Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

Trigo Caparros, Francesc Xavier; García Font, Victor, dir. Desarrollo de un sistema web para la monitorización de redes en tiempo real. 2021. (958 Enginyeria Informàtica)

This version is available at https://ddd.uab.cat/record/238444

under the terms of the  license

Desarrollo de un sistema web para la monitorización de redes en tiempo real

Francesc X. Trigo Caparros

Resumen– Hoy en día, gran parte de las empresas dependen de aplicaciones online para abastecer a sus clientes y proporcionales un servicio de calidad. Por lo tanto, es esencial tener herramientas de monitorización que garanticen la calidad y operatividad de la infraestructura a medio-largo plazo. Existen multitud de soluciones software para este propósito, pero todas ellas se basan en tecnologías difícilmente mantenibles e inseguras debido a su antigüedad. Este artículo describe el desarrollo de un sistema web para la monitorización de redes de cómputo en tiempo real haciendo uso de las tecnologías más ágiles y orientadas a la concurrencia con el fin de estudiar su impacto. Para ello, se implementa una aplicación que envía información sobre el uso del host, y otra aplicación que las reciba, procesa y almacena en una base de datos, para posteriormente ser consultadas mediante una página web.

Palabras clave– Administración de redes, Base de datos de series temporales, Monitorización, Tiempo real

Abstract– Businesses of these days rely on online applications to provide high quality services to their clients. So it is essential to implement the correct monitoring tools to guarantee quality and functionality of the network infrastructure for the mid to long term. There are a lot of monitoring solutions, but they are based on hardly maintainable and unsafe technologies due to his age. This paper describes the development of a web monitoring system for network infrastructures in real time using agile and concurrent technologies to study it's impact. To achieve that, an application will be implemented to send data about the host's usage, and another application will be implemented to receive, process and store in a database to be requested by a website.

Keywords– Network management, Moniroting, Real-time, Time series database

1 Introducción

Existe una gran variedad de herramientas de monitorización, pero todas ellas se basan en C, un lenguaje de medio nivel procedimental. El principal inconveniente de este lenguaje es la seguridad, ya que según WhiteSource, abarca más del 47% del total de vulnerabilidades registradas [1], convirtiéndolo en el lenguaje más inseguro. No solo eso, la gestión manual de memoria y la comprobación de tipos de datos en tiempo de ejecución, hacen que este lenguaje sea difícil de mantener y que la ejecución de aplicaciones sea impredecible, dificultando el control de errores.

A pesar de ello, los principales sistemas de monitorización de redes como Nagios, Icinga2 o Zabbix siguen utilizando C para desarrollar sus sistemas de monitorización.

Además, Nagios o Icinga2, siguen utilizando bases de datos poco eficientes y muy limitadas como RRDTool o MySQL para guardar metricas, que son datos sobre el uso y estado de los hosts en un determinado instante de tiempo.

Sin embargo, hay aplicaciones escritas en C como Tor y Mozilla que están migrando su código a nuevos lenguajes en auge como Rust con el fin de mejorar la mantenibilidad, seguridad y rendimiento.

Del mismo modo, hoy en día existen bases de datos en auge orientadas a almacenar de forma eficiente métricas, como InfluxDB o TimescaleDB.

Haciendo uso de estas nuevas tecnologías, es posible conseguir, no solo un sistema de monitorización de redes más seguro y mantenible, sino más fiable, robusto y eficiente en

• E-mail de contacte: francescxavier.trigo@e-campus.uab.cat

• Menció realitzada: Tecnologies de la Informació

• Treball tutoritzat per: Victor García Font(Departament d'Enginyeria de la Informació i de les Comunicacions Àrea de Ciències de la Computació i Intel·ligència Artificial)

• Curs 2020/2021

comparación a los que hay en el mercado.
El propósito de este artículo es implementar un sistema de monitorización web utilizando las tecnologías más recientes, con el fin de obtener un sistema de monitorización más mantenible, fiable y seguro que el resto de alternativas.
Para ello, este artículo explica las fases de desarrollo del proyecto. Empezando por la metodología que se va a utilizar para desarrollar el proyecto, junto a los objetivos de este. Seguidamente, los conocimientos previos para entender el diseño del sistema de monitorización. Posteriormente se comenta como se ha implementado cada uno de los componentes del sistema. Y finalmente se valora la planificación del proyecto, se explica las líneas de trabajo a realizar en un futuro y se exponen las conclusiones.

2 METODOLOGÍA

Este proyecto ha utilizado la metodología Kanban, una metodología ágil para conocer el estado de cada una de las tareas, con el objetivo de conocer en todo momento el estado del proyecto mediante el número de tareas completadas.
Esta metodología destaca por su simplicidad y flexibilidad, sin embargo ha sido necesario adaptarla a este proyecto para abordar su complejidad.
Concretamente, ha sido necesario agrupar las tareas en fases, ya que existen tareas que no pueden ser realizadas si no se completa una determinada fase. Por ejemplo, las tareas que pertenecen a la fase de implementación, no pueden ser empezadas hasta que no finalice la fase de diseño.

3 OBJETIVOS

El principal objetivo de este proyecto es desarrollar un sistema de monitorización en tiempo real utilizando tecnologías recientes con el fin de estudiar su impacto. Para alcanzar este objetivo, hace falta satisfacer los siguientes subobjetivos:

- Realizar un estudio sobre la gestión de redes, para conocer que es monitorización de redes y su propósito.
- Analizar los lenguajes de programación y bases de datos que se ajustan a las características de este sistema de monitorización.
- Diseñar el sistema de monitorización flexible, que permita integrar fácilmente nuevas tecnologías que mejoren la red.
- Implementar el sistema de monitorización utilizando tecnologías ágiles y concurrentes.
- Realizar pruebas de rendimiento para compararla con otros sistemas de monitorización.
- Documentar el desarrollo del proyecto, el resultado de las pruebas y las conclusiones de estas.

4 BACKGROUND

Esta sección recoge todos los conocimientos previos para entender el desarrollo del proyecto. Para ello, primero se explica que es y en que consiste la gestión de redes, que es un sistema gestor de redes y que protocolos se utilizan para la gestión de redes.

