Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo Fin de Grado

Análisis y aplicación del protocolo SNMP en el ámbito de la monitorización de recursos.

Autor: Santiago Marcos García

Tutor/es: Julia Clemente Párraga

2021

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado
Análisis y aplicación del protocolo SNMP en el ámbito de la monitorización de recursos

Autor: Santiago Marcos García

Tutor/es: Julia Clemente Párraga

TRIBUNAL:

Presidente: María Dolores Rodríguez Moreno

Vocal 1º: Concepción Batanero Ochaíta

Vocal 2º: Julia Clemente Párraga

FECHA: 25 de Marzo de 2021

Agradecimientos

A mi familia, ellos han sido mi roca y mi ánimo todos estos años.
A Bea, mi compañera de viaje y mi rayo de luz cuando mis fuerzas flaqueaban.
A mis compañeros de Unatec, de los que he aprendido tantísimo. No sólo conocimientos útiles, sino el valor de actitudes como la cooperación y la entrega.
Gracias a todos ellos, y también a aquellos que han sumado para hacer posible mi formación.

Santiago.

Resumen

En un mundo como el actual, con tanta presencia dispositivos electrónicos interconectados entre sí, a nivel profesional es de vital importancia tener la capacidad de supervisar cada uno de ellos, a fin de detectar fallas y anomalías, ya sea en el lado hardware o software. En muchas ocasiones se cuenta con estructuras formadas por decenas o cientos de estos sistemas a monitorizar, y es por eso que en este proyecto se pretende ofrecer una alternativa de visualización de alertas que haga más sencilla la labor de control. Para ello se llevará a cabo el despliegue de una plataforma que aprovecha las capacidades de un protocolo tan versátil como el SNMP. Esta plataforma facilitará al usuario tener un control exhaustivo de todo el entorno así como la inclusión de mejoras que incrementen sus funcionalidades.

Abstract

In a world like this, with so much presence of electronics devices interconnected with each other, at a professional level it is of vital importance being able to supervise each one of them with the purpose of detecting failures or anomalies, either in a hardware or software environment. Many times, those structures of devices consist of tens or hundreds of system to monitor, and that's why this project aims to offer an alternative form displaying alerts that make control tasks easier. Because of that, the intention is to deploy a platform that leverages the capabilities of a protocol as versatile as SNMP, which will allow the user to have an exhaustive control of the whole environment in a simple way, as well as the inclusion of improvements that increase the amount of functionalities.

Palabras Clave

Supervisión, monitorización, *SNMP*, control, plataforma

Índice General

Índice de Tablas

Índice de Figuras.

Parte I

Resumen Extendido.

Actualmente, cualquier sector que haga uso de sistemas informáticos tiene como una responsabilidad prioritaria el mantenimiento y control de una serie de recursos, equipos y servicios, los cuales necesitan de una supervisión constante para desempeñar sus funciones correctamente.

Estos recursos, que tanto facilitan la vida de muchas personas en el mundo a día de hoy, así como el desempeño de su trabajo, son una solución a las necesidades que van surgiendo con el paso del tiempo. Sin embargo, rápidamente pueden convertirse en un problema tanto para los usuarios como para el personal responsable de la administración de esos entornos ante situaciones problemáticas, es más, cualquier infraestructura es susceptible de sufrir un deterioro o un fallo puntual con el paso del tiempo. Sin este tipo de cuidados, el incremento de este tipo de problemas podría convertirse en algo cada vez más frecuente, dado que muchos de los trámites que antes se realizaban a través de métodos tradicionales, que implicaban presencialidad, en la actualidad se llevan a cabo por vía telemática. A ello se suma un aumento de personas con acceso a este tipo de servicios, y por ende, si hay más servicios, son necesarios más recursos, que pueden desencadenar más problemas.

Los fallos ocurridos por no ser conscientes del estado de salud del entorno, no suelen reducirse a la compra de nuevo material o a un restablecimiento de servicio inmediato. A menudo, estas circunstancias pueden acarrear pérdidas económicas ligadas a datos almacenados o a procedimientos de gran valor que se encontraban en curso. La mayoría de las grandes compañías cuentan ya con soluciones a medida, que les permiten tener un control exhaustivo de su infraestructura, y tratan de prevenir o mitigar estos lamentables sucesos. No obstante, no es del todo extraño encontrarse con noticias que cada cierto tiempo se hacen eco de este tipo de situaciones[1].

Es por ello que este proyecto pretende ahondar en esa necesidad de monitorizar aquellos sistemas que forman parte de los procesos informáticos. Para eso, a lo largo de esta memoria se verán recogidos una serie de conocimientos que pueden ser útiles para comprender el modo en el que estas soluciones suelen llevarse a cabo. Como pilar fundamental, el estudio se centrará en el uso del protocolo SNMP, *Simple Network Management Protocol* (F.Kurose & W. Ross, 2010), el cual ofrece una gran flexibilidad para visibilizar conjuntos de sistemas, de naturaleza muy diversa. A pesar de su origen en 1988, se ajusta a las necesidades del tiempo y ha sabido adaptarse gracias a

[1] https://www.computerworld.com/article/3412197/top-software-failures-in-recent-history.html#slide1

las mejoras introducidas en las nuevas versiones, de las que también se hablará a lo largo del presente documento.

Una vez presentados dichos conocimientos, el objetivo final de este trabajo reside en llevar a cabo el despliegue de una plataforma de monitorización que permita unificar la visibilidad de estos dispositivos. Para ello se hará uso de Icinga, un proyecto software de código abierto, basado en Nagios. La intención es incorporar en su configuración una ejecución periódica de solicitudes SNMP que posibilite visibilizar datos referentes a la salud de dispositivos y elementos que se encuentren en esa red. Para este aspecto se hará uso de *plugins* que admitan una integración con la plataforma mencionada anteriormente.

Este software hace uso de la arquitectura cliente-servidor para representar y ofrecer la información recolectada a través de una aplicación web, así como para comunicarse con los nodos de los que se desea obtener información. Dichos datos pueden ser referentes a los componentes hardware así como a algunos de los elementos del software que albergan.

La gran ventaja del protocolo SNMP es que, al ser un estándar, muchos de los fabricantes incorporan en sus procesos de fabricación y desarrollo, implementaciones propias de MIBs (o adaptaciones de las existentes), así como de servicios de escucha a este tipo de peticiones, lo que facilita con creces una inclusión rápida en los entornos de supervisión como el que se plantea en esta memoria. Tras la puesta en marcha, se realizarán pruebas de recogida de datos se hará una propuesta para la clasificación de los mismos en diferentes umbrales. De este modo, se ofrece la posibilidad de incorporar semáforos que simplifiquen la labor de clasificación de la información. Asimismo, se realizarán propuestas de pizarras donde visualizar esos datos en función de las preferencias que se estimen oportunas.

Una vez alcanzado este punto, se presentarán adicionalmente unas propuestas de mejora, entre las que se incluye la incorporación de un demonio que escuche *traps* de notificación, el cual permita recibir alertas en tiempo real en la plataforma de monitorización acerca de incidentes acontecidos en los nodos supervisados. Si bien desde Icinga se puede tener visibilidad de los mismos, de la misma manera se ofrece también el desarrollo de un visor con el que consultar y explotar esa información con mayor nivel de detalle. Adicionalmente, se ofrecerán otras mejoras que otorguen al sistema funcionalidades complementarias, y que permitan cumplir con algunos de los requerimientos que podrían encontrarse en entornos críticos o con necesidades de alta disponibilidad, ya que es habitual que estos entornos formen parte de infraestructuras delicadas cuyo funcionamiento continuado y seguridad son cruciales para el desempeño de sus funciones.

Parte II

Memoria del Trabajo.

Capítulo 1.

Introducción

El presente documento recoge conocimientos teóricos sobre la monitorización y el protocolo SNMP, así como la puesta en marcha y configuración de una plataforma de monitorización que permita monitorizar algunas de las constantes de nodos interconectados en red a través del protocolo SNMP.

A lo largo de este primer capítulo se ofrece un primer enfoque del problema, así como una perspectiva resumida de las fases de las que consta, los objetivos alcanzados durante el desarrollo del proyecto y las motivaciones que han suscitado el desarrollo de este proyecto.

1.1 Contexto

Bajo cualquier tipo de infraestructura de sistemas informáticos reside una serie de componentes (tangibles o no) que son susceptibles de incurrir en cortes de servicio, bien sea por el paso del tiempo, por los fallos circunstanciales o el deterioro por agentes externos (situaciones de emergencia, *malware*, etc.). La probabilidad de sufrir un contratiempo de estas características aumenta sensiblemente conforme se va añadiendo complejidad al sistema (debido a múltiples factores tales como la incorporación de nuevas funcionalidades, incremento en el número de elementos que lo componen, etc.), que va adquiriendo criticidad a medida que evoluciona.

Por lo tanto, suele ser clave la instauración de medidas de prevención para evitar que los fallos tengan un impacto mayor; esto puede lograrse de maneras diversas: añadiendo redundancia, adquiriendo dispositivos con mecanismos de protección ante desastres o sustituyendo aquellas partes que han entrado en una fase de envejecimiento que pueda poner en riesgo la integridad del resto de elementos. En ocasiones la incorporación de estas precauciones supone, ya en sí misma, un incremento de complejidad total por lo que, aunque el entorno esté más protegido a inclemencias de magnitud mayor, no desaparece la posibilidad de que éstas acaben por surgir en alguna de sus partes.

Al margen de los mecanismos de prevención que se puedan integrar para evitar estas situaciones, un factor que suele ser decisivo para mitigar el impacto es reaccionar a tiempo, ya que en ocasiones puede significar un restablecimiento más sencillo y rápido, o incluso prever una situación desagradable. Éstos serían procesos que no evitan la problemática de manera directa, pero que reducen las posibilidades de sufrir un suceso notificando al usuario que interactúa con ellos. Si observamos detenidamente, podemos encontrar este tipo de soluciones en otros ámbitos, por ejemplo: un vehículo puede pasar una revisión previa a un viaje que asegure su estado correcto con el fin de intentar prevenir posibles averías a medio camino. Sin embargo, una vez el viaje está iniciado, y a pesar de las precauciones tomadas, el coche podría, por ejemplo, quedarse sin combustible. Las posibilidades de que un suceso tan desagradable como este suceda, pueden verse reducidas gracias a la existencia de un indicador situado en el salpicadero del automóvil que informe de la capacidad de combustible restante; cuando éste está cerca de agotarse, el indicador emitiría una alerta al conductor para que sea consciente y, si este reacciona en un tiempo razonable, podría evitar una complicación mayor.

Sin duda, la monitorización de recursos se encuentra en este grupo de opciones ya que, en ocasiones, podría actuar como un mecanismo preventivo que muestre con anticipación factores de deterioro previo a un problema, permitiendo así una reacción anticipada. No obstante, en muchas situaciones las plataformas de monitorización simplemente muestran datos de una complicación ya existente a modo de diagnóstico de lo ya sucedido. Es en este tipo de soluciones en el que se apoya el resto de este proyecto.

El término monitorización (sin entrar en un contexto específico) está presente en muchos ámbitos (medicina, sociología, etc.), pero afinando en el área que nos ocupa, se puede definir de la siguiente manera (Colomer, Joaquim, & Jordi, 2000):

"Automatización de un proceso de vigilancia que dota al operario de los mecanismos necesarios para ser alertado del buen, o mal funcionamiento de uno o varios sistemas, así como una interacción amigable con el proceso y el registro de su evolución. Su propósito es facilitar la detección de situaciones anómalas y su diagnóstico a través de un seguimiento continuo de las variables que componen el proceso."

Este término tiene una diferenciación con el término "supervisar", ya que éste último supone que, además de tener conocimiento acerca del funcionamiento de la infraestructura, el personal responsable debe analizar estas desviaciones, deduciendo los motivos por los cuales se produce dicha anomalía, y resolver esa situación conflictiva. Además de eso, la supervisión conlleva la toma de medidas correctivas para evitar que esa casuística vuelva a suceder. Sin embargo, y con

el propósito de evitar una continua repetición del término “monitorización”, se utilizarán ambos conceptos de forma indistinta, estableciendo así una aproximación semántica entre ambos y contextualizado su significado en los sistemas de monitorización informáticos.

Muchos de las plataformas que permiten llevar a cabo estas labores de control, incorporan interfaces que hacen más sencillo dicho diagnóstico gracias al uso de gráficos, alarmas visuales/sonoras, u otras representaciones. No obstante, hay algo que supuso un cambio en la forma en la que los componentes de un entorno son supervisados es la llegada de los sistemas de comunicaciones. Antes de esto, dicha supervisión pasaba por tareas rutinarias que se realizaban de manera manual y periódica de cada elemento, a modo de inspección técnica. Si bien los primeros sistemas no tenían una complejidad comparable a muchos de los presentes en nuestro tiempo, aquellas tareas eran de vital importancia para los responsables de aquellos ámbitos. Con el tiempo muchos de esos sistemas incorporaron medidas que permitían al administrador conocer las constantes de los equipos locales que se estaban usando (por ejemplo, el equivalente al *Task Manager*[2] de Windows hoy día, o *top*[3] en Unix). Éstas son herramientas simples que permiten conocer el estado del nodo del que se está haciendo uso. Sin embargo, en otro tipo de dispositivos con menores capacidades, consultar estos datos no es tan sencillo ya que pueden incluso no disponer de una interfaz a través de la cual obtener datos de funcionamiento (por ejemplo una impresora).

La llegada de medios analógicos, como por ejemplo, los anillos RS232 (Notes, 2020), su primera especificación data de 1962, trajeron consigo grandes mejoras en términos de supervisión de componentes algo más industriales pero que son el fundamento de mucho de lo que se puede encontrar a día de hoy. De hecho, en muchos entornos del ámbito industrial es relativamente frecuente encontrarse todavía con este tipo de implementaciones.

Los datos recogidos a través de ese tipo de vías son representables de muchas maneras, pero hay un protagonista que aún a día de hoy acompaña muchos de los procesos de monitorización. Estamos hablando de los sistemas SCADA, *Supervisory Control And Data Acquisition* (IEEE, 2017). Éstos emergieron en la década de los 70 permitiendo controlar automatismos (admitían también interactuar con algunos de los sistemas que formaban esas redes), así como aportar una valiosa recolección de datos centralizada desde un MTU, *Master Terminal Unit* (National Institute of Standard and Technology, 2015).

[2] https://en.wikipedia.org/wiki/Task_Manager_(Windows)

[3] https://geekytheory.com/funcionamiento-del-comando-top-en-linux

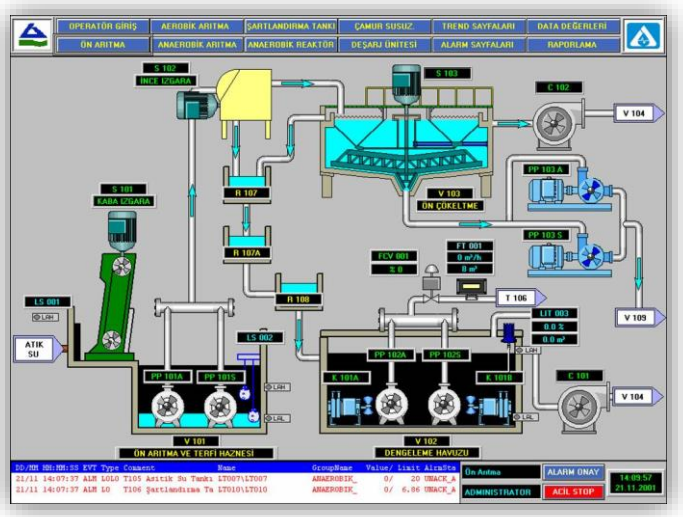

Figura 1, Ejemplo de la interfaz de un entorno SCADA a través de un MTU.

Mientras estas soluciones iban llegando a muchos sectores relacionados con la producción, la era de las comunicaciones se abría paso, dando lugar al conocido modelo de capas de red y el Modelo OSI. El origen de la primera comunicación a través de internet data de 1969, cuando a través de algunas universidades de Estados Unidos se realizó satisfactoriamente la primera conexión, aunque ya se había estado investigando con este tipo de estructuras durante toda la década.

Uno de los protocolos más famosos perteneciente a esta época, y que permitió tener visibilidad de dispositivos conectados a una red fue ICMP (*Internet Control Message Protocol;* sobre él se apoya el *ping*). Éste es un ejemplo más de un sistema que permite el envío de mensajes de error e información operativa muy básica. Aún a día de hoy sigue siendo utilizado para diagnosticar problemas de conectividad.

Ya en la década de los 80 se alcanza una cierta madurez en términos de comunicaciones, y con ello se hacen grandes avances en muchos de los campos. Sin duda, un hito fue, con la llegada del protocolo sobre el que se apoya gran parte del trabajo de este proyecto, el protocolo SNMP (*Simple Network Management Protocol*), del que se habla con mayor profundidad a lo largo del Capítulo 3.

Las tecnologías previas, entre otras, son de gran utilidad, dado que gran parte de la complejidad de algunos entornos reside en que estas infraestructuras pueden implicar factores a considerar, como: distintas ubicaciones, accesos, permisos, diversos fabricantes, etc., lo que se traduce en una unificación complicada. Hoy en día, sin ellas, llevar a cabo manualmente las labores de supervisión de sistemas sería algo completamente impensable.

