         file="<%=
  node["redBorder"]["proxy"]["rsyslog"]["remote_logs_dir"]
  %>/norm_iptables.log"
         template="norm_iptables"
         )
```

```
#Acción para enviar los logs a Apache Kafka
&   action(type="omkafka" topic="rb_vault" broker="localhost:9092"
            template="norm_iptables"
                      )
  Stop
}
```

/redBorder-proxy/templates/default/rsyslog_06-parse_Cisco_WLC.conf.erb

```
template(name="norm_Cisco_WLC" type="list"){
    constant(value="{")
    constant(value="\"raw-message\":\"")
        property(name="rawmsg" format="json")
    constant(value="\",\"message\":\"")
        property(name="msg" format="json")

    constant(value="\",\"pri\":\"")
        property(name="pri" format="json")
    constant(value="\",\"pri-text\":\"")
        property(name="pri-text" 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="\",\"protocol-version\":\"")
        property(name="protocol-version" format="json")
    constant(value="\",\"timestamp\":\"")
        property(name="timestamp" dateFormat="unixtimestamp")
    constant(value="\",\"hostname\":\"")
        property(name="hostname" format="json")
    constant(value="\",\"fromhost-ip\":\"")
        property(name="fromhost-ip" 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="\",\"input-module\":\"")
        property(name="inputname" format="json")
    constant(value="\",\"proxy-uuid\":\"")
        constant(value="<%= `cat /opt/rb/etc/rb-uuid | tr -d "\n\r"` %>")
    constant(value="\",\"parsed-info\":")
        property(name="$!")
          constant(value="}")
  }

  if $app-name == 'WLC1' <% if
    node["redBorder"]["proxy"]["rsyslog"]["enable_client_proxy_logs"] %>
    and $fromhost-ip != '127.0.0.1' <% end %> then
  {
  #Extracción campos mediante reglas, usando el modulo liblognorm
    action(type="mmnormalize" ruleBase="<%=
    node["redBorder"]["proxy"]["rsyslog"]["rules_dir"]
    %>/Cisco_WLC.rules")
```

```
#Acción para enviar los logs a Apache Kafka
&   action(type="omkafka" topic="rb_vault" broker="localhost:9092"
            template="norm_Cisco_WLC"
                        )
  stop
}
```

/redBorder-proxy/templates/default/rsyslog_06-parse_Cisco_WLC.conf.erb

```
template(name="norm_rfc5424" type="list"){
   constant(value="{")
    constant(value="\"raw-message\":\"")
        property(name="rawmsg" format="json")
    constant(value="\",\"message\":\"")
        property(name="msg" format="json")

    constant(value="\",\"pri\":\"")
        property(name="pri" format="json")
    constant(value="\",\"pri-text\":\"")
        property(name="pri-text" 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="\",\"protocol-version\":\"")
        property(name="protocol-version" format="json")
    constant(value="\",\"timestamp\":\"")
        property(name="timestamp" dateFormat="unixtimestamp")
    constant(value="\",\"hostname\":\"")
        property(name="hostname" format="json")
    constant(value="\",\"fromhost-ip\":\"")
        property(name="fromhost-ip" 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="\",\"structured-data\":\"")
        property(name="structured-data" format="json")
    constant(value="\",\"tag\":\"")
        property(name="$!rfc5424-sd!rbvault@39483!tag" format="json")
    constant(value="\",\"sensor-uuid\":\"")
        property(name="$!rfc5424-sd!rbvault@39483!sensor-uuid"
        format="json")
    constant(value="\",\"hash\":\"")
        property(name="$!rfc5424-sd!hash@39483!hash" format="json")

    constant(value="\",\"inputname\":\"")
        property(name="inputname" format="json")
    constant(value="\",\"proxy-uuid\":\"")
        constant(value="<%= `cat /opt/rb/etc/rb-uuid | tr -d "\n\r"` %>")
    constant(value="\"}")
  }
```

```
<% if node["redBorder"]["proxy"]["rsyslog"]["enable_client_proxy_logs"]
   %>
if $fromhost-ip != '127.0.0.1' then
{
<% end %>
   action(type="omfile"
          file="<%=
   node["redBorder"]["proxy"]["rsyslog"]["remote_logs_dir"]
   %>/norm_rfc5424.log"
          template="norm_rfc5424"
          )
#Acción para enviar los logs a Apache Kafka
&    action(type="omkafka" topic="rb_vault" broker="localhost:9092"
            template="norm_rfc5424"
                      )
   stop
<% if node["redBorder"]["proxy"]["rsyslog"]["enable_client_proxy_logs"]
   %>
{
<% end %>
```

/redBorder-proxy/templates/default/iptables.erb

```
<% if node["redBorder"]["proxy"]["rsyslog"]["enable_tcp"] %>
 -A INPUT -m state --state NEW,ESTABLISHED,RELATED -m tcp -p tcp --dport
    <%= node["redBorder"]["proxy"]["rsyslog"]["tcp_port"] %> -j ACCEPT
<% end %>
<% if node["redBorder"]["proxy"]["rsyslog"]["enable_udp"] %>
 -A INPUT -m udp -p udp --dport <%=
    node["redBorder"]["proxy"]["rsyslog"]["udp_port"] %> -j ACCEPT
<% end %>
```