4.1 Gestión de redes

La gestión de redes es un conjunto de disciplinas, concretamente *operations, administration, maintenance and provisioning* (OAMP).
Sin embargo, en el caso de las redes de cómputo se menciona OAM debido a que el *provisioning* consiste en la instalación de nuevos elementos o la introducción de nuevas tecnologías (referente a las redes de telecomunicaciones).
Cada una de estas disciplinas agrupa un conjunto de responsabilidades, tal y como define M. Subramanian [2]

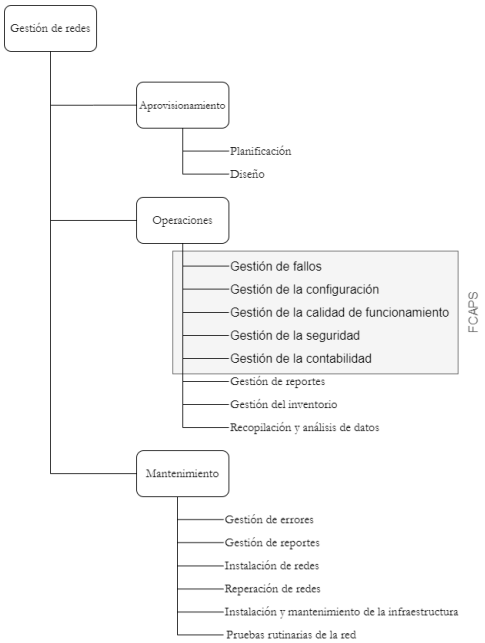

Fig. 1: Responsabilidades OAM

Por otra parte, con el fin de homogeneizar y centralizar la gestión de redes, se obtiene las responsabilidades que tienen en común todos los tipos de red, que son *fault, configuration, accounting, performance and security* (FCAPS), definido en la recomendación M.3400 (02/2000) [3] del *International Telecommunication Union* (ITU).
A partir del documento anterior, se puede concluir que la monitorización de redes es un subconjunto de las FCAPS (sin la gestión de seguridad). De esta forma, el sistema de monitorización se orientará a cumplir parte de las funcionalidades descritas de la recomendación M.3400 (02/2000) con el fin de asegurar la compatibilidad y centralización de la gestión de redes, característica fundamental de un sistema de monitorización de redes.

4.2 Network Management System

Un *network management system* (NMS) o sistema de gestión de redes, es un conjunto de aplicaciones que abarca las áreas funcionales de los FCAPS, con la finalidad de servir como base para la implementación de sistemas que tengan como fin gestionar la red. La figura 2 representa la arquitectura de un NMS de redes TCP/IP.

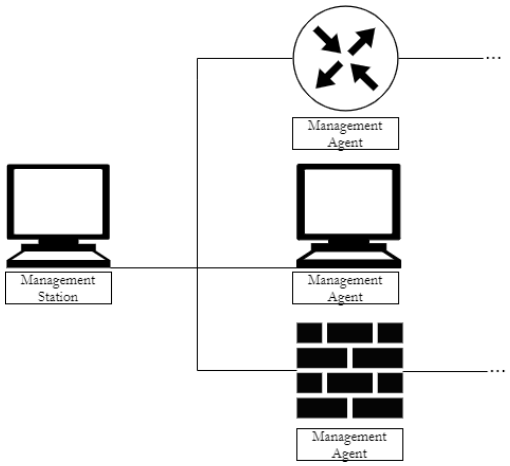

Fig. 2: Arquitectura de un NMS

4.3 Protocolo de gestión de redes

Con el objetivo de que tanto administrador como el agente puedan intercambiar información como configuración o datos de uso, es necesario un protocolo que haga efectiva esta comunicación, independientemente del *hardware* y *software* del agente.
Para ello, existen diferentes protocolos de gestión de redes TCP/IP como SGMP y CMIT, pero ambos quedaron obsoletos ante la llegada de SNMP definido en el estándar RFC 1157. El protocolo se compone de 5 operaciones básicas (7 en las versiones SNMPv2 y SNMPv3): *get*, *getnext*, *set*, *getresponse* y *trap*. Debido a su simplicidad, popularidad y compatibilidad, sigue siendo el protocolo de gestión de redes más usado a pesar de otras alternativas como NETCONF, definido en el estándar RFC 4741.
Al igual que en un NMS, el protocolo SNMP se compone del agente y el administrador. El agente contiene un base de datos llamada *Management Information Base* (MIB) que define los elementos monitorizables del sistema (objetos). El administrador accede a cada uno de los objetos de la MIB mediante un *object identifier* (OID). La figura 3 representa la arquitectura del protocolo SNMP.

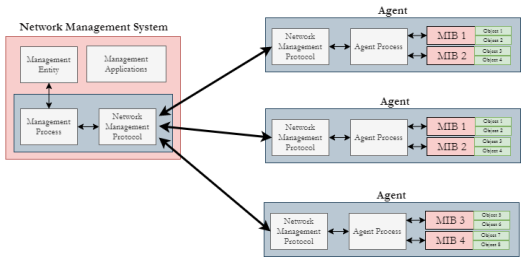

Fig. 3: Arquitectura del protocolo SNMP

Finalmente, SNMP incorpora los *traps* que es un mensaje para alertar al administrador en caso de fallo. Ahora bien, la principal desventaja de este mecanismo es que no es posible garantizar la entrega del mensaje. Sin embargo, en versiones superiores de SNMP (SNMPv2 y SNMPv3), incorpora una nueva operación llamada *inform*, que permite confirmar la llegada de la alerta.

4.4 Monitorización de redes

A partir de lo anterior, la monitorización de redes es una área funcional de la gestión de redes, cuya propósito es garantizar la disponibilidad y funcionamiento óptimo de los agentes, mediante el análisis y procesamiento de las métricas obtenidas de los agentes y la gestión de la configuración de los agentes.
Para ello, se utiliza los sistemas de monitorización de redes, una parte de los NMS compuesto por 2 componentes, el agente (dispositivo a monitorizar) y el administrador.
Para que el agente y el administrador puedan compartir esa información, utilizan el protocolo gestión de redes que permite asegurar la compatibilidad entre ambos, independientemente del *hardware* o *software* en el que se ejecute.

5 Diseño

A partir de esta sección es donde se empieza a construir el proyecto. Empezando por definir la arquitectura del sistema para dar una visión global de este, el protocolo de comunicación entre el agente y el administrador, donde se explica los datos que se usan y como se envían, y finalmente cada uno de los servicios y aplicaciones que componen el sistema, concretando sus responsabilidades y como se ha modelado.