Como se mencionaba anteriormente, la mayor parte de las herramientas tienen una interfaz sencilla que, de manera intuitiva, hacen posible una supervisión eficaz por parte del personal destinado a estas tareas. Muchas de ellas pueden ser administradas desde aplicaciones Web accesibles desde varios equipos, en lugar de tener un puesto único de monitorización (como es el caso de SCADA). El objetivo de este trabajo es utilizar la plataforma Icinga (Icinga Development Team, 2020) como una muestra del potencial de este tipo de entornos que permiten integrar soluciones variadas como es el caso de SNMP, al igual que otras tecnologías de control de sistemas. De este modo, se puede conseguir un entorno completamente funcional de monitorización.

1.2 Motivación

Como ya se ha indicado en el apartado anterior, cada vez es más frecuente encontrar servicios que permiten realizar funciones de manera telemática. Es por esto que, con el paso del tiempo, se ha considerado de especial interés el aprendizaje de técnicas que permitan conocer el estado de esos dispositivos en tiempo real.

En este sentido, una de las problemáticas que es necesario abordar es la amplia dispersión de fabricantes, modelos, o capacidades en los dispositivos desplegados, por lo que encontrar un factor unificador para todos ellos supone una gran mejora en términos de calidad de supervisión y de ahorro. Uno de estos factores es el uso de *agentes SNMP* que, mediante el empleo de este protocolo permiten comprobar ciertos datos referentes a su estado de salud.

Con este trabajo se pretende desarrollar capacidades que permitan una mejor comprensión del protocolo, así como su implantación y la de otros métodos en una herramienta que haga sencillo el desempeño de los cuidados de una infraestructura. Para ello, se hará uso de una plataforma derivada de *Nagios* (Icinga Development Team, 2009). Ésta posee gran presencia en el ámbito profesional así como una comunidad que la dota de una gran capacidad de adaptación al paso del tiempo, la cual facilitará la incorporación de nuevas funcionalidades que cubrirán las necesidades que vayan surgiendo a medida que avanza la tecnología. Es de uso tan extendido que se ha considerado de gran interés la comprensión de su funcionamiento. Dicha plataforma aprovecha la arquitectura *Cliente-Servidor* para recoger información de todos los nodos, así como para representarla y consultarla a través de una aplicación Web. Este tipo de herramientas ofrecen una solución sencilla y con un coste reducido a empresas que requieren de este tipo de flexibilidad, o

que no pueden permitirse la integración de herramientas propietarias diseñadas a medida. Para el desarrollo de este trabajo se han tenido en cuenta los conocimientos adquiridos profesionalmente durante el desempeño de funciones como personal *IT/Sistemas*.

1.3 Estructura del proyecto

Con el fin de facilitar la lectura de la memoria de este Trabajo Fin de Grado, se ofrece una breve sinopsis de los capítulos en los que está estructurado:

- **Capítulo 1: Contexto y Motivación.**

  Este capítulo presenta de forma resumida una visión general del proyecto así como un contexto del marco de trabajo y una serie de factores que han motivado la elaboración de éste. Adicionalmente, el primer capítulo recoge esta misma estructura.

- **Capítulo 2: Planteamiento del Problema y Supuestos de Trabajo.**

  El segundo capítulo presenta una serie de objetivos a cumplir durante la integración y el desarrollo de conocimientos acerca de las plataformas de monitorización, además de una sucesión de requisitos. Se encuentran aquí también las tecnologías sobre las que se apoya la plataforma.

- **Capítulo 3: Estado del arte.**

  En este capítulo se ofrece el conocimiento teórico acerca de las plataformas de monitorización, así como una vista en profundidad del funcionamiento del protocolo SNMP. Adicionalmente se recogen las principales características de Icinga y de sus medios de aplicación en la actualidad.

- **Capítulo 4: Implantación de la monitorización vía SNMP en Icinga.**

  A lo largo de este cuarto capítulo se describe paso a paso el proceso de despliegue y configuración de la herramienta de monitorización mencionada, así como la integración con *plugins* que permitan el uso de SNMP y de otros métodos de monitorización. Esta sección también recoge una guía de aplicación de plantillas y parámetros que ajustan las funcionalidades de la plataforma.

Como último apartado, se ofrece la creación de umbrales para representar con semáforos los datos recolectados, además de la implementación de filtros y pizarras que representen la información de forma concisa.

- **Capítulo 5: Implantación de mejoras al alcance.**

  A lo largo del quinto capítulo se lleva a cabo la implementación de mejoras que dotan a Icinga de un valor añadido, mediante la integración de funcionalidades que pueden resultar interesantes en entornos como el sugerido en el apartado anterior. En esta sección podemos encontrar la incorporación de un demonio de *traps*, un visor que facilite la explotación y consulta de los mismos, el uso de SSL para la aplicación Web, además de otras extensiones, como la creación de *plugins* personalizados y la aplicación de alguna técnica para incrementar la alta disponibilidad.

- **Capítulo 6: Conclusiones.**

  En este capítulo se expone un análisis de lo que ha supuesto este trabajo, además de las fortalezas y debilidades detectadas en este tipo de soluciones, así como dificultades encontradas y un breve juicio de opinión del empleo de estas herramientas.

- **Capítulo 7: Líneas de trabajo futuro.**

  Por último, en este séptimo capítulo, se detallan algunas de las posibles vías de evolución del entorno.

Capítulo 2.

Planteamiento del Problema y Supuestos de Trabajo.

Una vez contextualizado el trabajo, su alcance en el mundo empresarial, y conocido el ámbito general de este trabajo, se procede a detallar los objetivos a cumplir durante su desarrollo, los requisitos que se deben alcanzar, así como algunas restricciones.

2.1 Objetivos del proyecto

En base a lo expuesto anteriormente en el apartado 1.2, en el que se detallaban las motivaciones, y conociendo situaciones reales del personal destinado al mantenimiento de infraestructuras informáticas con los que se han intercambiado experiencias e inquietudes, se ha elaborado una serie de objetivos que este proyecto pretende cubrir:

- **Objetivo 1:** Alcanzar y exponer conocimientos en materia de monitorización, así como ofrecer un análisis teórico de las características y propiedades que ofrece un protocolo como SNMP.

- **Objetivo 2:** Despliegue de una herramienta de monitorización que permita de manera sencilla supervisar el estado de diferentes dispositivos, caracterizada por ser escalable y con facilidad de mantenimiento. La herramienta debe permitir también la consulta de datos históricos previamente recogidos.

- **Objetivo 3:** Integrar los conocimientos adquiridos durante el primero de los objetivos, de tal forma que se puedan adherir a la plataforma ya desplegada, así como recabar información vía *SNMP* de los nodos extendidos a lo largo de una red de comunicaciones.

- **Objetivo 4:** Utilizar *plugins* y estrategias versátiles que faciliten la ampliación en términos de operatividad de la plataforma.

- **Objetivo 5:** Utilizar módulos de código abierto durante el proceso.

- **Objetivo 6:** Utilizar paquetes que garanticen un mantenimiento futuro, como *Net-SNMP*, que permitan evitar la obsolescencia debida al paso del tiempo.

- **Objetivo 7:** Realización de pruebas y recabado de datos, así como su clasificación mediante umbrales, filtros y pizarras que faciliten su explotación.

- **Objetivo 8:** Ofrecer una guía didáctica del uso de esta plataforma (que será transversal a lo largo del desarrollo de toda la memoria) con el fin de facilitar la curva de aprendizaje de este tipo de entornos.

- **Objetivo 9:** Integrar mejoras adicionales que doten a la aplicación web de un valor añadido en función de las necesidades y carencias analizadas previamente con personal destinado a estas labores, los cuales permanecen constantemente en contacto con este tipo de entornos.

- **Objetivo 10:** Ofrecer una visión crítica de las virtudes y carencias encontradas durante el proceso.

2.2 Requisitos del proyecto

Una de las grandes ventajas de usar plataformas como *Icinga* es que los requisitos referentes al usuario final son realmente bajos, ya que bastaría con un navegador web para poder interactuar y tener visibilidad de la plataforma completa, así como de todos los datos referentes a los nodos supervisados, evitando así el uso de clientes pesados instalados en los puestos de trabajo. De hecho, otra de las ventajas es que puede ser consultada de manera simultánea por tantos usuarios como se desee.

Por supuesto, lo anterior conlleva que el peso de la aplicación resida en un servidor al que el resto de los usuarios tengan acceso (ya sea vía *LAN* o vía Internet), así como que este mismo equipo tenga comunicación con los equipos a supervisar. Es habitual encontrar disposiciones, en términos de arquitectura de la red, con implementaciones de redes *DMZ* (*Demilitarized Zone*, o zona desmilitarizada), o bien en las que la plataforma de monitorización tenga acceso a las dos redes, siendo ésta la única con visibilidad directa de los sistemas por supervisar.

Respecto a la composición interna de la herramienta, ésta se basa en una serie de ficheros de configuración que pueden ser ampliados a placer, por lo que su escalabilidad en términos de ampliación la hace sumamente flexible. Icinga cuenta además con una REST API que podría facilitar incrementos funcionales mediante desarrollos futuros.

Es más, aunque en ocasiones su funcionalidad básica (considerando la configuración predeterminada una vez se realiza la instalación), pueda quedarse corta dependiendo del tipo de necesidades, la incorporación de nuevos *plugins* y/o módulos de expansión, es una práctica común entre las compañías que adoptan estas soluciones. Es el caso de la inclusión de SNMP, un mecanismo al que se podrían añadir modificaciones para complementar su funcionalidad, tal como se podrá apreciar más adelante. Adicionalmente, y como última ventaja, el hecho de emplear plataformas o*pen source*, permite que cualquiera con el conocimiento suficiente, modifique su estructura a placer.

Sistemas como *Icinga* cuentan con autenticación de usuarios, pudiendo crear perfiles y roles que permitan al personal operar con determinados sistemas en función de su perfil de administración. Pueden darse de alta tantos usuarios como se necesite y todo ello es consultable a través del *backend* escogido, incluyendo los datos recogidos.

2.3 Marco de trabajo

Antes de comenzar, conviene limitar el área de trabajo sobre el que se desarrolla este proyecto y su alcance:

- Todos los datos mostrados a lo largo de esta memoria son datos reales obtenidos durante el despliegue y configuración de la herramienta, así como de sus complementos.

- Para la interacción con Icinga y su acceso se trabajará con navegadores web integrados en equipos de escritorio, así como con dispositivos móviles, a fin de verificar que la funcionalidad no varíe dependiendo del dispositivo, ya que cada vez es más habitual encontrar el uso de *smartphones* y *tablets* para la realización de estas tareas.

- En la fase de experimentación SNMP se utilizarán tantos dispositivos reales (teléfonos móviles, otros servidores desplegados en el hipervisor, etc.) que puedan actuar a modo de *agentes*.

- La captura de requisitos y las soluciones adoptadas para realizar toda la configuración son una simulación, pero en su mayoría son un compendio de las necesidades reales encontradas en situaciones similares durante el desarrollo de la actividad profesional.

2.4 Tecnología empleada.

A continuación se ofrece un breve resumen de las tecnologías sobre las que se apoya este proyecto. En este apartado no se incluye el propio *Icinga*, ya que será tratado con mayor detalle en el capítulo 3.

2.4.1 Lenguajes empleados

La puesta en marcha de este proyecto, a pesar de no necesitar de un desarrollo concreto, conlleva la creación y modificación de *shell scripts*, con el fin, entre otros, de implementar su configuración interna. Sobre todo es sumamente recomendable estar familiarizado con la sintaxis de los ficheros de configuración de *Nagios* e *Icinga*, así como con el de las estructuras de las que hace uso SNMP para configurar el servicio (como es el caso del lenguaje empleado en las MIB). Adicionalmente, se incluyen elementos que han sido desarrollados a través de *Powershell* en el apartado de mejoras al alcance (Capítulo 5).

2.4.2 Microsoft Hyper-V

Hyper-V (Microsoft, 2018) es un hipervisor, o sistema de virtualización, propiedad de Microsoft permite la ejecución de máquinas virtuales. Su lanzamiento se produjo como una *beta* en la versión Windows Server 2008, durante ese mismo año. Dentro de la documentación de Windows Server está clasificado como un rol adicional (Microsoft, 2016), mientras que en los equipos de escritorio como Windows 10 está catalogado como una característica que puede ser instalada en las versiones Pro y Enterprise (únicamente en distribuciones de 64 bits).

Otra de las prácticas habituales en este tipo de entornos es la virtualización del servidor de monitorización, dado que dota a la infraestructura de mayor flexibilidad en caso de ser necesaria, por ejemplo, una ampliación de recursos o una migración, así como de facilidad de

experimentación, al permitir al administrador poder disponer de puntos de control[4] a los que volver en caso de ser necesario. Además, facilita mucho el sistema de copias de seguridad. Por lo tanto, se ha decidido mantener este tipo de estructura, a fin de hacer de la plataforma algo portable e integrable de forma rápida y sencilla en nuevos entornos.

Durante la fase de experimentación con Icinga se hace uso de adaptadores de red virtuales conectados a un *Virtual Switch* (que es un conmutador de red vía *software* que sirve la red a este entorno virtual, así como a otros entornos adicionales que también podrían conectarse a él) de tipo *bridge*, también conocido como externo, con la intención de facilitar la visibilidad de nodos que se encuentren en la red local del lugar en el que se realizan las pruebas[5]. Asimismo, se utilizan discos de expansión dinámica que, aunque tienen un rendimiento inferior, permiten un crecimiento gradual a medida que se hace uso de disco.

A nivel de hardware virtual no se necesitan de grandes especificaciones, salvo que la estructura de nodos y variables sea demasiado grande (> 1000 aprox.). Al ser este un entorno de pruebas, se ha dotado a la máquina virtual del siguiente hardware virtual:

- Máquina virtual con hardware virtual de versión 1, en su versión de configuración 9.0.
- Internamente se ha utilizado Ubuntu Server en su versión 18.04 (LTS)
- Disco: expansión dinámica (en formato .vhdx) con 127GB de tamaño máximo (ampliables). Internamente el almacenamiento de Ubuntu está configurado con LVM, lo que facilitaría su ampliación en caso de ser necesario.
- 4 vCPU.
- 4096 MB de RAM.
- Adaptador de red virtual externo, como se comentaba anteriormente. Para las pruebas se ha considerado una configuración interna por DHCP, aunque puede ser variado mediante la configuración de *netplan* desde Ubuntu.

2.4.3 Apache2

Como se ha dicho anteriormente, Icinga funciona como una aplicación web, por lo que hace falta un servidor HTTP. La parte web de Icinga es un *framework* de PHP de aspecto minimalista, y que es complementario al motor de la aplicación, conocido internamente como Icinga Web 2.

[4] https://docs.microsoft.com/es-es/virtualization/hyper-v-on-windows/user-guide/checkpoints
[5] https://docs.microsoft.com/es-es/windows-server/virtualization/hyper-v-virtual-switch/hyper-v-virtual-switch

Existen varias opciones de despliegue entre las que se incluyen Nginx y Apache, habiendo utilizado en el proyecto esta última opción. Para la instalación del servidor HTTP se hará uso de los repositorios de paquetes habituales.

Apache (Apache Documentation, 2020), como servidor HTTP, vio la luz en 1995 bajo el desarrollo de Apache Software Fundation, y fundamentalmente está escrito en el lenguaje C. Es uno de los proyectos de código abierto más populares gracias a su rápida expansión motivada por el auge de las tecnologías Web.

2.4.4 MySQL

Tanto para el almacenaje de los datos obtenidos durante el proceso de supervisión, como de los datos de los usuarios con acceso a la aplicación, o metadatos necesarios para el funcionamiento de la plataforma, se necesita de una base de datos de carácter relacional donde depositar los datos en reposo. Según la documentación de Icinga se admiten IcingaDB (un *backend* aún en desarrollo propio del proyecto Icinga), PostgreSQL o MySQL/MariaDB.

En este proyecto se ha decidido usar MySQL (MySQL, 2021) por una cuestión de familiaridad con este motor de base de datos, además de que su uso es una práctica bastante extendida en el ámbito profesional. Si se compara con PostgreSQL, en términos de rendimiento, MySQL tiene la ventaja de que no necesita hacer un *fork* por cada conexión cliente a base de datos (pues en el caso de PostgreSQL se necesita de una reserva 10MB de memoria por cada una de esas conexiones). Esto hace que puedan usarse más recursos para otras tareas que pueden ser necesarias cuando hay una carga elevada de trabajo, por ejemplo, durante la monitorización. A pesar de que PostgreSQL pueda incorporar algunas ventajas en funcionalidad, no son determinantes en absoluto para el desempeño de Icinga.

La primera versión de MySQL[6] fue liberada en 1995, su código está escrito en C/C++, y aunque inicialmente fue desarrollado por MySQL AB, ésta fue adquirida por Sun Microsystems, que a su vez fue adquirida por Oracle tiempo después. MySQL está considerado como una de las bases de datos de código abierto más populares.

[6] https://es.wikipedia.org/wiki/MySQL

2.4.5 Monitoring Plugins y NET-SNMP

Tal y como se ha mencionado anteriormente, el propio motor de Icinga hace uso de ficheros binarios y *scripts* con los que realizar las consultas a los nodos, ya que por sí solo no puede comprobar servicios externos. El proyecto Monitoring Plugins (originalmente conocido como Nagios Plugins)[7] incluye un conjunto de más de 50 *plugins* completamente estándar, compatibles con servicios como Icinga, Nemon, Nagios, Shinken o Sensu entre otros. Cada uno de estos *plugins* es una herramienta en sí misma que permiten realizar consultas por línea de comandos sobre algunos servicios (HTTP, uso de CPU, espacio en disco, salud de base de datos, etc.), entre los que se encuentra uno que nos será de gran utilidad durante este proyecto: *check_snmp*.

Este *plugin* en concreto hace uso internamente del comando *snmpget* incluido dentro del paquete NET-SNMP (NET-SNMP, 2020). Por su parte, Net-SMMP es una suite de herramientas software que se usa para configuraciones con este protocolo. Soporta las versiones v1, v2c, v3 así como AgentX. Está desarrollado en C, Perl y Python, y comenzó su desarrollo en 1992. Desde entonces, se ha convertido en uno de los paquetes más populares. Dentro de esta herramienta podemos encontrar aplicaciones (las cuales serán vistas más en profundidad en el Capítulo 3.) como *snmpget, snmpwalk, snmptrapd*, entre otras que serán vistas con mayor profundidad, durante la implantación de mejoras del Capítulo 5.

[7] https://www.monitoring-plugins.org/

Capítulo 3.

Estado del arte

En este tercer capítulo se ofrece una vista teórica más minuciosa, así como un acercamiento a los avances hasta la fecha de realización de este proyecto, acerca de los tres pilares sobre los que se apoya este Trabajo Fin de Grado:

- El término *Monitorización*, del que ya se ha ofrecido una primera aproximación, si bien en este capítulo se ofrece una vista más detallada que clasifica sus variantes, y que centra especialmente la atención en un enfoque en red, que es la variante de la que se hace uso en este proyecto.

- El *protocolo SNMP*, una definición más amplia, entrando en detalle en su funcionamiento, principales componentes, versiones y características.

- La herramienta *Icinga*. En este apartado se encuentra una síntesis teórica de su origen, campo de trabajo así como de sus funcionalidades y presencia en el ámbito profesional.

3.1 La monitorización como herramienta.

Si hay una palabra que se acerca lo suficiente a la monitorización (como término genérico), es vigilancia, siendo ésta una muestra inequívoca de observación y atención. De hecho, si partimos de un punto de vista semántico, la Real Academia de la Lengua Española, define la monitorización como:

"Observar mediante aparatos especiales el curso de uno o varios parámetros fisiológicos o de otra naturaleza para detectar posibles anomalías."

Adaptando esta definición al ámbito de la informática, podemos decir que la monitorización de recursos, en un afán por conocer el estado de salud de los objetos al cuidado, supone un

acercamiento *visual* a los componentes físicos y a los marcadores software de uno, o varios equipos que ofrecen un servicio, con el fin de detectar anomalías en su funcionamiento (Digital Ocean, 2017).

Centrando esta definición sobre el campo que nos ocupa, que es la monitorización de recursos en red, esto se traduce en un control de los diferentes sistemas de una infraestructura con el fin de detectar anomalías. Cabe destacar que todas las distinciones que se van a tratar a continuación varían en función de las necesidades encontradas por los responsables de estos sistemas, así como por el tipo de infraestructura. En muchas ocasiones es frecuente encontrar mayor detalle y granularidad en entornos de producción, a diferencia de aquellos destinados a las pruebas o el *testing*.

En primera instancia, se podría realizar una clasificación en función de la naturaleza del objeto de control:

- **Monitorización del hardware:** normalmente va más dirigida a detectar de manera eficaz posibles carencias materiales en los componentes físicos de los sistemas, así como a encontrar el origen en incidentes de mayor magnitud. Un buen ejemplo de esto podría ser el control del estado en el que se encuentran las fuentes de alimentación de los servidores. Muchos de ellos cuentan con redundancia, por lo que detectando un problema en una de las fuentes, podemos anticiparnos a un problema de alimentación cuando la fuente de alimentación redundante falle. Lo mismo sucede con el estado en el que pueda encontrarse un grupo de discos dentro de una configuración RAID, el estado de la refrigeración, etc.

  Mantener un control minucioso de cada elemento de una infraestructura, y hacerlo de forma eficiente, permite adelantarse a los problemas haciendo una detección temprana, y ahorrando coste económico en el material destinado a reparaciones -pues permite sustitución de componentes antes de lamentar males de mayores-, coste de mano de obra y de otros tipos asociados a la pérdida de producción durante un suceso que podría ser de mayor magnitud (Castaños, 2017).

  Muchos de los fabricantes de hardware incorporan sistemas adjuntos al propio equipo que permiten interactuar con el servidor a través de una interfaz de red separada del resto del servidor. Es el caso de las *iDrac* de *Dell* (Dell Inc., 2019) o de las *iLO* de *HP* (Hewlett Packard Enterprise, 2020). Desde estas interfaces se pueden obtener datos de prácticamente la totalidad del servidor, así como interactuar con la propia máquina,

pudiendo encender un servidor que se encuentra en estado apagado, o incluso formatear un equipo con medios de arranque virtuales que se encuentren en otro punto de la red (algunas de estas funciones van sujetas al licenciamiento contratado de este tipo de interfaces).

- **Monitorización de rendimiento hardware:** tiene como cometido principal ofrecer al responsable de la infraestructura datos referentes al aprovechamiento de recursos de los equipos que componen un entorno. Esto normalmente se traduce en marcadores de rendimiento de los equipos (consumo de CPU, consumo de RAM, espacio en disco disponible, etc.). Su fin principal es detectar cuellos de botella y obtener datos relacionados con el desempeño del servicio ofrecido. De esta manera, se pueden anticipar colapsos y planificar nuevos despliegues con mayor precisión en materia de disponibilidad.

Normalmente la implantación de este tipo de monitorización conlleva la ejecución de ciertos agentes que funcionan sobre el sistema operativo o el *firmware* del dispositivo, y que transfieren esas constantes a un servidor central. Un ejemplo de agentes de este tipo podría ser *Munin*[8] , también de código abierto y escrito en Perl. Éste permite visualizar estadísticas de uso en gráficos bastante intuitivos:

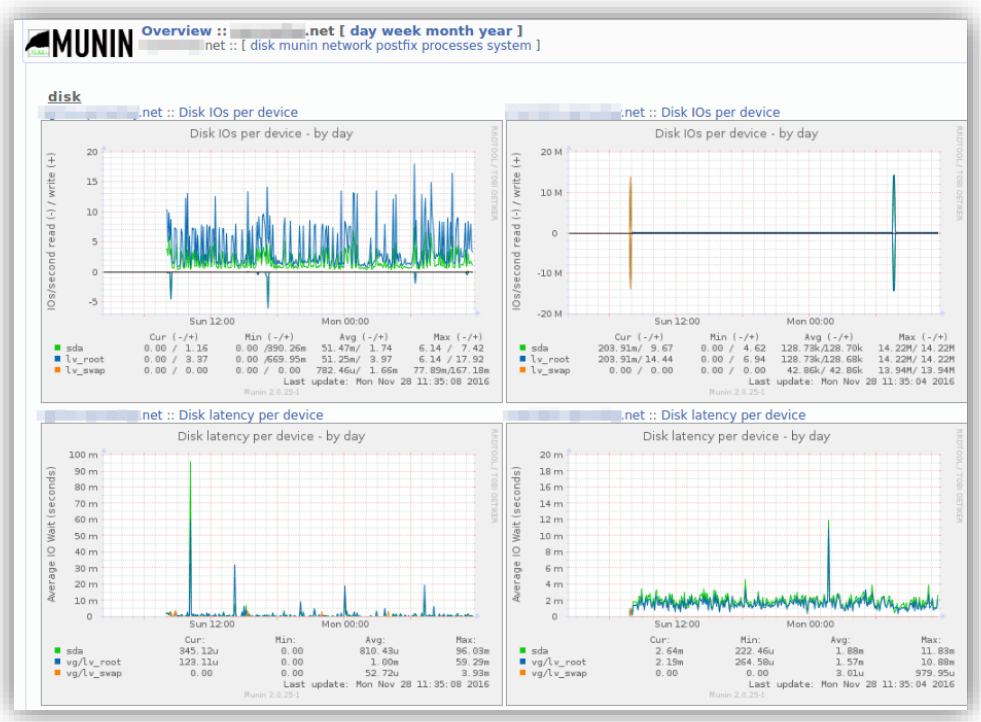

Figura 2. Interfaz gráfica de Munin.

[8] http://munin-monitoring.org/

Otro de los ejemplos más representativos y extendidos, que además permite una integración con Icinga, es el uso de servicios que ejecuten periódicamente scripts de comprobación del estado local del equipo y envíen dichos resultados a través de NSCA, *Nagios Service Check Acceptor* (Nagios, 2016).

En ocasiones donde la infraestructura es lo suficientemente grande, estas métricas en lugar de ser individuales (de un solo equipo), pueden ir centralizadas junto con otros equipos, dando una visión de la disponibilidad de un conjunto de máquinas, a esto se le conoce como disponibilidad del *pool*. Esto es frecuente encontrarlo en *clusters* de todo tipo (bases de datos, *pools* de gestión de contenedores, etc.).

- **Monitorización del software:** normalmente está enfocada a detectar la salud de las aplicaciones que dan servicio y de aquellas partes que las componen. Un ejemplo de ello es la monitorización de un portal web expuesto en un servidor HTTP, el rendimiento de una base de datos o la disponibilidad de una API. En este estrato se pretende algo similar a lo citado en el tipo anterior pero enfocado al servicio ofrecido como tal a través de un componente. Esto nuevamente ofrece la posibilidad de detectar carencias de rendimiento con el fin de, a través de esas métricas, los equipos de desarrollo puedan optimizar los recursos durante la programación. Una puesta en marcha correcta de este tipo de monitorización, normalmente se traduce en un servicio más eficiente, una reacción rápida ante imprevistos, así como una mayor satisfacción por parte de los usuarios que hacen usos de las aplicaciones.

  En el caso de servicios publicados a internet como, por ejemplo, servicios web, mensajería, etc., podemos encontrar otra nueva subdivisión en cuanto a la monitorización, y es la siguiente (Wikipedia, 2020):

  - *Interna:* en ella, el control se realiza localmente o dentro de la misma red donde está expuesto el servicio. Esto es bueno para conocer el estado de la aplicación como tal únicamente (y el resto de componentes software).

  - *Externa:* en este caso, la vigilancia se establece desde un punto externo y totalmente ajeno al entorno, a modo de observador independiente. Éste consulta parámetros que permiten saber la calidad del servicio. Por supuesto, dicha calidad es mucho más completa, ya que permite tener un punto de vista más objetivo al desempeñar la función de un "usuario externo" y, además, posibilita probar todas

las piezas que entran en juego (*firewall*, ancho de banda, caídas del proveedor de servicios, etc.).

Los ejemplos expuestos anteriormente son sólo un fragmento de aquello que puede tener interés en situaciones reales. No hay que olvidar que estas clasificaciones pueden ser encontradas en diferentes ámbitos, como la supervisión de la red y sus componentes, de sistemas y servidores, o en materia de seguridad.

Adicionalmente se puede realizar una clasificación de la monitorización en función de la forma en la que se obtiene la información. En esta clasificación entran en juego la disposición del entorno, así como del tipo de equipos que se encuentren en él (pues no todos ofrecen las mismas capacidades). Por lo tanto hay que diferenciar entre estos dos tipos de perspectiva (Ferri, 2019):

- **Monitorización activa:** esta primera variante establece que los dispositivos o nodos por supervisar se mantienen a la espera hasta ser interrogados por un elemento externo. Una vez el sistema recibe la petición, comprueba los datos y envía de vuelta la información solicitada. Pueden incluso establecerse permisos sobre quién puede interrogar a determinados dispositivos, o qué información puede recopilar. Se los conoce como sistemas de monitorización activos, debido a que ello conlleva intentos accionados a través de una programación generalmente periódica (manual, si es bajo demanda). A esta técnica de interpelar a los nodos también se la conoce como *polling.*

- **Monitorización pasiva:** por el contrario, en esta otra variante, el que se mantiene a la espera es la entidad que monitoriza los nodos. De este modo, cuando el dispositivo remoto envía un dato, es el servidor de monitorización el encargado de recoger la medida a través de uno o varios servicios de escucha y de representarla en la interfaz según esté establecido. Esta información puede ser transmitida por dos motivos:

    - En base a una programación periódica del equipo remoto para enviar el dato (por ejemplo, una planificación horaria).

    - El dispositivo tiene información relevante que transmitir, la cual viene motivada por un evento circunstancial sucedido en el lado del nodo (por ejemplo, un dispositivo conectado a la placa de un servidor ha sufrido un error inesperado de disco).

En la siguiente imagen podemos encontrar una descripción gráfica de esta clasificación:

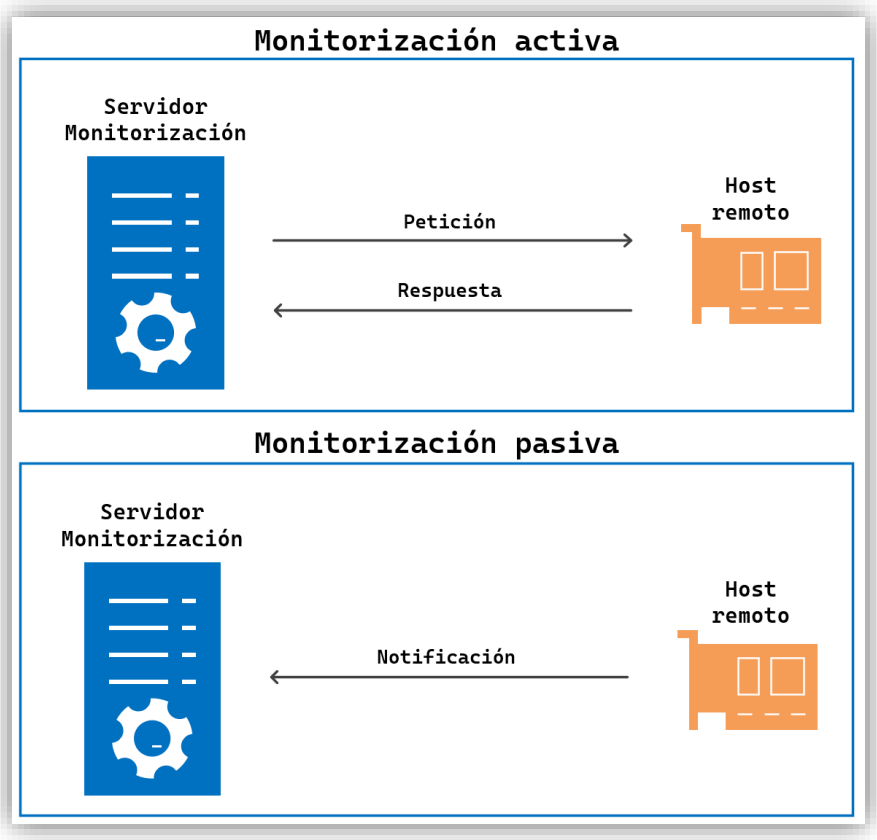

Figura 3. Representación gráfica de la monitorización activa y pasiva. Elaboración propia.

Afortunadamente, y a pesar de que pueda parecer lo contrario, las opciones presentadas en la Figura 3, no son excluyentes (Hein, 2019), sino complementarias, pudiéndose configurar en muchos casos, que determinadas comunicaciones (sobre todo aquellas urgentes) no se hagan esperar al siguiente ciclo de interrogación y sean transmitidas de forma inmediata (pasivamente) y, sin embargo, aquellas que sean menos prioritarias, mantenerlas bajo *polling*.

No hay una ventaja clara de una sobre la otra, salvo el coste en términos de rendimiento de envío de información, es decir, si tenemos un número de nodos muy elevado, el hecho de realizar *polling* sobre todos ellos puede tener cierto impacto en el servidor de monitorización. Lo mismo puede pasar si teniendo una monitorización puramente pasiva se sobrecarga el nodo con una programación demasiado continuada, o con un número de variables demasiado elevado que enviar. Por lo tanto, se deben tener en cuenta las capacidades de ambas partes durante la fase de diseño de la plataforma de supervisión, aunque como ya se ha especificado, por cuestiones asociadas al incremento en funcionalidad, lo más extendido suele ser un modelo híbrido (siempre que sea posible).

Independientemente de qué técnica se aplique, y en qué variante de las que se han visto anteriormente, es recomendable establecer unos criterios de interpretación de la información obtenida por los canales anteriormente citados. Esto quiere decir que es conveniente clasificar los valores obtenidos dentro de unos umbrales, los cuales permitan conocer al supervisor cuán acuciante o prioritaria puede ser una intervención en el entorno cuando se produzca una situación de emergencia. Generalmente, estas clasificaciones es frecuente encontrarlas en forma de semáforos, gráficos, o bien mediante clasificaciones de nivel (baja, media o alta), pudiendo establecerse tantos niveles de alarma como se crea oportuno (parcialmente alta, crítica, etc.).

Por último, y no menos importante, es la configuración de alertas. La mayoría de los sistemas de monitorización incorporan módulos de notificación que pueden ser configurados para enviar determinados eventos por canales como el email o SMS, entre otros. Suele ser de gran utilidad cuando no hay personal para cubrir el cuidado de la infraestructura durante la totalidad del tiempo, o bien cuando se desea conocer cierta información con la mayor celeridad posible. Normalmente estas alertas pueden ir configuradas en función de la responsabilidad o la jerarquía que ocupa el personal dentro del equipo de supervisores.