5.1 Arquitectura del sistema

Con el fin de obtener un sistema de monitorización, no solo fácilmente actualizable y mejorable, sino adaptable a los nuevos cambios que sufra la infraestructura de la red, la arquitectura del sistema se basa en el patrón *Composable Monitoring* descrito por Mike Julian (2018), que consiste en: “usar múltiples herramientas especializadas, mínimamente acopladas“ (pp. 27-28).
Para ello, Mike Julian sugiere que el sistema está compuesto las siguientes herramientas:

- **Colección de datos (Processor)**: Un servicio para obtener series temporales, es decir, un conjunto de métricas en un determinado instante de tiempo.
- **Almacenamiento de datos (Data base)**: Una base de datos flexible para, no solo para almacenar series temporales de forma eficiente, sino almacenar las notificaciones e información de los agentes y usuarios.
- **Visualización (Web Service)**: Una aplicación web para que el administrador visualice los datos en tiempo real de forma rápida y sencilla, con el objetivo de que reducir el tiempo para tomar una decisión.
- **Alertas (Notify Service)**: Un servicio que envía las alertas por diferentes medios de comunicación, con el fin de aumentar la probabilidad de que el administrador reciba la notificación.

Es cierto que el autor menciona la herramienta "Análisis y Reporte", pero ha sido descartada debido a que el sistema está orientado a conocer el estado de los agentes, no a garantizar y demostrar la disponibilidad de la red mediante un *Service Level Agreement* (SLA).

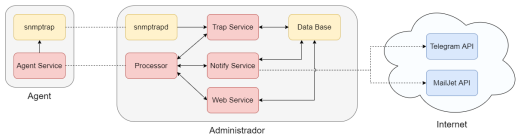

Fig. 4: Arquitectura del sistema

Para posteriores referencias, el *backend* está formado por los servicios Processor, Trap y Notify, mientras que el *frontend* está formado por el servicio Web y la aplicación web.
Por otra parte, de la figura 6 cabe destacar que la comunicación entre las herramientas del administrador es realizada mediante *pipes* (a excepción de la base de datos, que se realiza mediante *socket*), debido a que es el *Inter-process communication* (IPC) más sencillo y eficiente, a diferencia de otros como los Sockets que requiere más llamadas de sistema; o la memoria compartida que, a pesar de ser más rapida que los *pipes*, es mucho más complejo de implementar. Se ha considerado usar los *named pipes*, ya que permiten tener un *pipe* bidireccional y persistente, aumentando la fiabilidad y eficiencia de la comunicación. Sin embargo, una de las principales desventajas es su implementación, ya que depende del sistema operativo, lo que provoca que aumente la complejidad del código y limite su compatibilidad.

5.2 Protocolo de comunicación

La comunicación del agente y administrador, no solo se basa en enviar métricas, sino también en enviar comandos para autorizar el agente o comprobar la disponibilidad del administrador. Como la comunicación se realiza sobre redes TCP/IP, ha sido necesario diseñar el datagrama que se muestra en la figura 5 con en fin de diferenciar comandos y métricas.

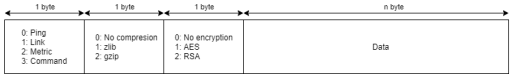

Fig. 5: Datagrama

Una de las ventajas que permite la encapsulación de datos, es poder implementar algoritmos de compresión, con el propósito de reducir la probabilidad de *overflow* del socket UDP del servidor y aumentar el ancho de banda de la red; O algoritmos de encriptación, para asegurar la confidencialidad de los datos en caso de que la red no sea segura.

Es importante destacar que los comandos se envían mediante un socket TCP, ya que al tratarse de acciones críticas, es necesario un mecanismo fiable que garantice la llegada del comando. Mientras que las métricas, al enviarse constantemente, es necesario usar un socket UDP, ya que es más rápido y eficiente, aunque no garantice la llegada de las métricas.
Hasta este punto, se han diseñado los siguientes datagramas:

- **Metric**: Es utilizado para enviar las métricas de uso de CPU y RAM del agente. También incluye el *timestamp* del momento en el que se ha obtenido las métricas (ya que un datagrama puede ser procesado tardiamente por el administrador) y el id del agente, para poder identificarlo.

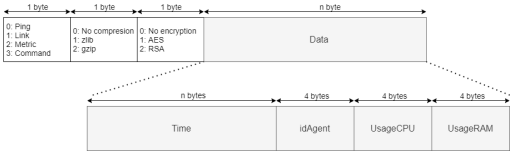

Fig. 6: Datagrama metric

- **Ping**: Comprueba si el administrador está disponible y completamente funcional.
- **Link**: Enlaza el agente con el administrador.

Es importante añadir, que las únicas métricas que se pueden obtener con este sistema es el uso de RAM y CPU, ya que son las métricas más básicas para conocer el estado del agente. El resto de métricas como el número de paquetes entrantes y salientes de una interfaz de red o el uso de CPU y RAM de una determinada aplicación, quedan como futuro trabajo.

5.3 Servicio Processor

Es el responsable de procesar las métricas y peticiones de los agentes, además de gestionar el resto de servicios. El propósito de este servicio es centralizar el control de las herramientas del sistema de monitorización y llamar al servicio requerido en función del comando.
Ahora bien, con el fin de desacoplar la recepción del *trap* de su posterior notificación, ha sido necesario aplicar el patrón software *Chain-of-responsibility* tal y como se puede ver en la figura 7. Este patrón permite desacoplar el emisor del receptor. De esta manera, a largo plazo es posible implementar una nueva herramienta automaticé las acciones a realizar dependiendo del *trap*.
También ha sido necesario aplicar el patrón *Dependency Injector*, para que los comandos puedan usar cualquier objeto sin tener que ser pasado por parámetro. Este patrón utiliza un container, que guarda los punteros de los objetos registrados en el este (junto a un nombre para ser referenciados). De este modo, lo único que se pasa como parámetro es el puntero del container, no los objetos. Gracias a este patrón, el código es más legible y mantenible, ya que no es necesario gestionar los parámetros de los comandos.