3.2 El protocolo SNMP

3.2.1 Origen y nociones básicas

El protocolo SNMP, *Simple Network Management Protocol* (R. Mauro & J. Schmidt, 2005) es un protocolo que permite la administración de dispositivos en redes IP. Dada su compatibilidad con multitud de dispositivos y aplicaciones (ya que no sólo permite recopilaciones de datos acerca de hardware), una correcta implementación permitiría adquirir una visión general acerca del estado de una infraestructura en pocos instantes; de hecho, en algunas situaciones, este tipo de sistemas admitiría incluso la toma de acciones automatizadas que eviten problemas de índole superior. Lo que hace interesante SNMP, es que está simplemente compuesto de un juego de operaciones que permiten la recopilación de información de agentes de casi cualquier naturaleza cuyo contenido permite a los administradores reaccionar de manera efectiva en base a las comprobaciones que éste realiza.

El predecesor de SNMP fue SGMP *Simple Gateway Monitoring Protocol*, (R. Mauro & J. Schmidt, 2005) , definido en 1987 a través de la RFC 1028 por el IETF (*Internet Engineering Task Force*). SGMP fue diseñado por un grupo de investigadores universitarios versados en el ámbito de las redes, así como por usuarios y gestores. Este protocolo permitía únicamente la ejecución de una serie de comandos compatibles con routers. Dichos comandos admitían la lectura/escritura en destino de números enteros o cadenas de octetos. SNMP llegó para reemplazarlo un año más tarde, ya que fue presentado en 1988 con el fin de satisfacer la necesidad de administrar diferentes tipos elementos de una red a través de un protocolo estándar, que ya poco tenía que ver únicamente con las puertas de enlace. La primera versión aparece referenciada como estándar en la RFC 1157 y, al igual que su predecesor, se apoya sobre el protocolo UDP para realizar el intercambio de mensajes, siendo ambos protocolos de la capa de aplicación, como se puede apreciar en la Figura 4 (Wikipedia, 2021):

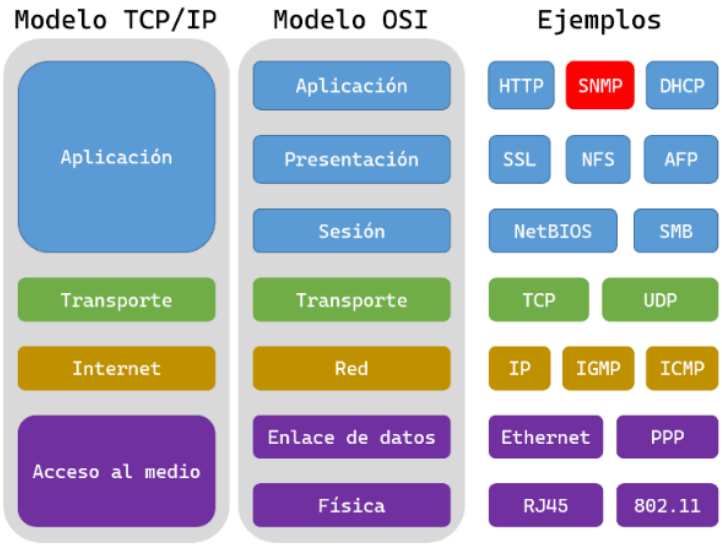

Figura 4. El SNMP en el modelo de capas. Elaboración propia.

La estructura de comunicaciones vía SNMP está conformada por elementos que cumplen únicamente dos roles (R. Mauro & J. Schmidt, 2005):

- ***Manager* o NMS *(Network Management Station)*:** este rol representa una estación, generalmente un servidor, que mantiene en ejecución algún tipo de servicio software que recopile los datos de aquellos elementos conectados a la red (ya sea vía *polling* o bien por la escucha de *traps)*, lo que también se puede entender como el lugar donde se ubica el servidor de monitorización. En este proyecto, este rol lo juega el servidor de Icinga, que por defecto únicamente admite la capacidad de interrogar de forma activa a los diferentes nodos configurados.

- ***Agentes*:** esta segunda entidad representa los elementos a monitorizar, *hardware* o *software*. Por norma general los dispositivos compatibles lo llevan integrado como un servicio propio del sistema operativo o *firmware*, aunque también existen agentes SNMP cuyo sustento es un servicio aparte (o un demonio, expresado en lenguaje *Unix).*

  Generalmente, el servicio SNMP de los agentes se mantiene a la escucha de peticiones, ya sean de lectura o de escritura (pueden leerse los valores de los registros que contienen la información o bien, enviar a dicho agente una solicitud SNMP de escritura de uno de esos registros, que modificará su valor si se cumplen una serie de condiciones). Estas peticiones serán procesadas y devuelven un valor a la entidad que interroga. Asimismo, los agentes también pueden generar los *traps* de notificación provocados por algún evento.

De forma general, este tipo de arquitectura se sustenta en un único *manager* y en una cantidad variable (bajo las circunstancias del entorno) de 1 a *n* agentes SNMP. Esto se aplica de igual manera para cada una de las 3 versiones en funcionamiento del protocolo.

De este modo, y siguiendo con la lógica de la Figura 3, expuesta en el capítulo anterior, podemos encajar estos roles con el comportamiento de las diferentes entidades, ofreciendo así una vista general de su funcionamiento mediante la pila de protocolos, tal y como se muestra en la siguiente figura:

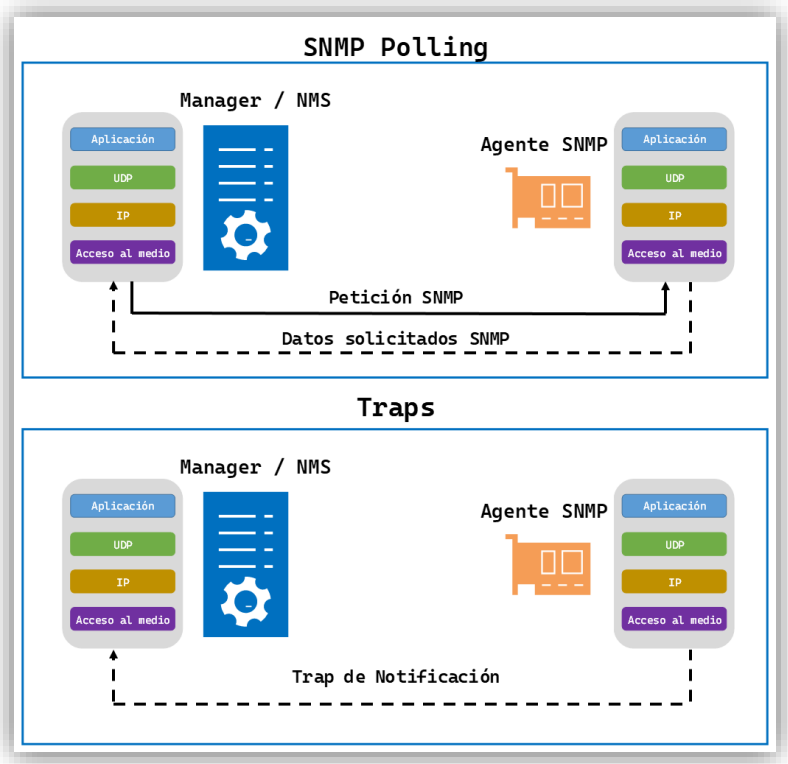

Figura 5. Entidades SNMP y su estructura. Elaboración propia.

3.2.2 Mensajes SNMP

Según lo especificado anteriormente, SNMP utiliza un protocolo no orientado a la conexión, como es UDP, para intercambiar mensajes entre ambas entidades SNMP. El puerto por defecto para la escucha de peticiones y respuestas SNMP en ambas entidades es el puerto UDP número 161, y el puerto destinado a la escucha de *traps* en el *manager* es el 162, si bien esta configuración es configurable en muchas ocasiones en función de las preferencias y necesidades del responsable de la infraestructura.

Cada uno de estos mensajes por norma general podemos dividirlos en las tres partes que se muestran en la Tabla 1 (IETF, 1990).

Fragmento	**Descripción**
Versión	En las transmisiones SNMP se debe especificar la versión del protocolo que se está usando, ya que de ello depende la utilización de una serie parámetros que pueden existir, o no, en función de dicha versión.
Comunidad	Se debe especificar la comunidad correcta para poder llevar a cabo la solicitud. Existen comunidades con dos niveles de permisos: lectura y escritura.
PDU	La unidad mínima de información de SNMP que contiene los campos, así como los datos a transmitir.

Tabla 1. Estructura de un mensaje SNMP.

La parte que tiene el mayor peso, como se puede apreciar, es la PDU, que puede tener dos tipos de estructura, en función de si son peticiones/respuestas, o *traps* de notificación.

Para el primer caso, podemos encontrarnos con una composición de PDU con la estructura mostrada en la Tabla 2 (IETF, 1990).

Campo PDU	**Descripción**
Tipo	La clase de comunicación o tipo de PDU enviada. Esta clasificación puede encontrarse en el siguiente apartado donde se definen los tipos de PDU existentes para cada versión de SNMP.
Identificador	Es un ID que utilizan ambas entidades para poder reconocer el registro OID al que se hace referencia en la petición.

Estado de error	En el caso de que se produzca algún tipo de error durante el procesamiento de la petición puede devolverse uno de los siguientes valores para conocer más información: - 0: No se han producido errores. - 1: Dato demasiado grande. - 2: El registro solicitado es inexistente. - 3: Valor incorrecto. - 4: El registro es de sólo lectura (para peticiones de escritura no válidas). - 5: Otro tipo de error.
Índice de error	Sólo se hace uso de este campo cuando el campo "Estado de error" es diferente de cero. Su uso está concebido para aportar más información acerca del error reportado.
Enlazado de variables	Es un campo que contiene nombres de registros y sus valores

Tabla 2. Estructura de una PDU de petición/respuesta.

Para el caso de los *traps* de notificación, hay un tipo de estructura diferente de la que hace uso el agente cuando hay un cambio de estado o evento, para el cual hay configurado un envío de información automatizado, tal y como se muestra en la Tabla 3 (IETF, 1990).

Campo PDU	**Descripción**
Tipo	En este campo figuraría que el mensaje enviado es un *trap*, y no una respuesta por parte del agente.
Enterprise	Contiene la identificación del subsistema SNMP que se envía desde el agente.
Dirección	La dirección IP del agente que envía dicha notificación.
Tipo genérico de trap	Algunos *traps* pueden ser genéricos, y pueden indicar mediante un valor numérico, sucesos tales como: un reinicio del agente, la caída de una interfaz de comunicación, etc.
Tipo específico de trap.	Los fabricantes pueden incorporar *traps* privados que ofrezcan reacciones a más situaciones aparte de los *traps* genéricos.
TimeStamp	Es la cantidad de tiempo que ha transcurrido desde que se inició el agente SNMP hasta que se ha producido el envío del *trap*.
Enlazado de variables	Este campo permite incorporar información adicional acerca del evento que se ha producido en el agente.

Tabla 3. Estructura de una PDU de tipo trap.

3.2.3 Versiones de SNMP

Desde su origen a finales de los ’80, encontramos en funcionamiento 3 versiones de SNMP (v1, v2 y v3) que se fueron sucediendo a medida que se añadían mejoras y soluciones a necesidades encontradas. En este apartado se pretende ofrecer un breve resumen de las variaciones encontradas en su evolución hasta la actualidad (R. Mauro & J. Schmidt, 2005).

3.2.3.1 SNMPv1

Es la versión inicial del protocolo y, como se ha especificado, es uno de los estándares históricos del IETF. El mecanismo de seguridad que implementa esta versión para poder hacer intercambios de comunicaciones entre dispositivos está basado en un parámetro llamado *comunidad*, que no es más que una cadena de texto que funciona a modo de contraseña con la que se identifica el *manager* durante la solicitud SNMP, a fin de establecer una relación de confianza. Si la cadena coincide con la que tiene configurada el agente, se puede realizar la operación. Típicamente podemos encontrar tres tipos de comunidades por defecto: *read-only* (sólo lectura), *read-write* (que permite realizar operaciones de escritura sobre algunos registros del sistema) y los *traps* (eventos esporádicos).

Por defecto, muchos de los agentes tienen configurado de manera genérica el valor “public” como comunidad de lectura y “private” para solicitudes de escritura. Es altamente recomendable su modificación, dada la peligrosidad que supone el conocimiento popular de esta costumbre. Una de las curiosidades en materia de seguridad que también se incluye con esta versión, es la capacidad de algunos agentes para configurar el envío de *traps* al *manager* cuando son interrogados a través de una comunidad que no está configurada en dicho agente.

En esta versión existen 5 tipos de comunicaciones o PDU (*Protocol Data Unit*) que pueden ser enviadas en comunicaciones entre agentes y *managers* (IBM, 2020):

- **GETREQUEST**: es la solicitud de información acerca de un registro enviada por el NMS al agente SNMP.

- **GETNEXTREQUEST**: también es una solicitud enviada por el *manager* al agente SNMP, la cual permite interrogar a dicho nodo por el siguiente registro al solicitado en la petición. Este tipo de PDU se utiliza fundamentalmente para poder recorrer todos los registros disponibles dentro del agente, y así tener una visión completa de todas las variables que presenta.

- **SETREQUEST**: es una petición de escritura que se envía desde el *manager* al agente para sobrescribir el registro indicado en la PDU de la petición.

- **GETRESPONSE**: Es la respuesta que envía el agente frente a cualquiera de las tres PDU de solicitudes especificadas hasta ahora.

- **TRAP**: este tipo comunicación hace referencia al mensaje enviado por el agente al *manager* sin haber sido solicitado por éste. Sucede ante la ocurrencia de eventos configurados previamente.

3.2.3.2 SNMPv2

Es también conocida ampliamente como SNMPv2c y aparece definida en las siguientes RFCs: RFC 3416, RFC 3417 y RFC 3418. En esta versión se sigue manteniendo el mismo mecanismo de funcionamiento que en la versión previa. De hecho, en términos de funcionalidad, no supone un cambio radical. Entre sus capacidades incluye el manejo de comunidades, que mantiene el problema de que las comunidades utilizadas durante el intercambio de información se envían en texto plano, lo que supone un riesgo evidente en términos de seguridad. Por lo tanto, es recomendable añadir a la infraestructura mecanismos de refuerzo para evitar vulnerabilidades. Un ejemplo de ello podría ser establecer reglas en el *firewall* que únicamente permitan tráfico procedente del *manager* a los agentes y no de otros orígenes, así como otras medidas que puedan reforzar esta carencia.

Por su parte, SNMPv2 incorpora dos tipos de PDU que se añadirían a las presentes en la versión previa (F.Kurose & W. Ross, 2010):

- **GETBULKREQUEST**: es una alternativa a GETNEXTREQUEST, la cual permite solicitar una cantidad considerablemente grande de datos (como, por ejemplo, tablas) en una sola petición que realice la operación de forma iterativa a diferencia de la anterior, que conllevaba también la iteración de solicitudes.

- **INFORMREQUEST**: este tipo de PDU permite transmitir información entre dos NMSs, a fin de notificar información sobre los elementos administrados.

Dada la amplia presencia y versatilidad que, aún a día de hoy, ofrece esta versión y su predecesora, es en las que se va a basar este proyecto de monitorización para su desarrollo.

3.2.3.3 SNMPv3

Es la última versión liberada hoy en día, aunque su fecha de publicación es del año 2002, y aparece regulada en la RFC 3410 (R. Mauro & J. Schmidt, 2005). Más tarde serían publicadas las RFC 3411 y RFC 3418, en las que se define SNMPv3 como estándar actual del protocolo, considerando las versiones anteriores como obsoletas. Aunque poco a poco se vayan incorporando sistemas que soportan SNMPv3, es todavía muy frecuente encontrar dispositivos que sólo implementan las versiones anteriores del protocolo, bien porque son reacios al cambio, o bien porque el ciclo de vida de esos dispositivos y elementos (cuyo diseño es anterior a la incorporación de SNMPv3) aún no se ha cerrado, por lo que además de la lentitud al cambio del mercado, se suma la existencia de dispositivos antiguos.

Como se ha comentado con anterioridad, una de las carencias más importantes de SNMP es en materia de seguridad y es precisamente aquí donde destaca la nueva versión, pues ofrece una solución criptográfica en las comunicaciones que se llevan a cabo entre el NMS y los agentes. Actualmente, la seguridad ya no se basa en una gestión de comunidades que se intercambian en texto plano, sino que se admite autenticación de usuario y contraseña, la cual se apoya en algoritmos como MD5 o SHA para no enviar también esas credenciales como algo legible. Adicionalmente, y como mecanismo de privacidad, ofrece un servicio de encriptación que utiliza el algoritmo DES. Éste es el responsable de encriptar y desencriptar los mensajes enviados. En la actualidad, éste es el único algoritmo incorporado para esta labor, si bien podrían agregarse otros en un futuro. Esto no evita, en ningún caso, que SNMPv3 no pueda ser objetivo de ataques por fuerza bruta o ataques de diccionario.

Otro de los cambios más importantes mediante la actualización de versión es que se abandonan los conceptos de *manager* y agentes SNMP, pasándose a llamar ambos: entidades SNMP. Por lo que cada una de las partes se concibe ahora como un mecanismo SNMP que hace uso del protocolo por sí mismo y que, adicionalmente, cuenta con una serie de aplicaciones internas que, según la RFC 3411, permitiría la agregación de más módulos conforme vaya avanzando el tiempo (Ver Figura 6).

Cabe destacar que en la versión 3 del protocolo, todas las entidades SNMP van identificadas por un identificador único llamado *SnmpEngineID*, que se usa como método de identificación unívoco de las entidades y nada tiene que ver con temas que puedan afectar al direccionamiento, registros DNS u otros. Por su parte, SNMPv3 no cambia el número de PDUs mencionadas en las versiones predecesoras, por lo que las aplicaciones internamente continúan utilizando las

expuestas en los apartados anteriores, si bien tienen diferencias en su composición, dadas las mejoras en materia de seguridad.

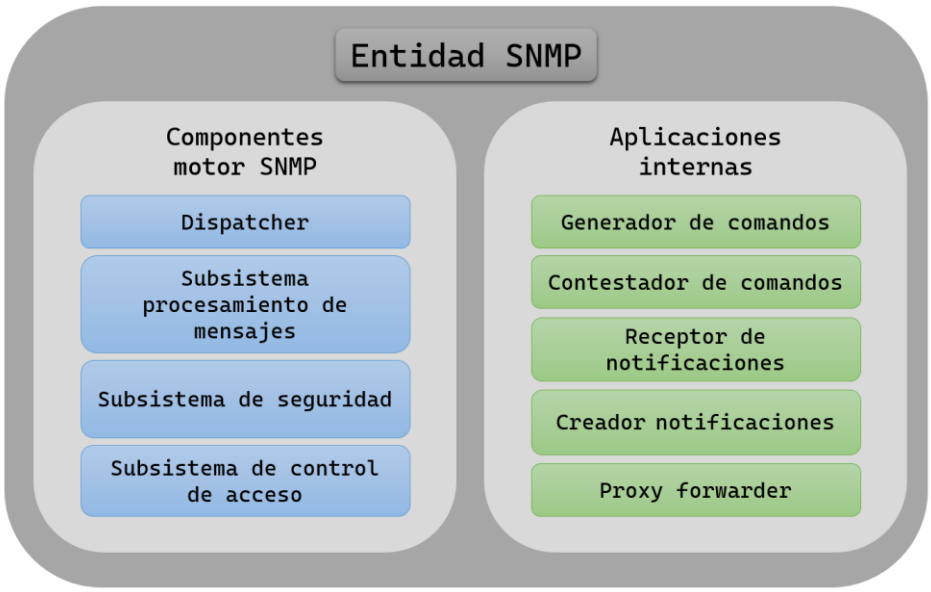

Figura 6. Entidad SNMP en SNMPv3. Elaboración propia.

3.2.4 MIBs

Como se ha especificado con anterioridad, las comunicaciones SNMP hacen referencia a un objeto (al que también se puede llamar registro), el cual a su vez también hace referencia a un valor que va cambiando a medida que varían las circunstancias. Dicho elemento puede ser leído o sobrescrito a través de una petición, ya sea bajo demanda o propiciado por algún evento.

Estos registros permanecen definidos en una estructura, a fin de ser clasificados en ámbitos, además de conceder un cierto orden para que facilite el acceso. Esta estructura puede ser entendida como una base de datos que contiene todos los objetos que el agente SNMP tiene configurados en el sistema. A esto se le conoce como la MIB, *Management Information Base* (F.Kurose & W. Ross, 2010).

Cualquiera de los objetos que se encuentran definidos en estas estructuras puede ser accedido por un NMS (u otros dispositivos, si cuentan con los parámetros adecuados) en un agente que tenga una definición compatible de esa MIB. Para cada uno de esos registros, figura una descripción en forma, generalmente, de comentario, una serie de valores que puede tomar (si procede, en caso de ser un valor acotado), así como el tipo de datos utilizado para representar la información.

La MIB es un fichero que utiliza una sintaxis propia y que permite organizar toda esta información en una estructura de forma jerárquica. Esta estructura puede ser interpretada de forma más visual a través de diferentes *softwares*. En la Figura 7 se ofrece la representación gráfica de una MIB (*RFC1213-MIB.mib*) mediante el uso del programa *MIB Browser*[9].

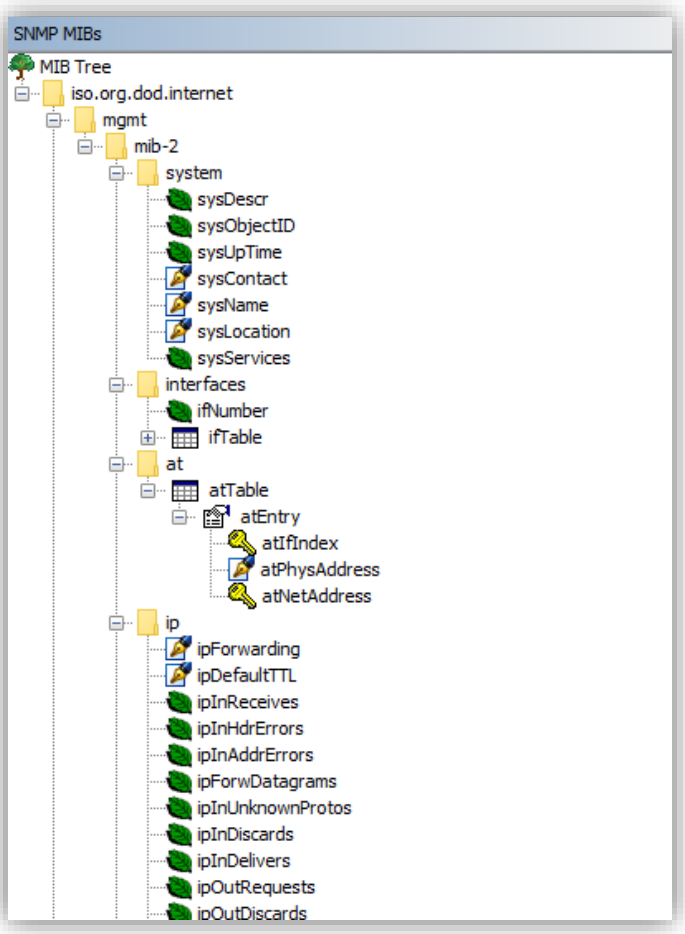

Figura 7. Representación gráfica de una MIB.

Como puede apreciarse en la Figura 7, hay dos tipos de elementos representados a lo largo de la estructura en forma de árbol; los elementos que son terminales o finales (a los que se llama escalares) y aquellos que, que bajo ellos, tienen la definición de otra serie de objetos, conocidos como tabulares. Por lo que, para acceder a uno de los registros terminales, habrá que especificar el camino recorrido desde la raíz de la MIB. Por ejemplo, un acceso a *ifNumber* conlleva especificar en una petición SNMP lo siguiente:

iso.org.dod.internet.mgmt.mib-2.interfaces.ifNumber

[9] https://www.ireasoning.com/mibbrowser.shtml

A esto hay que añadirle que, cada uno de los objetos recorridos internamente en la MIB están asociados con un valor numérico conocido como OID (*Object IDentifier*), que no es más que la consecución de todos los identificadores recorridos desde la raíz en la ruta al objeto solicitado, al igual que se ha hecho con los nombres anteriores. Para el caso anterior, el OID equivalente para *ifNumber*, sería:

.1.3.6.1.2.1.2.1

Este OID es la correspondencia seguida internamente por SNMP para acceder al valor de un registro, por lo que la MIB también actúa como traductor de OIDs, haciendo legibles a nivel humano tanto los campos referenciados en una solicitud como los *traps* recibidos en algún momento determinado. Sin esa interpretación, sería difícilmente comprensible a qué se está refiriendo la información asociada en una transmisión SNMP. Como se ha descrito anteriormente, cada OID tiene internamente programados los valores que puede tomar (en caso de ser un valor acotado por estados y no un valor libre, como sería la temperatura), así como las descripciones que se consideren oportunas y el tipo de datos utilizado. Para *ifNumber*, la definición dentro de la MIB para esa variable sería:

```
-- the Interfaces group

-- Implementation of the Interfaces group is mandatory for
-- all systems.

ifNumber OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The number of network interfaces (regardless of
            their current state) present on this system."
    ::= { interfaces 1 }
```

Siendo “interfaces 1” el último componente del OID especificado antes (.1.3.6.1.2.1.2.1), y el resto de identificadores situados a la izquierda, las partes superiores en la jerarquía vista en la Figura 7. Junto a esta definición, dentro del fichero MIB, figuran otras muchas referentes al resto de elementos que podrían recorrerse en el árbol.

Una vez explicado el concepto de MIB queda explicado, se puede entender más fácilmente lo que realizan algunas operaciones.

3.2.5 Operaciones a través de NET-SNMP

Anteriormente se ha hecho referencia a esta *suite* de herramientas que se usará a lo largo de la implantación de la plataforma de monitorización, y es ahora, una vez explicadas las principales características del protocolo SNMP, cuando conviene hacer hincapié en este paquete de herramientas, que incluye internamente una serie de aplicaciones utilizadas para interactuar de forma más intuitiva con SNMP. Si consultamos la documentación de la web oficial[10], podemos encontrar una clasificación de las aplicaciones incluidas por línea de comandos, las cuales serán utilizadas más adelante (Ver tabla 4).

Aplicaciones	**Descripción**
snmpget *snmpgetnext*	Mediante el uso de las PDU GETREQUEST y GETNEXTREQUEST vistas anteriormente, permiten solicitar (a un agente) un dato de un registro único referente la estructura MIB de un agente.
snmpwalk *snmptable* *snmpdelta*	Mediante GETBULKREQUEST, o iterando con las demás, permiten recorrer estructuras (bien de la MIB, o bien de una tabla) con el fin de conseguir una respuesta múltiple. *Snmpwalk* permite recorrer todos los OIDs disponibles de una MIB de un agente SNMP, funcionalidad muy útil.
snmpset	Permite la asignación de valores a registros que puedan ser modificados a través de una comunidad de escritura y haciendo uso de la PDU SETREQUEST.
snmpdf *snmpnetstat* *snmpstatus*	Estos comandos permiten recuperar información ejecutando de forma remota los comandos *df*, *netstat* y *service status snmp* en agentes que tengan compatibilidad. De esta manera, la información llega mejor representada.
snmptranslate	Uno de los comandos más útiles, ya que permite convertir entre el valor numérico de un OID y la correspondencia textual dentro de la MIB.

Tabla 4. Aplicaciones disponibles a través de NET-SNMP.

Además de estas aplicaciones, NET-SNMP tiene publicadas otras herramientas que pueden ser instaladas y configuradas de manera adicional en caso de necesitarse, entre las que se incluyen:

- *Tkmib*: un explorador gráfico de MIBs.
- *Snmptrapd*: un demonio que permite la escucha de *traps* de notificación, así como un archivo binario que permite enviarlos.
- *Snmpd*: permite instalar un agente SNMP en la máquina donde se despliegue.
- Una librería para desarrollar nuevas aplicaciones SNMP, llamada también NET-SNMP.

[10] http://www.net-snmp.org/docs/man/

3.3 Icinga Monitoring.

En esta sección se pretende ofrecer un contexto histórico del desarrollo de la herramienta, así como una visión teórica de la plataforma a utilizar durante el trabajo, con el fin de que los conceptos generales ayuden al lector a una mejor comprensión de la parte práctica, a la vez que le permite familiarizarse con la terminología.

3.3.1 Historia de Icinga

Icinga[11] es una herramienta *open source* de monitorización que comprueba la disponibilidad de los recursos presentes en una red para posteriormente, ofrecer una visualización de los resultados a través de una aplicación web, alertando así de comportamientos anómalos a los administradores del entorno. No obstante, no podría hablarse de Icinga, sin hablarse de Nagios[12], ya que este proyecto es un *fork* (entendido como una bifurcación en su desarrollo) del conocido sistema.

Por su parte, Nagios podría tener la misma definición, ya que a nivel funcional no difiere demasiado e igualmente es código abierto. Su fecha de lanzamiento fue durante la primera mitad de 1999 y rápidamente se hizo popular gracias a su versatilidad para recuperar información acerca del estado de casi cualquier parámetro de interés en un sistema (*hardware* o *software)*, así como por su capacidad modular, que se detallará a continuación. Nagios está escrito internamente en C y Perl (así como también Icinga). Inicialmente se llamaba NetSaint, pero tuvo que cambiar de nombre debido a un conflicto comercial con el nombre del producto. Aunque originalmente fue creado para funcionar sobre GNU/Linux, actualmente se pueden encontrar adaptaciones para otros sistemas Unix e incluso en Windows.

El motivo de la existencia de Icinga y el *fork* realizado sobre el proyecto original se debe a que el proyecto original tenía una evolución lenta al incorporar soluciones y parches, algo que llegaba con mucha lentitud a los usuarios, y que en muchas ocasiones terminaba con la insatisfacción de integradores que no veían llegar la evolución. A esto se sumaba que, en aquel entonces, el equipo que daba mantenimiento a Nagios era mucho más pequeño, dificultando aún más la salida de ese estancamiento. Es por esto que, un grupo de miembros de la comunidad de Nagios, así como desarrolladores de algunas de sus extensiones y personal de Netways encabezaron esta bifurcación, a fin de continuar con una evolución activa del proyecto.

[11] https://icinga.com/docs/icinga2/latest/doc/01-about/
[12] https://www.nagios.com/

En la primera versión liberada de Icinga se corrigieron algunos *bugs* que habían perdurado durante algún tiempo y se hicieron mejoras en las conexiones a base de datos. Además de ello, se añadió una API estandarizada que simplificase la integración con otros elementos (algo bastante útil para recuperar informes de rendimiento y en algunos procesos de negocio). Según lo publicado en el anuncio que tuvo lugar tras la bifurcación[13], cuando se liberó la primera versión del proyecto (2009) se hizo especial hincapié en que el sistema sería totalmente compatible con su predecesor en funcionamiento, a pesar de que se añadieran algunas mejoras nuevas solicitadas por la comunidad.

Ya en el año 2012, y debido a que el proyecto tuvo una buena acogida, se optó por publicar una versión del proyecto que contuviese una reescritura del código asociado al núcleo, y que reemplazaría al anterior corrigiendo algunas carencias detectadas (como la compleja configuración y limites en términos de escalabilidad). Esta nueva rama es la que conoce bajo el nombre de Icinga 2 (la usada en este proyecto) y que fue publicada como una versión estable dos años más tarde (2014). A día de hoy, Icinga es un sistema con bastante madurez que se encuentra extendido como solución en países de todo el mundo (fundamentalmente en empresas dedicadas al desarrollo *software*, las tecnologías de la información y las telecomunicaciones), sin importar demasiado el tamaño de la empresa en la que se encuentre, como se puede apreciar en gráfico de la Figura 8 (Enlyft, 2021):

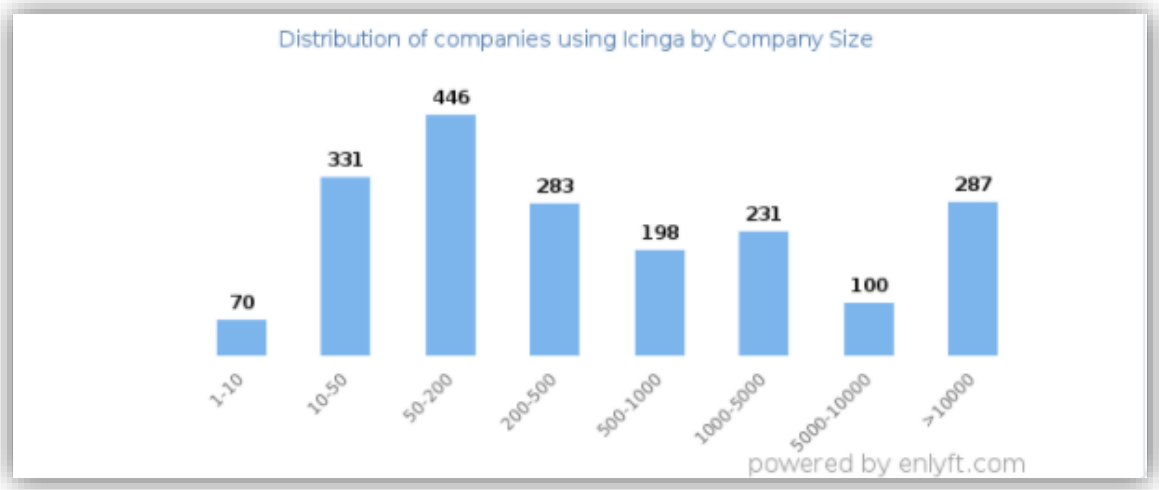

Figura 8. Compañías que usan Icinga en función del número de empleados.

Para hacerse a la idea del alcance, algunas de las empresas más notorias son, entre otras: Audi, Adobe, Siemens u ONO (Icinga Development Team, 2020). Aunque, obviamente, esto no es algo determinante, sí que deja entrever que herramientas como ésta (no necesariamente Icinga) son

[13] https://icinga.com/blog/2009/05/06/announcing-icinga/

determinantes en las tareas de monitorización llevadas a cabo diariamente por compañías de todo el mundo.

La última versión publicada de Icinga (a fecha de redacción de esta memoria) es de mediados del 2019. A modo de curiosidad para el lector, la palabra Icinga es un término de la lengua zulú que significa “busca”, “piensa” o “navega”.