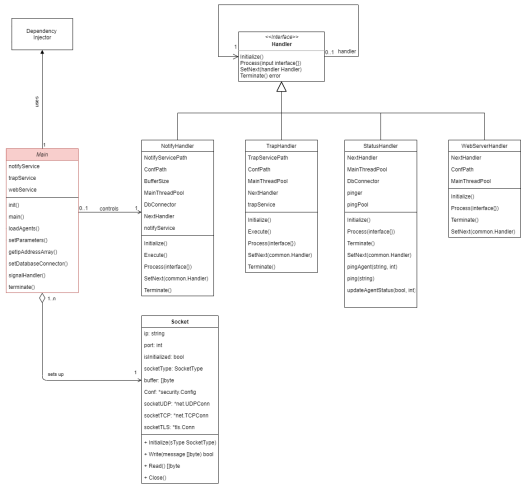

Fig. 7: Diagrama de clases - Servicio Processor

5.4 Aplicación Notify

Este servicio se encarga de obtener las notificaciones de la base de datos y enviarlas a una *Application Programming Interface* (API).
Cada API requiere de diferentes parámetros como el teléfono o email para enviar una notificación, por lo que no es posible implementar una interfaz que permita homogeneizar el envío de notificaciones. Para solventar este problema, ha sido necesario implementar el patrón software *vistor*, tal y como se puede ver en la figura 8. Ya que este patrón que permite separar la implementación de la estructura.

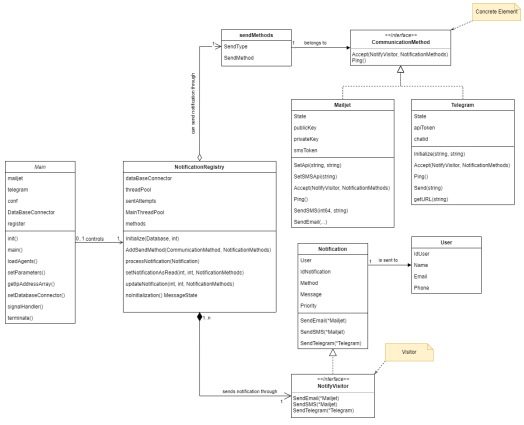

Fig. 8: Diagrama de clases - Servicio Notify

5.5 Servicio Trap

Es el responsable de recibir y parsear los traps. Para ello, este servicio actúa como intermediario entre el servicio Processor y la implementación del protocolo de gestión de redes, con el fin de poder implementar cualquier protocolo de gestión de redes.

5.6 Servicio Web

Es el responsable de ofrecer la interfaz gráfica de usuario. Para que los administradores de redes puedan comprobar el estado y uso de los agentes en tiempo real, desde cualquier dispositivo, la interfaz de usuario será accedida mediante navegador.

6 IMPLEMENTACIÓN

Esta sección explica las tecnologías usadas en el sistema, con el fin de justificar su uso y compararla con otras alternativas.
Cabe añadir, únicamente se describirá las principales características y funcionalidades de los componentes, además de como funcionan, sin indagar en la codificación.

6.1 Base de datos

Tal y como se ha mencionado en la sección 5.1, los datos a procesar son series temporales (además de la información de los agentes y usuarios). Las *time series database* (TSBD) son un tipo de base de datos optimizadas para series temporales. A diferencia de otras bases de datos, los TSBD no solo están orientados a almacenar y procesar de manera eficiente las series temporales, sino a asegurar la coherencia y consistencia de los datos limitando las operaciones a 4: selección, inserción, agrupación y agregación. Además de ofrecer mecanismos para reducir el tamaño de la base de datos mediante la agrupación de series temporales antiguas.
Con el fin de no estar únicamente limitados a utilizar series temporales y poder adaptarse a nuevos cambios o mejoras, la base de datos del sistema de monitorización es administrada por el sistema gestor de bases de datos (SGBD) PostgreSQL junto a la extensión TimescaleDB. A diferencia de otras bases de datos como InlfuxDB, TimescaleDB ofrece una mayor tolerancia a fallos y fibiabilidad tal y como demuestra el artículo *Benchmarking TimescaleDB vs. InfluxDB for Time-Series Data* [5]. Además de poder implementar funciones PL/SQL con el fin de hacer el código del sistema más legible, ofrecer transparencia y asegurar la atomicidad de la transacción.
Cabe destacar que TimescaleDB está basado en la escalabilidad vertical. Esto puede suponer que a largo plazo esté más limitado si no se utiliza el *hardware* adecuado. Sin embargo, TimescaleDB ofrece técnicas para mitigar los problemas de escalabilidad como la compresión de datos.

6.2 Leanguaje de programación

6.2.1 Backend

La mayoría de los sistemas de monitorización como Nagios, Icinga2 o Zabbix, están escritos el lenguaje de programación C.
Sin embargo, existen alternativas en auge como Go y RUST, que a diferencia de C se orientan al desarrollo de software ágil, mantenible y concurrente, eliminando por completo la gestión de memoria.
Dicho lo anterior, ambos se basan en el principio *Composition over Inheritance* que permite implementar un sistema más robusto, escalable y fácil de testear debido a la fuerte encapsulación de los objetos. Por ende, tanto Go como RUST no implementan mecanismos de programación orientada a objetos (POO) como la herencia o la sobrecarga de métodos, lo que provoca que la fase de diseño sea más compleja.
Es cierto que ambos comparten gran parte de sus características, pero finalmente se ha elegido Go debido a la poca pronunciación de su curva de aprendizaje, convirtiéndolo en un lenguaje fácil de aprender a diferencia de RUST.

6.2.2 Frontend

Con el objetivo de diseñar un sistema web en tiempo real, el frontend se basará en el modelo *Single-Page Application* (SPA), ya que se consigue una experiencia más fluida, además de reducir la carga del servidor.
Existen diferentes técnicas para implementar un SPA, pero la más usada y extendida son los *Frameworks* de Javascript debido a que ofrecen una capa de abstracción sobre Javascript, con el objetivo de diseñar y desplegar rápidamente aplicaciones web. Debido a su popularidad, se destacan Angular y Vue.js.
Angular proporciona una de las características de un sistema de monitorización web, la robustez. Aunque, su principal desventaja es la curva de aprendizaje debido a que usa Typescript, un lenguaje más complejo que Javascript debido a su fuerte tipado. No solo eso, Angular hace uso del DOM real, por lo que es más difícil encontrar errores y reduce su rendimiento, en comparación al uso del DOM virtual.
Por lo tanto, se escoge Vue.js debido a su sencillez, rapidez, ligereza, eficiencia, curva de aprendizaje y, además, permite el *two-way databinding* en comparación a Angular.

6.3 Protocolo de comunicación

Para agilizar el desarrollo del sistema y asegurar su compatibilidad con todas las versiones SNMP, se ha utilizado el paquete Net-SNMP, que ofrece un conjunto de servicios y aplicaciones para usar el protocolo SNMP. Concretamente, se han utilizado las siguientes herramientas:

- **snmptrap**: Comando para enviar traps
- **snmptrapd**: Servicio para recibir traps
- **snmpd**: Servicio para recibir peticiones SNMP

De todos modos, el sistema hace uso del protocolo SNMP únicamente para enviar y recibir traps, no para recibir y enviar metricas. Estas se envian mediante el datagrama especificado en la seccion 5.2. Esta decisión se debe a los siguientes motivos:

- **Complejidad**: Un dispositivo puede ser compatible con una MIB, pero no integrar todas sus subramas. Lo que provoca que la base de datos quede incompleta y sea ineficiente, debido a la gran cantidad de valores nulos por tupla.
- **Obsolescencia**: La antiguedad de las MIBs como RMON2 (2006) o MIB-II (1991) limita la adaptación de nuevas tecnologias de la capa física como la fibra óptica, Wifi o Lifi. Esto se debe a la imposibilidad de actualizar las MIBs debido a su estandarización y a su fuerte tipado. La solución pasa por usar MIBs privadas, lo que provoca no solo aumentar la complejidad del sistema, sino poner en compromiso la legalidad del sistema, ya que se requiere de licencias para usarlas y distribuirlas.
- **Ambigüedad léxica**: Las MIBs son simplemente un estadandar, cada empresa puede describir los objetos de la MIB, a su manera, provocando ambigüedad e inconsistencia en los datos. Por ejemplo, la empresa Circitor interpreta el objecto *etherStatsPkts*, como "*El número total de paquetes (incluyendo los paquetes de errores) recibidos.*", mientras que Cisco la interpreta como "*El número total de paquetes (incluyendo paquetes erróneos, broadcast, y multicast) recibidos*" [6].

Esta decisión implica omitir la recomendación 4.1 del RFC 1052 de "*adoptar el protocolo SNMP como base para la administración de redes en todo el sistema*" [7]. Sin embargo, con el objetivo de mantener la compatibilidad con otros NMS basados en SNMP, el agente implementa el servicio *snmpd* para poder recibir peticiones de administradores SNMP.

6.4 Servicio Processor

Al tratarse del servicio central, ha sido necesario centralizar la configuración de todos los servicios con el fin de simplificar la configuración del sistema. Para ello, la configuración se almacena en un fichero en formato YAML, debido a que es el lenguaje más intuitivo y legible por los humanos, tal y como se puede apreciar en la figura 9.

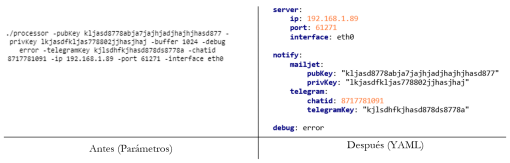

Fig. 9: Ejemplo de configuración

Sin embargo, una de las principales desventajas es que la configuración es parseada en una estructura donde gran parte de sus elementos son nulos, ocupando espacio innecesariamente.
Despues de verificar y parsear la configuración, el servicio crea los Sockets mediante la estructura Socket. Esta estructura es uno de los componentes esenciales del sistema, ya que permite utilizar las 3 primitivas básicas de los sockets (*READ*, *WRITE*, *CLOSE*) independientemente del protocolo de la capa de transporte que se esté usando (UDP, TCP o TCP/TLS), ofreciendo transparencia al programador y simplificando la gestión de la conexión.
Con el fin de flexiblizar el procesamiento de datagramas, por cada conexión (TCP) o datagrama entrante (UDP) se llama a un *callback* para procesarlo. Esta estrategia permite centralizar la gestión de errores, simplificar el código y eliminar código redundante, haciendo más legible y mantenible.
Ahora bien, a nivel de código TCP y UDP tienen una gran diferencia, la concurrencia. En el caso TCP, cada conexión es gestionada por una *goroutine* (similar a un *thread* pero más ligero), ofreciendo un alto grado de paralelización a pesar de consumir más recursos. En cambio, los sockets UDP se basan en un único buffer que se va leyendo al mismo tiempo que se va escribiendo. Eso implica a que la concurrencia queda limitada al número de datagramas que caben en el buffer y a la probabilidad de *overflow* del buffer.
Respecto al datagrama Link de la sección 5.2, cabe destacar que únicamente se ha tenido en cuenta 3 escenarios alternativos, tal y como se puede ver en la figura 10: cuando el token del agente no concuerda, cuando el agente ya existe

en la base de datos y cuando el agente no recibe respuesta. Es evidente que existen más escenarios de excepción, pero debido a la falta de tiempo y complejidad del proyecto, únicamente se han implementado los más probables y esenciales.

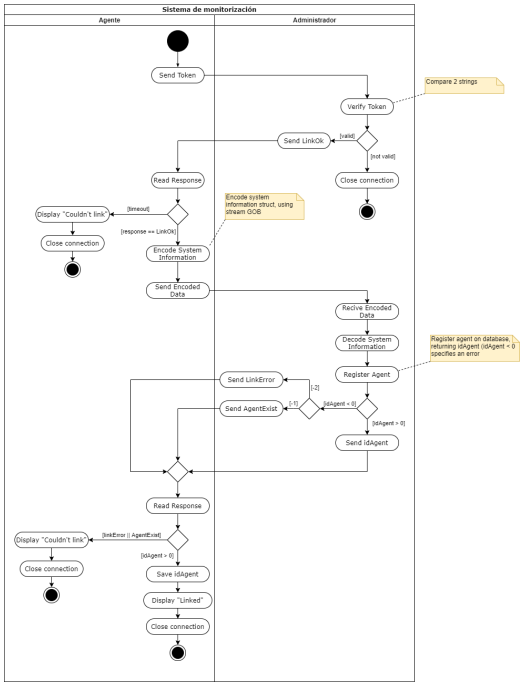

Fig. 10: Diagrama de actividades

6.5 Servicio Trap

Este servicio se encarga de ejecutar y gestionar el servicio *snmptrapd* descrito en la sección 6.3, con el objetivo de obtener los traps y conocer el estado del servicio en todo momento.
Para ello, el servicio Trap ejecuta el servicio *snmpdtrapd* para crear un *pipe*. De esta forma, se evita hacer uso de técnicas más ineficientes como *polling* para conocer su estado.
Cabe destacar que para obtener los datos de *snmptrapd* al instante (sin esperar a que su buffer de salida se llene), ha sido necesario el uso del comando *stdbuf* con el fin de forzar la descarga de los datos el buffer de salida de *snmptrapd*, tal y representa la figura 11.