3.3.2 Funcionamiento de Icinga y características principales.

El comportamiento de esta plataforma es sencillo de entender, pero requiere de una serie de nociones básicas antes de realizar cualquier configuración. Es por eso que, a nivel general, nos puede ayudar bastante la Figura 9.

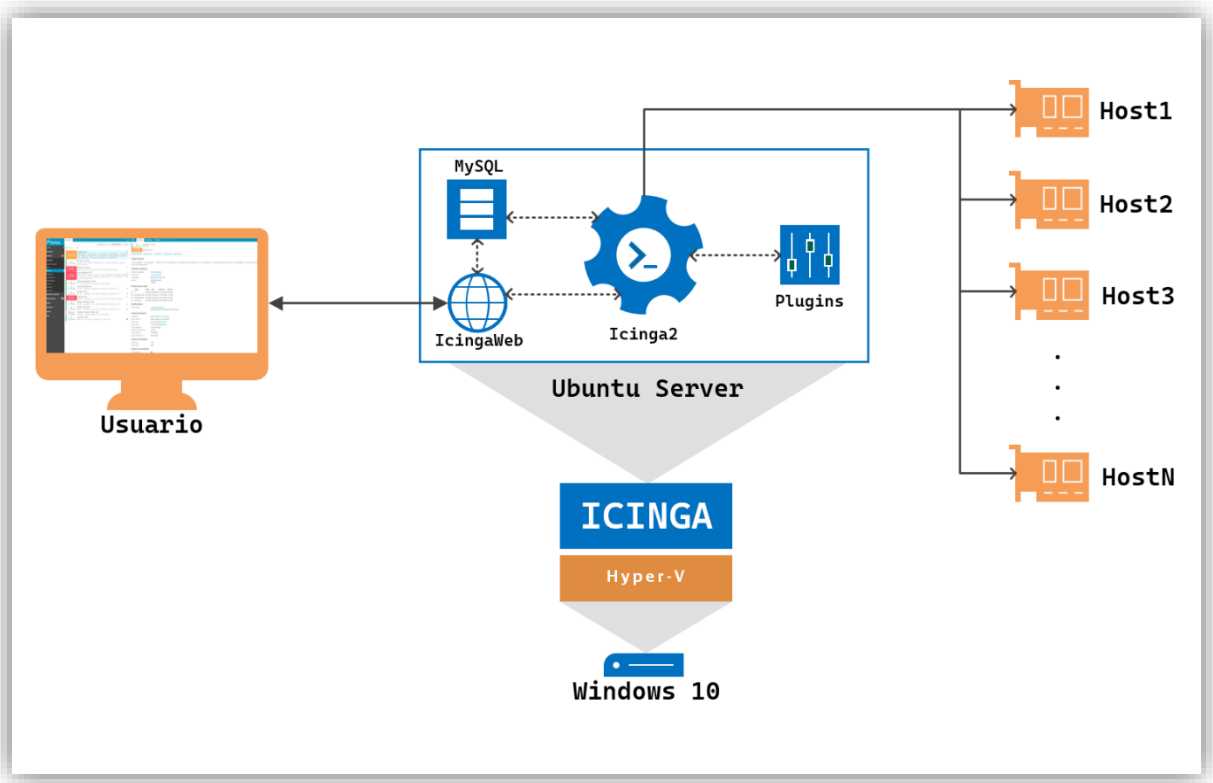

Figura 9. Esquema de funcionamiento de Icinga. Elaboración propia.

En la imagen anterior podemos ver una serie de elementos, los cuales conforman la arquitectura que se seguirá durante la puesta en marcha de esta herramienta. Un esquema que bien podría ser compatible con otros entornos cuyas necesidades sean similares y que, en el caso de sufrir variaciones, serían de aquellos elementos intermedios añadidos a la lógica de red (*firewalls*, *routers*, servidores, etc.). Como ya hemos dicho en el capítulo anterior, y para esta situación en

concreto, el sistema se apoyará sobre Hyper-V, no siendo en absoluto un requisito para el despliegue, dado que este tipo de implementaciones pueden llevarse a acabo en un servidor físico o incluso en la nube. Las partes concernientes a esta primera subdivisión (4) van enmarcadas en el interior del rectángulo de color azul de la figura anterior, y sus funciones principales son las que se describen a continuación en cada una de ellas:

- **MySQL**: es la base de datos o *backend* de la aplicación. En ella se almacenan todos los datos referentes a las comprobaciones realizadas a los sistemas, metadatos de la aplicación (los destinos configurados, variables que comprobar por cada uno, configuración del propio Icinga, etc.), entre otros.

  Durante la instalación de Icinga se solicita adicionalmente la configuración de IDO MySQL (*Database Icinga Data Output*) -en sistemas PostgreSQL hay un componente equivalente-, que es una característica que se utiliza para exportar los datos de configuración y estado de Icinga a base de datos.

- **Icinga Web**: es la parte *frontend* de la plataforma, y es el método de interacción con el usuario. Desde ella se puede comprobar la representación de los datos, así como realizar operaciones determinadas sobre la monitorización de forma manual. Adicionalmente pueden administrarse algunos parámetros del funcionamiento de la propia web (apariencia, usuarios, etc.). Desde aquí se pueden también habilitar/deshabilitar módulos desplegados previamente en el interior del servidor de Icinga, los cuales pueden añadir nuevas funcionalidades adicionales al conjunto total de herramientas dispuestas originalmente, siempre y cuando se satisfagan los prerrequisitos del de dicho módulo.

  Como se puede apreciar en la imagen, la parte web también hace uso de la base datos, ya que en ella se almacenan datos de usuarios, históricos, metadatos, etc.

- **Plugins**: son una serie de medios adicionales añadidos bajo demanda del administrador, que permiten, generalmente en implementaciones activas, interrogar a dispositivos de distinta manera en función del dato que se desee recoger. Esto incluye *scripts* desarrollados por el usuario, así como herramientas de terceros que, situados en el directorio adecuado dentro de la máquina virtual, son reconocidos por Icinga como un *set* de herramientas que pueden ser utilizadas en el caso de que se haya configurado alguna comprobación que dependa de ellas. Aquí figuraría *check_snmp* en el caso de este proyecto, junto a otras.

- **Icinga2**: es el motor de la aplicación, y su lugar sobre el sistema operativo es en forma de servicio. Dicho de servicio recoge la configuración establecida en unos ficheros donde se pueden encontrar todos los parámetros que condicionarán el comportamiento de la aplicación. Éstos están ubicados en diferentes directorios (se tratarán con mayor profundidad durante la parte práctica, a partir del apartado 4.4.2).

  En una configuración activa, Icinga cuenta con un planificador que introduce en una espera circular todas las comprobaciones que se van a realizar sobre los sistemas remotos, y a medida que va llegando su momento, se realizan las llamadas (con los parámetros configurados en los ficheros de configuración) al *plugin* concreto que interroga al destino acerca del dato. Si éste responde de manera adecuada, se representa la información, se almacena su estado y se vuelve a entrar en espera circular hasta que vuelva a ser su turno. En el caso de que el *host* remoto esté caído, podrían configurarse reintentos. Por último, en el caso de que éstos no sean satisfactorios se almacenaría esa ausencia en una base de datos, dando a entender que ese sistema no responde.

  Cuando la configuración es pasiva, Icinga no realiza operaciones sobre los equipos remotos, por lo que utiliza servicios adicionales que escuchan el dato (según corresponda) y éstos escriben los resultados en un fichero de comandos que lee Icinga continuamente (*icinga2.cmd*). Un caso frecuente de este tipo de implementaciones es el uso de NSCA (*Nagios Service Check Acceptor*).

Por lo demás, en la Figura 9 se pueden apreciar en el lado derecho a los sistemas remotos que se van a monitorizar que, en la nomenclatura de Icinga, son los llamados *hosts*, y que son establecidos como fuente de datos en la configuración de Icinga. Cada una de las variables comprobadas en un *host* (temperatura, disco, etc.) recibe el nombre de *service* o servicio. Por último, en el lado izquierdo de la imagen se encuentra el usuario(s) que, a través del navegador web, comprobaría(n) el flujo de información que se va representando en forma de estados.

Capítulo 4

Implantación de la monitorización vía SNMP en Icinga Monitoring.

4.1 Introducción

A lo largo de este cuarto capítulo se explicará con detalle cómo se llevaría a cabo un proceso práctico de implantación en base a los conocimientos adquiridos a lo largo del capítulo 3. El objetivo final es proporcionar una plataforma de monitorización completamente funcional con la que se pueda controlar un conjunto de elementos conectados a una red, vía SNMP. Para satisfacer este propósito, se contará con la herramienta Icinga como pilar fundamental del proceso.

Tal y como se ha citado anteriormente, lo que se pretende es ofrecer un conocimiento transversal a todas las fases que conlleva el despliegue de una herramienta de supervisión. Dicho proceso podría dividirse en 3 fases:

- **Instalación**: Esta primera fase, parte de un estado inicial en el que no hay nada configurado, salvo el propio sistema operativo, y se detallará paso a paso el procedimiento para satisfacer los requisitos de instalación de Icinga. Para ello se parte de una instalación limpia de Ubuntu Server 18.04.

- **Configuración**: En esta etapa figurará el proceso a seguir para hacer de Icinga algo realmente funcional con lo que supervisar una serie de recursos a través de SNMP. Dicho proceso contará con recursos monitorizados bajo una serie de necesidades y requisitos hipotéticos, que bien podrían darse en entornos reales.

- **Administración/manejo**: Por último, en esta parte se encontrarán las estrategias que permitirían al personal destinado al uso de la herramienta consultar la información obtenida, así como interpretar y manejar la plataforma.

Si bien estas etapas parecen bastante delimitadas, con el fin de facilitar el proceso al lector, algunos de los conceptos de las fases 2 y 3 se encuentran entrelazados para ir ofreciendo información de lo que se va logrando a medida que avanza la configuración. No obstante, al final de este capítulo es donde más se puede apreciar el contenido asociado a la tercera parte, dado que es donde figuran las pizarras, la interpretación de los semáforos y algunas nociones acerca del manejo de la interfaz web.

4.2 Instalación del entorno de monitorización.

Como se ha aclarado en la introducción, esta fase comienza con una instalación limpia de Ubuntu Server 18.04, y bastaría poder acceder al terminal en local o vía SSH (*Secure SHell*). En este trabajo, dicho sistema ha sido dispuesto en forma de máquina virtual sobre Hyper-V.

A pesar de tener una instalación reciente, es una buena práctica que el primer paso antes de comenzar una instalación sea actualizar la lista de paquetes disponibles en los repositorios configurados para la distribución utilizada. Además, dado que se van a instalar varios componentes, conviene hacerlo a través de su versión más reciente. Por lo tanto, se ejecuta en la línea de comandos:

```
$ sudo apt-get update
```

Como siguiente paso, se deberían satisfacer algunos prerrequisitos que son necesarios para continuar con la instalación. Es el caso de los siguientes paquetes:

- **apt-transport-https:** que habilita el transporte de descargas a través de *https* del gestor de paquetes, utilizando para este propósito *libapt-pkg*.
- **wget:** permite recuperar contenido (en forma de ficheros) alojado en internet, a través de *FTP*, *SFTP*, *HTTP* y *HTTPS*.
- **gnupg:** también conocido como *GNU Privacy Guard* es una herramienta de cifrado y firmas digitales licenciado bajo GPL (*General Public License*).

Por lo tanto, para cumplir con la instalación de estos prerrequisitos se ejecutará la siguiente orden:

```
$ sudo apt-get -y install apt-transport-https wget gnupg
```

Una vez completada la instalación, se puede proceder con el siguiente paso, que será añadir los repositorios de paquetes donde se encuentra Icinga, ya que de otra manera no podríamos descargar el software. Para ello, lo primero es descargar la clave pública del repositorio y añadirla al sistema:

```
$ sudo wget -O - https://packages.icinga.com/icinga.key | apt-key add -
```

A continuación, se añadirán a *sources.list* los repositorios necesarios para nuestra distribución concreta. Esto puede hacerse mediante la siguiente orden:

```
$ . /etc/os-release; if [ ! -z ${UBUNTU_CODENAME+x} ]; then
DIST="${UBUNTU_CODENAME}"; else DIST="$(lsb_release -c| awk '{print $2}')";
fi; \
 echo "deb https://packages.icinga.com/ubuntu icinga-${DIST} main" > \
 /etc/apt/sources.list.d/${DIST}-icinga.list
 echo "deb-src https://packages.icinga.com/ubuntu icinga-${DIST} main" >> \
 /etc/apt/sources.list.d/${DIST}-icinga.list
```

Tras haber añadido los repositorios, nuevamente es necesario actualizar el catálogo con las versiones de paquetes disponibles, ya que ahora, se encuentran entre ellos los necesarios para instalar Icinga.

```
$ sudo apt-get update
```

Una vez hecho esto, el gestor de paquetes podrá descargar la versión más reciente de Icinga a través de la siguiente orden:

```
$ sudo apt-get install icinga2
```

Si todo ha ido correctamente, icinga2 habrá quedado instalado en el sistema. Tras ello se procederá a instalar aquellos *plugins* que serán referenciados desde la configuración de icinga2 al monitorizar. Entre ellos, se encuentra el que se usará para efectuar las llamadas por SNMP (como se ha mencionado con anterioridad, este *plugin* aparece bajo el nombre de *check_snmp*).

```
$ sudo apt-get install monitoring-plugins
```

Por defecto, la instalación de estos *plugins* se encuentra en /usr/lib/nagios/plugins.

Una vez icinga2 está instalado, éste se manifiesta en el sistema como un servicio más. Por tanto, para comprobar que está funcionando correctamente podemos hacerlo a través de la orden:

```
$ sudo systemctl status icinga2
```

En el caso de que la instalación se haya realizado de manera correcta, debería mostrar que el servicio se encuentra ya en ejecución, como se muestra aquí:

```
  icinga2.service - Icinga host/service/network monitoring system
   Loaded: loaded (/lib/systemd/system/icinga2.service; enabled; vendor
preset: enabled)
  Drop-In: /etc/systemd/system/icinga2.service.d
           └─limits.conf
   Active: active (running) since Sat 2020-11-28 09:01:34 UTC; 3h 4min ago
 Main PID: 1495 (icinga2)
    Tasks: 50
   CGroup: /system.slice/icinga2.service
```

Para evitar que, tras un reinicio de la máquina virtual, el servicio no arranque, es conveniente habilitar la ejecución automática del servicio con la siguiente orden:

```
$ sudo systemctl enable icinga2
```

Ya que la mayor parte de la configuración de la plataforma se realiza a través de la edición de los ficheros de configuración interna de Icinga, es recomendable también instalar algún complemento que pueda ayudar al administrador a identificar cromáticamente los elementos de esos archivos según su sintaxis. Esto resulta extremadamente útil cuando los ficheros adquieren unas proporciones que dificultan su edición, y puede instalarse de la siguiente manera (el paquete está disponible tanto para *vim* como para *nano,* si bien en esta ocasión se ha optado por el primero):

```
$ sudo apt-get install vim-icinga2 vim-addon-manager
$ sudo vim-addon-manager -w install icinga2
```

Por ahora, con este proceso se ha completado la instalación del servicio de icinga2. Sin embargo, quedaría por realizar la instalación del motor de base de datos, así como de la parte web. Lo primero que se hace es comenzar por *MySQL* (como se ha especificado anteriormente, podría optarse por un motor de *PostgreSQL* o de *IcingaDB*, que aún se encuentra en desarrollo). Para ello, se instalará tanto la parte cliente, como la parte servidor con la siguiente orden:

```
$ sudo apt-get install mysql-server mysql-client
```

Una vez hecho esto conviene realizar los pasos para completar la instalación y securizar la instancia:

```
$ mysql_secure_installation
```

Tras esto, es recomendable añadir usuarios que nos permitan administrar la base de datos en remoto con algún cliente más amigable, aunque esto es una preferencia opcional para continuar con el proceso. El siguiente paso es instalar DB IDO MySQL (*Database Icinga Data Output*), que no es más que una característica que permite la exportación de la configuración de aquellos elementos configurados a base de datos. Esta tarea se realiza mediante la siguiente orden:

```
$ sudo apt-get install icinga2-ido-mysql
```

Durante el proceso se debe responder afirmativamente con el fin de que el componente se configure de manera automática. El siguiente paso es conectarse a la base de datos y crear un nuevo esquema, así como otorgar permiso al usuario *icinga* para operar sobre él con las siguientes órdenes:

```
> CREATE DATABASE icinga;
```

```
> GRANT SELECT, INSERT, UPDATE, DELETE, DROP, CREATE VIEW, INDEX, EXECUTE ON
icinga.* TO 'icinga'@'localhost' IDENTIFIED BY '*******';
```

Una vez creada la base de datos, se deberá importar el esquema de Icinga2 IDO descargado durante la instalación:

```
  # mysql -u root -p icinga < /usr/share/icinga2-ido-mysql/schema/mysql.sql
```

Tras la importación, la base de datos estará lista para conectar con la característica IDO MySQL, la cual debe ser habilitada a posteriori mediante la siguiente orden:

```
$sudo icinga2 feature enable ido-mysql
```

Cabe destacar que, para que este cambio tenga efecto, se deberá reiniciar el servicio de icinga2. La configuración adicional de esta característica podemos realizarla mediante la edición del fichero destinado a ello, ubicado por defecto en: */etc/icinga2/features-available/ido-mysql.conf*.

Una vez está lista la base de datos, se puede proceder con la parte web de la herramienta. En esta ocasión, se ha optado por hacerlo a través de *Apache*, aunque también se admite su despliegue sobre *Nginx*. Para instalar apache2, puede hacerse mediante:

```
$ sudo apt-get install apache2
```

En caso de tener la intención de securizar esta parte de la instalación, se deberán añadir las reglas de acceso necesarias en el *firewall* configurado:

```
$ sudo iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
$ sudo service iptables save
```

Icinga Web 2 se apoya sobre una REST API que permite al usuario interactuar con algunos elementos para realizar algunas operaciones como realizar reprogramaciones de las solicitudes a elementos que se vayan a configurar o recuperar datos de algunos objetos. Dentro de todo este proceso de instalación, la API se encuentra bajo la apariencia de una característica que podemos habilitar desde *icingacli*, a través de las siguientes órdenes:

```
$ sudo icinga2 api setup
$ sudo systemctl restart icinga2
```

Esto instalará los certificados necesarios y creará un usuario de la API con una contraseña autogenerada en el fichero de configuración (situado por defecto en */etc/icinga2/conf.d/api-users.conf*), la cual podrá ser modificada a posteriori, o bien añadiendo un nuevo usuario. Tras haber hecho esto, se debe completar el proceso reiniciando el servicio de icinga2.

Como se especificó en el capítulo 3, la parte web de Icinga es un *framework* de PHP. Por ello, ahora se debe poner a punto ese prerrequisito antes de instalar icingaweb2. Para esto, se actualizarán los repositorios, se instala el paquete *software-properties-common* (que ayuda a administrar más fácilmente algunos repositorios) y se añade el repositorio del que se descargará PHP. Para ello haremos uso de las siguientes órdenes:

```
$ sudo apt-get update
$ sudo apt -y install software-properties-common
$ sudo add-apt-repository ppa:ondrej/php
$ sudo apt-get update
```

En la documentación de Icinga2 se especifica que la versión de PHP deberá ser la 5.6.0 o superior. En esta ocasión se ha empleado la versión 7.4. Además del propio PHP, en la documentación se indica también que hay algunos módulos adicionales que habrá que instalar además de la parte básica. Dichos módulos cuentan con las siguientes funcionalidades (Ver Tabla 5).

Módulo	**Descripción**
Php-curl	Este módulo se utiliza para enviar comandos externos a través de la API de icinga2.
Php-gettext	Es un complemento que traduce código para facilitar su edición.
Php-intl	Es una extensión que se utiliza para la normalización de algunos elementos como los *timezones* u otros elementos de internacionalización.
Php-mbstring	Un módulo que se apoya sobre la librería *libmbfl* para dar soporte al chino simplificado, el chino tradicional, el ruso y el japonés.
Php-ldap	Necesario en el caso de que se desee implementar autenticación contra un directorio activo. En otro caso es opcional.
Php-mysql	O su módulo de *postgres* equivalente, en caso de haber optado por el otro tipo de *backend*. Es el módulo que hace posible la conexión a base de datos.
Php-gd	Para la exportación de datos y vistas en PDF
Php-imagick	Se utiliza para exportar gráficos en PDF.

Tabla 5. Módulos de PHP necesarios para realizar la instalación de IcingaWeb2

Se instala PHP, satisfaciendo dichos prerrequisitos con el siguiente conjunto de órdenes:

```
$sudo apt -y install php7.4
$ sudo apt-get install php7.4-curl
$ sudo apt-get install php7.4-gettext
$ sudo apt-get install php7.4-intl
$ sudo apt-get install php7.4-mbstring
$ sudo apt-get install php7.4-ldap
$ sudo apt-get install php7.4-mysql
```

Tras esto, quedaría instalar el paquete que provee la compatibilidad entre apache2 y PHP, así como el propio icingaweb2. Para ello se introduce el siguiente comando:

```
$ sudo apt-get install icingaweb2 libapache2-mod-php icingacli
```

Una vez hecho esto, es recomendable, antes de iniciar la configuración web, crear un esquema de base de datos que pueda utilizar la web, así como un usuario destinado exclusivamente para ello como se muestra a continuación:

```
$ mysql –u root -p

  > CREATE DATABASE icingaweb2;
  > GRANT ALL ON icingaweb2.* TO icingaweb2@localhost IDENTIFIED BY '******';
```

Sería también de utilidad la creación de un usuario nuevo en IDO Mysql. Para ello, se edita el fichero *ido-mysql.conf*, y se añade una nueva conexión:

```
object IdoMysqlConnection “ido-mysql” {
        user = “icinga2”,
        password = “*****”,
        host = “localhost”,
        database = “icinga2”
}
```

Como último paso de la instalación, se debería (de no haberlo hecho ya), crear un usuario nuevo de la API. Editando el fichero *api-users.conf*, añadiríamos una nueva entrada para dicho usuario:

```
object ApiUser "apiuser" {
        password = "*****"
        // client_cn = ""
        permissions = [ "*" ] //otorgamos todos los permisos
}
```

En este punto podría decirse que concluye la instalación de paquetes y requisitos para el funcionamiento de la herramienta.

4.3 Configuración de la infraestructura

Una vez se ha efectuado la instalación de todos los componentes necesarios para el funcionamiento de Icinga, el siguiente paso es generar un *token* de Icinga que nos servirá como mecanismo de seguridad para comenzar con la configuración inicial, referida únicamente a la

infraestructura, no a elementos monitorizables. Dicho *token* se generará mediante la ejecución de la siguiente orden:

```
# icingacli setup token create
```

Una vez ejecutado el comando, éste devolverá el código de la siguiente manera:

```
The newly generated setup token is: fdb7035e6267d28f
```

Tras ello, el siguiente paso será abrir un navegador web y dirigirse a la siguiente dirección:

http://*ipodnsdelentorno*/icingaweb2/setup

Esto lleva a un *wizzard* que se debe completar para concluir con la fase inicial de configuración. Como primer paso, se solicitará el *token* generado previamente (Ver Figura 10).

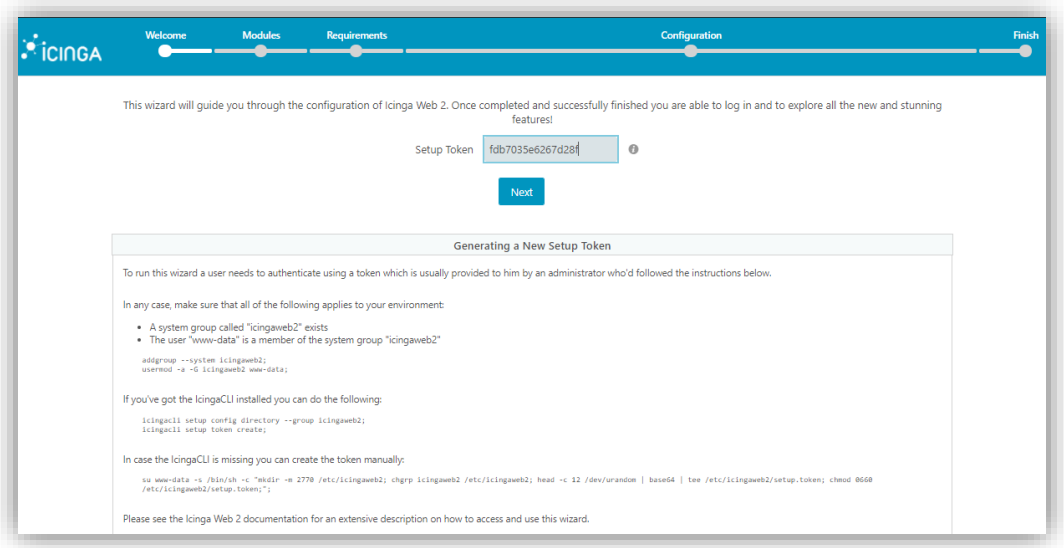

Figura 10. Pantalla inicial de configuración de Icinga Web 2.

A continuación, se ofrece al usuario la posibilidad de habilitar alguno de los módulos adicionales, como son el de migración de la plataforma o el de entorno de pruebas. En este caso únicamente queda marcado el de *Monitoring*, que es el que aplica para el proceso de monitorización descrito en esta memoria.

El paso siguiente es un verificador de compatibilidad y presencia de prerrequisitos. En caso de no haber instalado algún componente, o haber desplegado versiones incompatibles de los

mencionados en esta primera fase, se notificará al usuario de ello. Por el contrario, si la instalación cumple con todos los criterios debería tener una apariencia similar a la mostrada en la Figura 11:

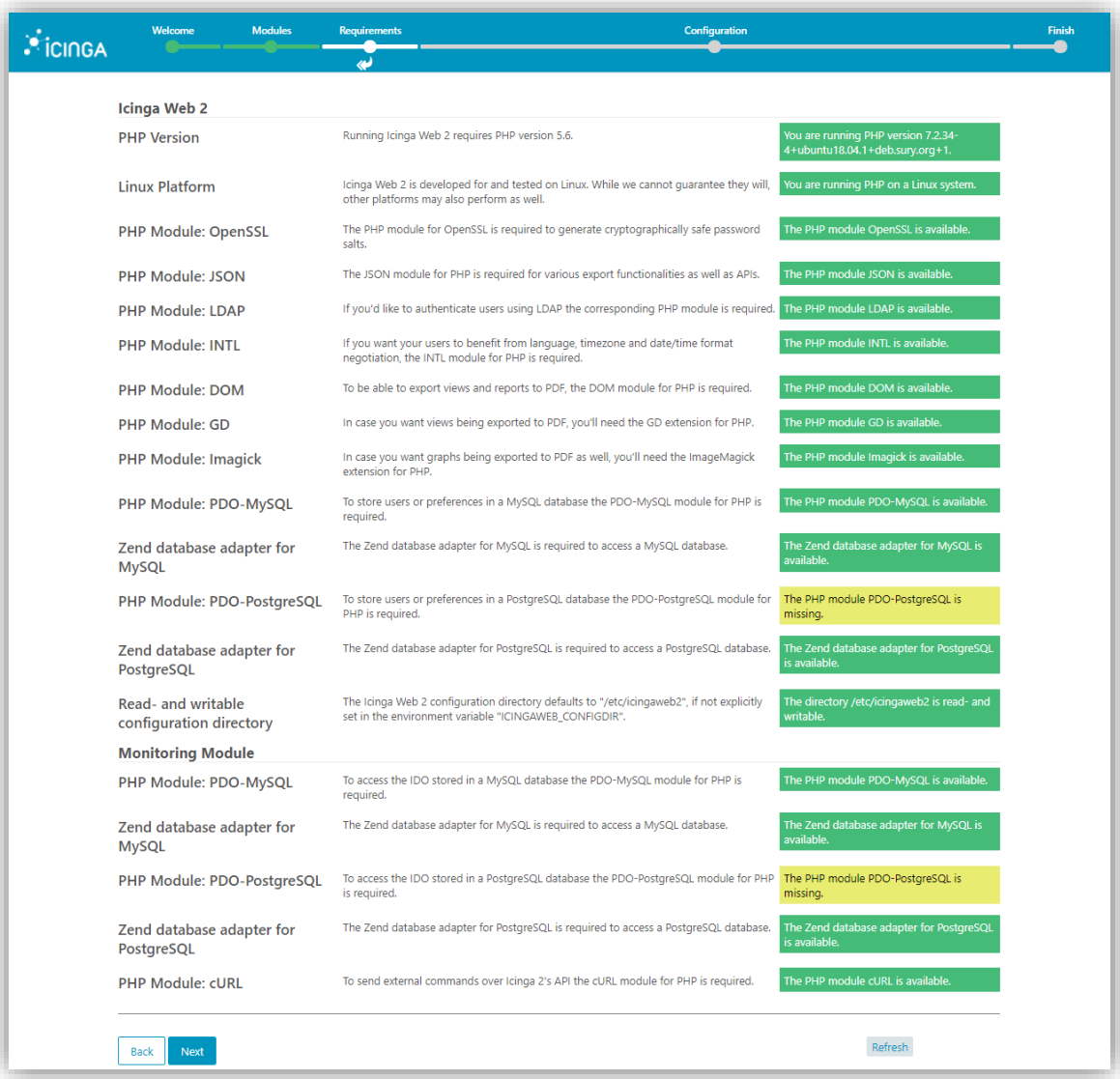

Figura 11. Satisfacción de prerrequisitos de Icinga.

En la Figura 11 se muestran dos prerrequisitos sin cumplir, dado que no son necesarios en esta implementación. En ambos casos, son módulos que hacen referencia a *PostgreSQL*, no necesarios dado que se hace uso de *MySQL* en su lugar.

El siguiente paso consiste en la especificación de valores que debe tomar *icingaweb2* para hacer uso de la base datos (Ver Figura 12).

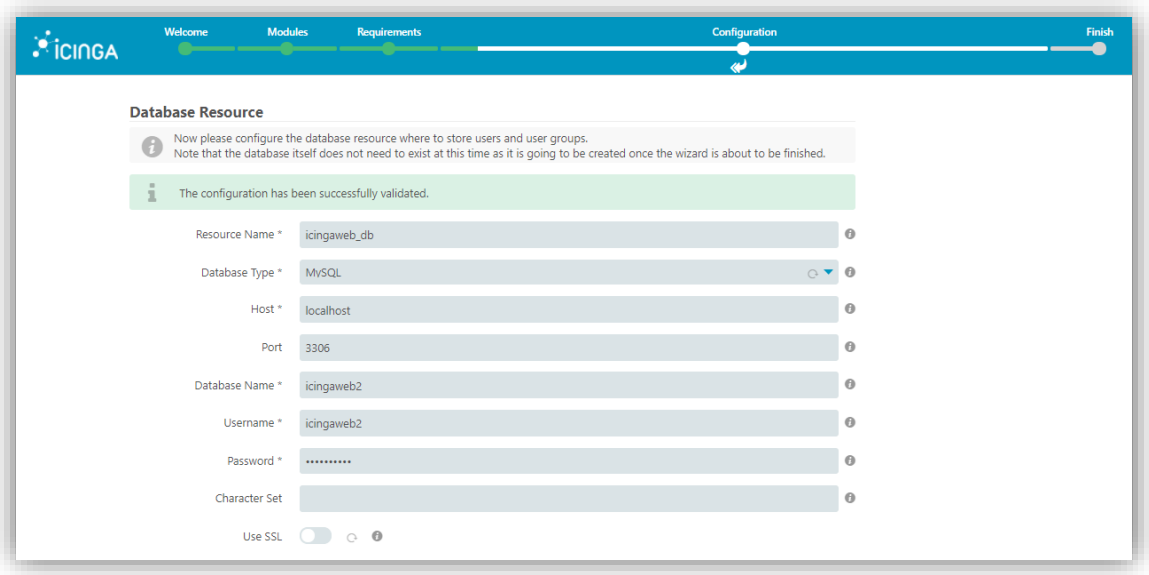

Figura 12. Introducción de parámetros para la configuración de la base de datos con icingaweb2.

A continuación, una vez validada la conexión, se deberá introducir un usuario y una contraseña (ver Figura 13), los cuales se utilizarán para el acceso web a la herramienta de monitorización. Por defecto, éste será un usuario con permisos globales de administración sobre todos los aspectos de los que se puede hacer uso a través de la web. Éste, a su vez, también podría dar de alta nuevos usuarios web.

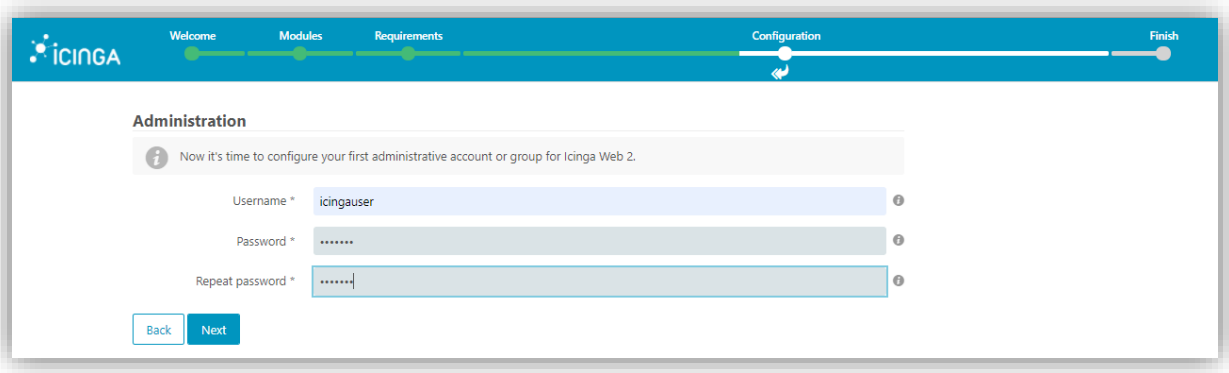

Figura 13. Creación de un usuario para la web con permisos globales.

En el siguiente paso se preguntará al usuario acerca del comportamiento que desea obtener en cuanto al tratamiento del log de la aplicación: contra qué fichero volcar la información, si se guarda en base de datos y el nivel de error deseado (ver Figura 14).