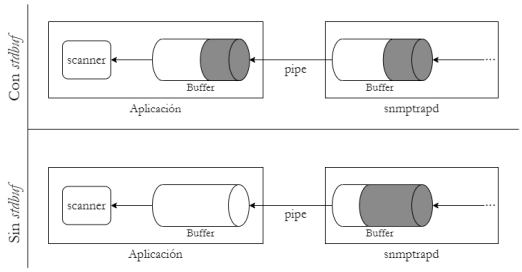

Fig. 11: Arquitectura del sistema

Finalmente, ha sido necesario implementar un gestor de señales (*signals*) para terminar el servicio *snmptrapd* correctamente, ya que si únicamente finaliza la aplicación padre (servicio Trap), el *snmptrapd* queda en segundo plano. Esto provoca que en la próxima ejecución del servicio Trap, no pueda ejecutar *snmptrapd* debido a que ya existe una instancia de esta.

6.6 Aplicación Notify

Esta aplicación envía un mensaje a un grupo de Telegram o un Email, mediante la API de Telegram o la de Mailjet respectivamente.
Una de las principales características de este servicio es que si una notificación no ha podido ser enviada, no queda descartada sino que se vuelve a intentar hasta N veces (siendo N el número de intentos especificados por el usuario), con el fin de aumentar la tolerancia de fallos y fiabilidad de la aplicación.
Finalmente, cabe destacar que el envió de notificaciones está totalmente paralizado mediante *gorutines* con el fin de enviarlas lo más rápido posible.

6.7 Servicio WebService

Este servicio está compuesto por 3 componentes:

- **SPA**: El *frontend* de la aplicación implementado en Vue 2, ya que Vue 3, aun ser más rápido que el anterior, tiene incompatibilidades con algunos paquetes que se usan en esta SPA.
  Por otra parte, la SPA ha sido implementada utilizando el gestor de paquetes NPM, para simplificar la actualización e instalación de dependencias. Entre los paquetes usados, se pueden destacar:
  - **vue-chartjs**: Es un *wrapper* de librería *Chart.js*, para poder implementar sus gráficas mediante componentes Vue. Ha sido utilizada para crear la gráfica que permite visualizar las métricas de CPU y RAM en tiempo real.
  - **bootstrap-vue**: Nuevamente es un *wrapper* de la biblioteca Bootstrap 4, para poder implementar elementos de Bootstrap mediante componentes Vue. Su propósito es mejorar la experiencia de usuario, facilitar el diseño web adaptativo y hacer la interfaz más intuitiva.
  - **vue-notification**: Es un conjunto de componentes Vue que permite mostrar notificaciones similares a los *toasts*. Es utilizado para mostrar mensajes de error o de confirmación.
  - **vuex**: Es una librería con el objetivo de centralizar el estado de la aplicación. Es utilizada principalmente para almacenar el *token* de la sesion y gestionar los errores que provienen de *Socket.io*.
  - **axios**: Librería que proporciona un cliente HTTP basado en promesas. Es usada únicamente para autentificar al usuario.
- **API**: Su principal propósito es autentificar al usuario mediante un *JSON Web Token* (JWT) para posteriormente, aceptar la petición de conexión Websocket. De esta forma, se evita hacer uso de *cookies* que suelen estar limitadas por los navegadores web. Además de aumentar la compatibilidad con otros mecanismos de comunicación.

- **Socket.io**: Es una librería que implementa mecanismos de comunicación bidireccional como WebSockets, XHR-Polling o JSONP-Polling (dependiendo de la red), con el fin de ofrecer transparencia a la hora de usarlos. Es usado para enviar y recibir datos en tiempo real, además de enviar y recibir peticiones.

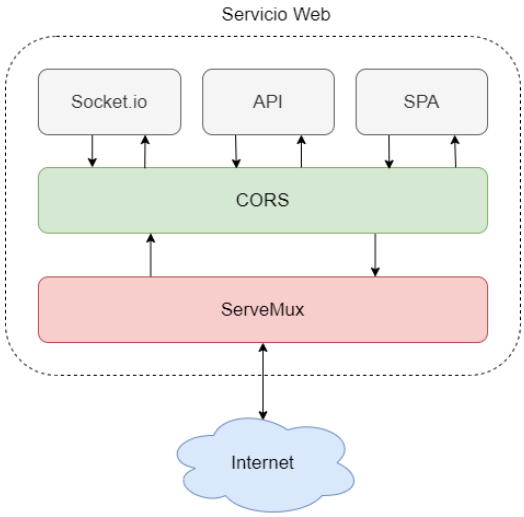

Fig. 12: Arquitectura del servicio Web

Finalmente cabe destacar que ha sido necesario incorporar el *middleware Cross-origin resource sharing* (CORS) de Go sobre el servidor, tal y como muestra la figura 12. De esta manera, tanto la API como Socket.io pueden ser accedidos desde un dominio diferente al del SPA, evitando errores relacionados con las políticas CORS que impiden el uso de la aplicación.

6.8 Agente

El agente ofrece 4 funcionalidades: él envió de métricas, la gestión del servicio *snmpd*, comprobar el estado del administrador y enlazarse con el administrador.
Para obtener las métricas de uso de CPU y RAM, se usa la librería *Prometheus* con el fin de evitar depender de más aplicaciones externas innecesariamente. Sin embargo, aunque esta Liberia sigue en desarrollo, se actualiza constantemente por lo que se reduce la posibilidad de fallos o incompatibilidades.
Esta librería accede al directorio *proc* como si fuese una estructura, facilitando su acceso y teniendo un mayor control de errores. El directorio *proc* es un sistema de ficheros en memoria que ofrece información del sistema, como por ejemplo el número de procesadores, la cantidad de memoria ocupada, o los procesos en ejecución.
Para calcular el uso de memoria se lee el archivo */proc/meminfo* para calcular la siguiente ecuación:

$$memUsage = (\frac{MemTotal - MemFree}{MemTotal}) * 100$$

Para el uso de CPU se lee el archivo */proc/stat* y se calcula mediante la siguiente ecuación:

$$nonIdle = User + Nice + System + Guest + IRQ + SoftIRQ + Steal + GuestNice$$
$$idle = Idle + IoWait$$
$$\Delta total = (idle + nonIdle) - (prevIdle + prevNonIdle)$$
$$\Delta idle = idle - prevIdle$$
$$cpuUsage = (\frac{\Delta total - \Delta idle}{\Delta total}) * 100$$

Los cálculos anteriores son la forma estándar de calcular el uso de CPU y RAM, ya que aplicaciones populares como *htop* lo utilizan para monitorizar el sistema.
Finalmente, es importante destacar que para obtener un valor consistente de la *cpuUsage*, tiene que haber un intervalo mínimo de 100ms con el fin de que la diferencia entre *prevIdle*, *prevNonIdle* respecto a *idle* y *nonIdle* sea lo suficientemente grande para obtener un tiempo de uso total realista y coherente.
Para conocer este valor, se ha forzado el uso de la mitad de los núcleos de la CPU mediante el comando *stress-ng* con el fin de llegar al 50% de uso de CPU. Por lo tanto, cuanto más alejado este el valor *cpuUsage* del 50% menos precisión tendrá. Dicho lo anterior, se ha probado en diferentes arquitecturas Intel y se ha obtenido los resultados que se muestran en la gráfica 13.

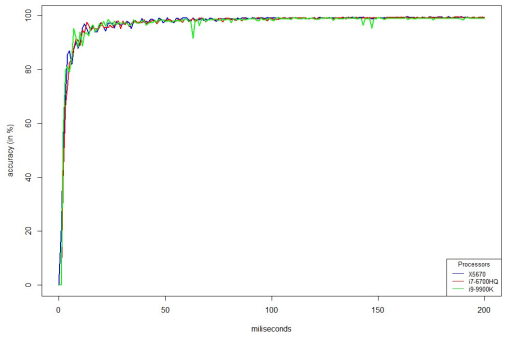

Fig. 13: Gráfica de precisión de uso de CPU

Tal y como se puede ver, la precisión se empieza ser estable a partir de los 100ms, siendo superior al 98%.

7 Trabajo Futuro

Es importante destacar que el proyecto aún sigue en fase de implementación, por lo que la prioridad a corto plazo es completar la planificación del proyecto.
Sin embargo, uno de los puntos a mejorar son el control de escenarios como que el usuario tenga una política estricta de firewall. Esto imposibilita utilizar la aplicación, ya sea como cliente o como administrador, por lo tanto, hay que encontrar una forma informar al usuario de que tiene que permitir el uso del puerto.
También es posible que uno o varios agentes que se encuentran detrás de un servidor proxy, lo que imposibilita el acceso a ellos. Por lo tanto, será necesario que crear una aplicación que actúe de proxy para acceder a dichos agentes.
No obstante, a lo largo de la implementación se han

omitido algunos escenarios de excepción o alternativos por falta de tiempo. Será necesario volver a revisar algunos de los elementos del sistema como los *sockets* o *pipes* para asegurar la robustez del sistema. Además, de diseñar, planificar y ejecutar test que corroboren y demuestren la robustez y estabilidad del proyecto.

8 Planificación

La planificación ha tenido que ser modificada con el fin de garantizar el éxito del proyecto y tener un sistema inmanente funcional para la entrega.

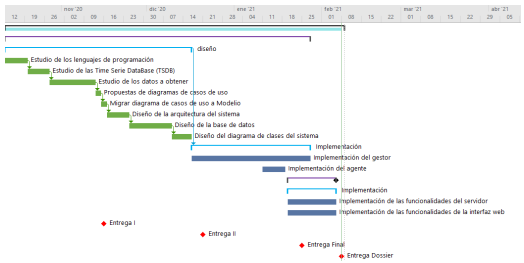

Fig. 14: Diagrama de Gantt

En la fase de implementación, se ha tenido que descartar el uso de SNMP para enviar y recibir métricas por las razones explicadas en la sección 6.3. Este cambio provocó no solo un restraso en el resto de actividades, sino un rediseño total de la base de datos y revisar de nuevo los datos a utilizar con el fin de garantizar un desarrollo fluido y esperado.
Por otra parte, debido a la inexperiencia, algunas tareas como el diseño de la base de datos o la arquitectura del sistema han requerido más tiempo del previsto, retrasando el resto de actividades.
Además, tareas como la implementación del administrador y la implementación del agente, han tenido que ser descompuestas con el fin de convertirlas en tareas más realistas, alcanzables y fáciles de gestionar. Esto ha supuesto revisar la planificación para aplicar los nuevos cambios.
Inicialmente, la implementación del *frontend* iba a ser posterior a la del *backend*, pero con el objetivo de obtener un sistema mínimamente utilizable para la entrega, ha sido necesario adelantar la fase de implementación del *frontend* y posponer la fase de pruebas del *backend*, tal y como se puede comprobar en la figura 14.

9 Resultados

A pesar de que el proyecto sigue en la fase de implementación y haya sufrido cambios a lo largo de la planificación, se ha obtenido un sistema de monitorización mínimamente operativo.
Hasta este punto se han implementado los servicios Processor, Trap y Web, y aplicación Notify con el fin de que el proyecto pueda continuar desarrollándose de forma fluida, sin grandes contratiempos y que pueda quedar terminado de corto a medio plazo.

Hasta este punto, el sistema es capaz de mostrar el estado de los agentes (figura 15), mostrar gráficas de uso de RAM y CPU en tiempo real (figura 16), enviar notificaciones por

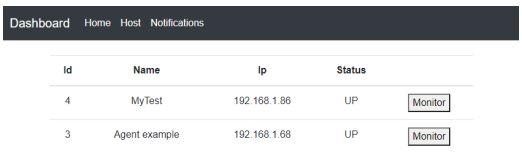

Fig. 15: *Dashboard* del sistema

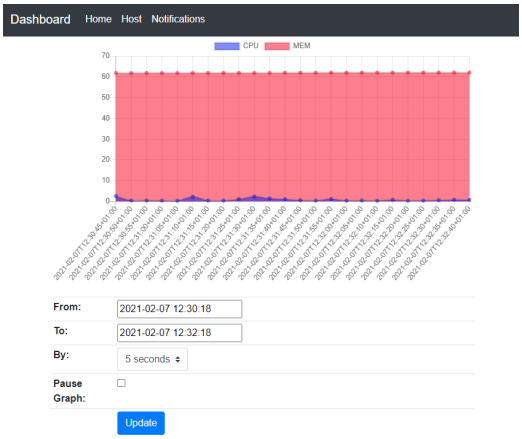

Fig. 16: Monitorización de uso de CPU y RAM

Telegram y Email, recibir y enviar traps, y enlazar el agente con el administrador para poder ser monitorizado.
Finalmente, el resultado obtenido es el de un sistema básico para conocer el estado de un conjunto de agentes según el uso de CPU y RAM que es capaz de enviar notificaciones por Telegram y Email para informar al administrador de redes en caso de que supere un cierto porcentaje de uso.
Por lo tanto, no se han cumplido con los objetivos debido a que se han cumplido los 3 primeros subobjetivos (estudio sobre la gestión de redes, análisis de los lenguajes de programación y bases datos, y el diseño del sistema). Sin embargo, el 4º aún sigue en desarrollo, ya que ha requerido más tiempo de lo esperado.

10 Discusión

Uno de los objetivos de SNMP es homogeneizar la administración y monitorización de redes mediante estándares. Sin embargo, esto impide implementar nuevas tecnologías para mejorar la red, ya que las tecnologías al año en el que se estandarizó (2006). Además de la dificultad de implementar nuevas MIBs.
Por lo tanto, no es de extrañar que empresas utilizan otras alternativas de SNMP para monitorizar sus dispositivos. Un ejemplo es Graphite usado por Booking.com para analizar los cambios en su infraestructura de red [8] o que sistemas como Icanga2 o Zabbix usen su propio protocolo.
Por otra parte, elegir Golang como lenguaje principal, ha permitido tener una buena gestión y control de los errores, a pesar de no utilizar mecanismos de excepción. De esta forma, se fuerza al programador a implementar código para controlar posibles errores, cosa que ayuda a obtener un sistema fiable.
Golang, ha demostrado tener una forma de implementar aplicaciones ligeramente diferente a la del resto de lenguajes orientado a objetos. El hecho de no tener constructores, parámetros opcionales, sobrecarga de métodos u otras

técnicas de polimorfismo, ha supuesto tener que refactorizar componentes esenciales del programa o utilizar patrones de software, lo que ha supuesto un coste temporal significativo. Sin embargo, esta filosofía de "forzar" al programador a diseñar estructuras sencillas con un propósito claro, ha permitido obtener un código legible, mantenible y robusto.
Finalmente, uno de los errores cometidos más importantes a lo largo del proyecto, ha sido no priorizar los requisitos con el fin de diseñar un sistema mínimamente funcional. Esto se debe a que se ha diseñado un sistema que ha intentado implementar gran parte de los requisitos, pero los esenciales no están aún totalmente completados, obteniendo un sistema "a medias".

11 Conclusión

Este artículo explica las fases de desarrollo para implementar un sistema de monitorización web. Empezando por el estudio sobre la gestión de redes, para definir la monitorización de redes como la gestión de fallos, configuración, calidad de funcionamiento, seguridad y contabilidad de la disciplina Operaciones de la gestión de redes.
Posteriormente, se ha concluido que todo sistema de monitorización de redes tiene que estar formado por el agente y el administrador, comunicados mediante un protocolo de gestión de redes. Sin embargo, durante la fase de implementación se ha descartado el uso de SNMP como protocolo de gestión de redes, debido a la complejidad, obsolescencia y ambigüedad léxica de algunos objetos de las MIBs estándares.
Por ese motivo, ha sido necesario diseñar un nuevo datagrama como alternativa a SNMP. Esto ha permitido, añadir algoritmos de compresión, con el fin de reducir la carga del servidor y el uso del ancho de banda de la red.
Por otra parte, el sistema de monitorización ha sido diseñado de forma modular con el objetivo de facilitar la adaptación y actualización de nuevas tecnologías. Para ello, se ha utilizado Go, que se basa en el principio *Composition over inheritance* el cual ha ayudado a implementar de forma modular el sistema, ya que reduce notablemente el acoplamiento entre objetos. Por otra parte, al trabajar con series temporales se ha utilizado TimescaleDB, una extensión para PostgreSQL que permite guardar de forma eficiente series temporales.
Finalmente, se ha implementado una aplicación web SPA implementado en Vue.js, que permite visualizar el uso de CPU y RAM en tiempo real mediante el uso de WebSockets. Sin embargo, ha sido necesario implementar un *middleware* en el servicio Web con el fin de evitar tener que utilizar el mismo dominio para la aplicación web y la API (CORS).

Agradecimientos

Me gustaría agradecer a todas las personas que me han estado apoyando y fomentando a lo largo de mi carrera educativa. Primero de todo a mi familia, ya que siempre han sido mi principal apoyo y han realizando grandes esfuerzos para que pueda tener un buen futuro. Segundo a mi mejor amigo Xavier Alegret, que siempre ha estado a mi lado en los momentos más difíciles de mi carrera. Y, finalmente, mi tutor Victor García por todo el tiempo que ha dedicado a darme consejos, orientarme y mejorar como ingeniero a lo largo de este proyecto.

Referencias

[1] WhiteSource (2019). Most Secure Programming Languages. Disponible en: https://www.whitesourcesoftware.com/most-secure-programming-languages/

[2] Subramanian, M. (2011), Network Management: Principles and Practises (ed.), Georgia Institute of Technology: Pearson. Goal of Network Management (p. 39)

[3] International Telecommunication Union. (2000). Telecommunications management network. Disponible en https://www.itu.int/rec/T-REC-M.3400-200002-I

[4] Julian, M. (2018). Practical Monitoring. (ed.) O'Reilly Media, Chapter 2. Monitoring Design Patterns (pp. 27-39).

[5] Timescale. Benchmarking TimescaleDB vs. InfluxDB for Time-Series Data. Disponible en: https://www.outfluxdata.com/assets/Timescale_WhitePaper_Benchm

[6] OIDRef. Reference record for OID 1.3.6.1.2.1.16.1.1.1.5 Disponible en: https://oidref.com/1.3.6.1.2.1.16.1.1.1.5

[7] Internet Engineering Task Force (1988). IAB Recommendations for the Development of Internet Network Management Standards. Disponible en https://tools.ietf.org/html/rfc1052

[8] Graphite (2014). Case Study: Booking.com. Disponible en: http://graphiteapp.org/case-studies/booking.html