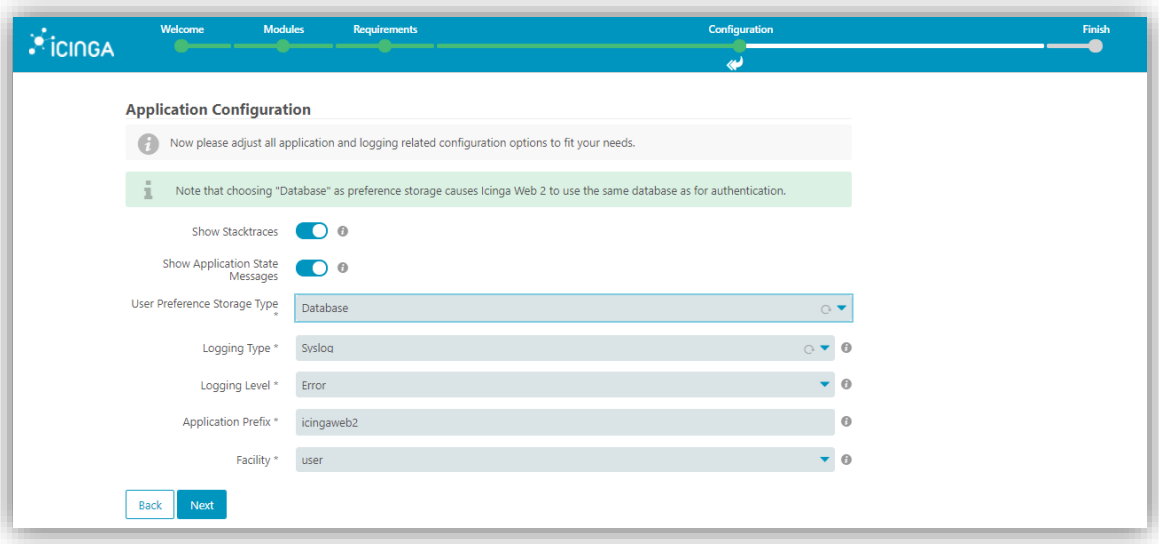

Figura 14. Configuración del log de icingaweb2.

Después, se muestra un resumen de los ajustes realizados previamente, y se indica al usuario que la configuración de la web ha terminado satisfactoriamente. Aunque el *wizzard* no termina todavía, ya que quedarían por especificar algunos parámetros más (por ejemplo, los ajustes de conexión a la base de datos utilizada por IDO MySQL). Este formulario es muy similar al mostrado en la Figura 12, como se puede apreciar a continuación (ver Figura 15).

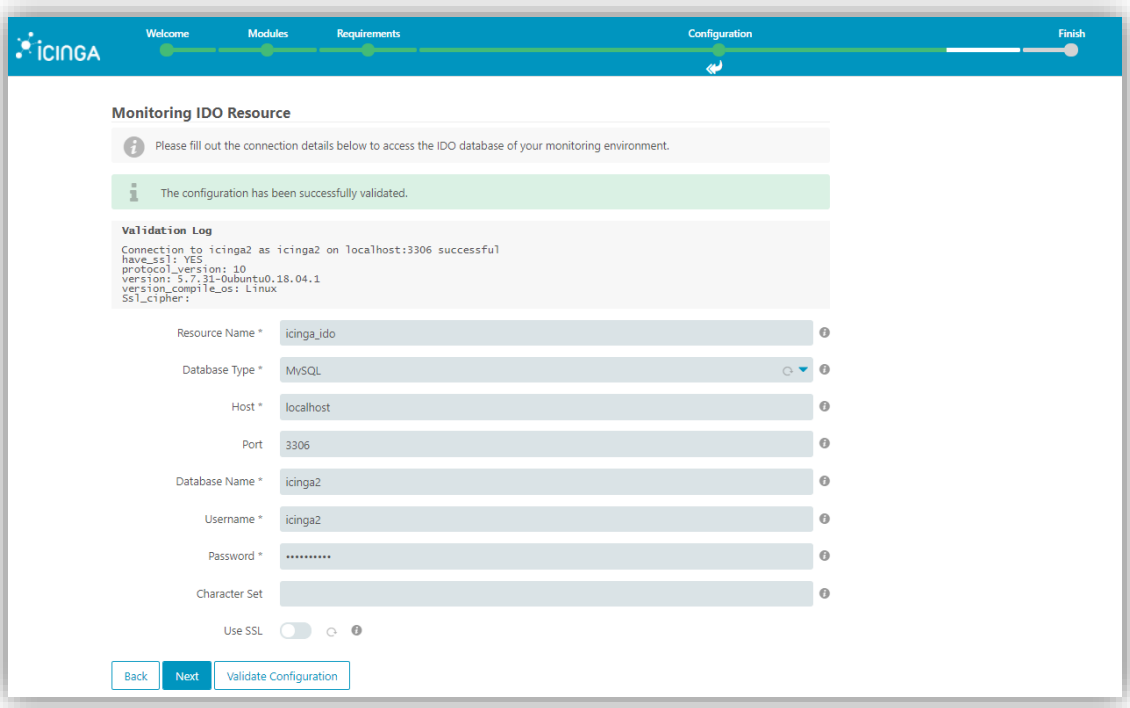

Figura 15. Especificación de parámetros para la conexión a base de datos utilizada por IDO MySQL.

Posteriormente, se debe especificar también el usuario creado anteriormente para la REST API, el cual será utilizado para el envío de comandos a aquellos elementos que se configuren posteriormente (ver Figura 16).

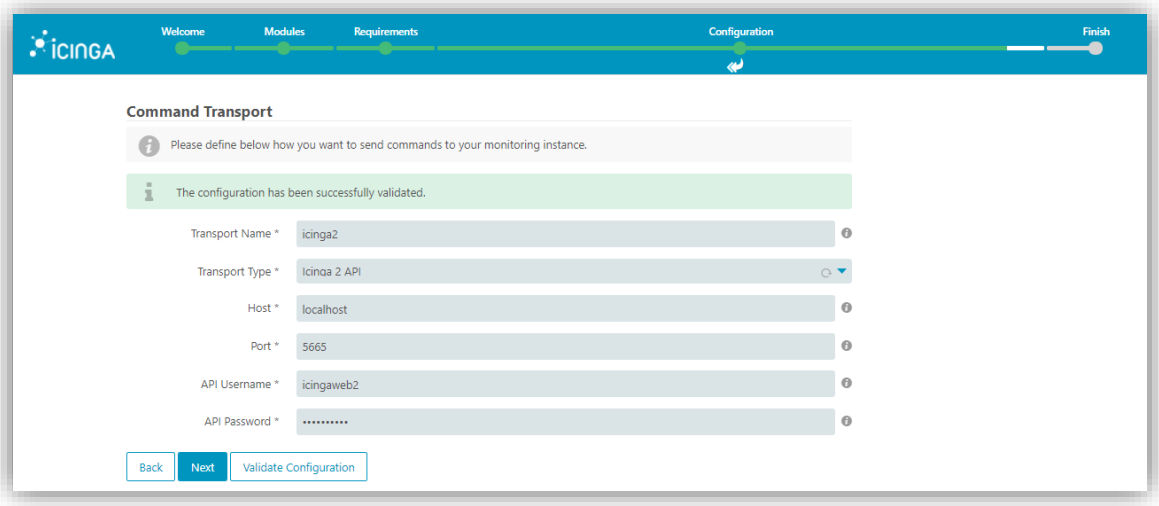

Figura 16. Especificación de parámetros de conexión a la REST API de Icinga.

Cabe destacar que, para el envío manual de comandos vía web, hay que habilitar una característica de icinga2 llamada *command* y reiniciar el servicio de icinga2. De otro modo, a pesar de haber configurado este elemento, no podrán enviarse dichos comandos (solicitudes de comprobación manuales adicionales a las programaciones de supervisión).

Una vez se ha completado este último paso, se muestra al usuario un nuevo resumen con estos últimos parámetros especificados, por si desea hacer algún cambio de última hora. Finalmente se visualiza una pantalla de verificación en caso de que todo haya ido bien (ver Figura 17).

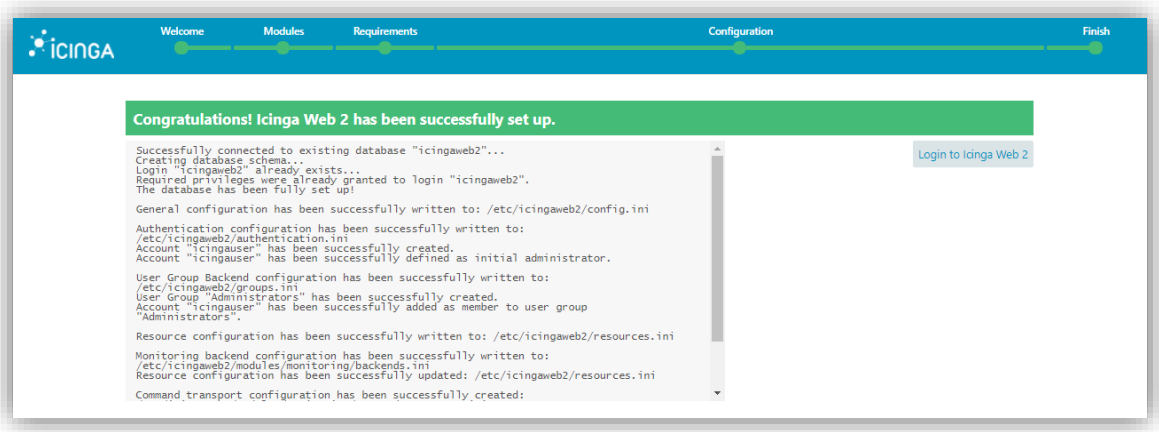

Figura 17. Pantalla de finalización de configuración inicial de Icinga.

En este punto, podría decirse que la fase de instalación ha concluido. Por lo tanto, el usuario podrá introducir los datos de inicio de sesión configurados previamente en la pantalla de *login* (ver Figura 18).

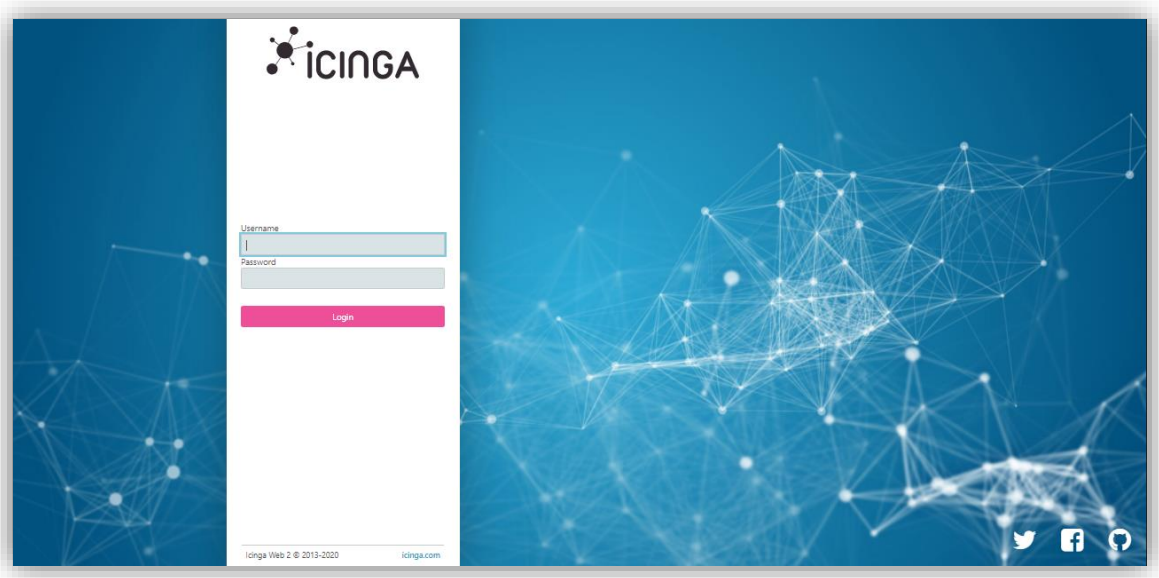

Figura 18. Pantalla de inicio de sesión a través de Icinga Web 2.

4.4 Implantación de recursos a monitorizar

Una vez realizada la instalación y configuración inicial de la plataforma de monitorización, cabe destacar que, a partir de este momento, todos los cambios que se hagan en la configuración de Icinga ya no irán enfocados a cambiar la propia herramienta sino que servirán para incluir nuevos elementos en la monitorización de los recursos.

A pesar de que en este trabajo se haga especial hincapié en la utilización de SNMP para supervisar sistemas, lo primero que habría que determinar ante cualquier situación genérica, una vez completado el proceso anterior, es el tipo de sistema del que se desea obtener información, con el fin de acotar el espectro de estrategias para llevar a cabo la obtención de datos. En ocasiones, puede interesar un tipo de monitorización muy específica para un elemento concreto; a lo mejor es más viable, por ejemplo, realizar esa tarea a través de *scripting* y NSCA o, por el contrario, podría ser de utilidad supervisar los elementos de forma activa a través de un proceso más sencillo. Es por esto que esta memoria parte de un punto hipotético en el que ya se habría considerado utilizar SNMP en detrimento de otros procedimientos. Sin embargo, no hay que olvidar que estas decisiones no son excluyentes entre sí y puede optarse por modelos híbridos.

Una vez concretado lo anterior, la siguiente tarea a realizar en un proceso de monitorización con SNMP sería concretar qué parámetros o constantes son susceptibles de ser consultados en los dispositivos, pues la mayoría de los agentes SNMP pueden llegar a incorporar en su configuración cientos de valores con información que puede ser consultada/editada. Con el fin de hacer de esta parte de la memoria algo púramente práctico pero, al mismo tiempo, facilitar la lectura la misma, se ha optado por escoger un primer agente SNMP a monitorizar que no entrañe demasiada complejidad para ser integrado dentro de la configuración, si bien más adelante se añadirá algún ejemplo con agentes que incluyen MIBs más extensas. No obstante, el proceso de integración sería exactamente el mismo.

Antes de comenzar con la incorporación del primer objeto a monitorizar, conviene explicar la lógica de los elementos que intervienen en la configuración de Icinga (en su modalidad de configuración activa) para la creación de nuevos objetos (ver Figura 19).

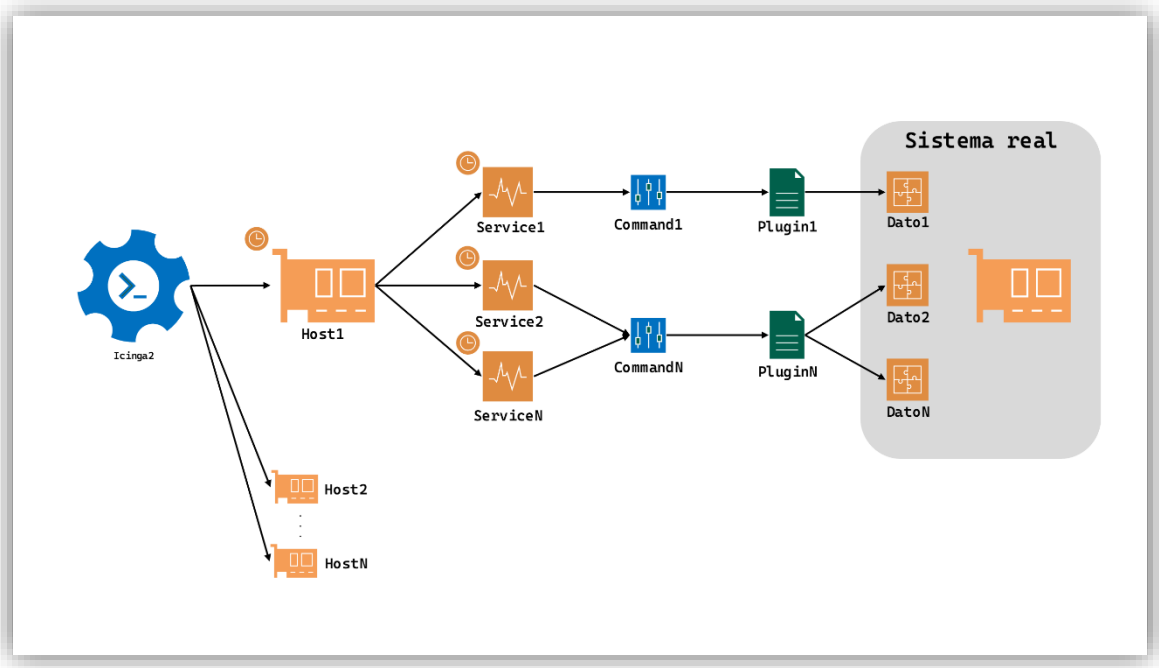

Figura 19. Esquema de configuración de los objetos de Icinga en configuración activa. Elaboración propia.

Como se puede apreciar en la Figura 19, Icinga es un servicio que, en base a una planificación periódica, comprueba el estado de una serie de objetos dados de alta en su configuración; que son aquellos marcados con un reloj naranja. La lógica de conjunto a este respecto se enmarca dentro de lo explicado en el apartado 3.3.2.

En la Figura 19 figuran, por un lado, los objetos de tipo *Host*, que son aquellos representados por una tarjeta de red. Éstos representan dentro de la configuración de Icinga a cada uno de los elementos que son terminales remotos de los que se desea conocer su estado. Por otro lado, cada

Host contiene una serie de S*ervices*, que serían cada una de las constantes a monitorizar en el mismo, y que también habrá que dar de alta en Icinga. Cada uno de esos S*ervices* (en el supuesto actual, con SNMP) contendrá la configuración de un OID, ya que el dato que devuelve hará referencia a un único objeto de la MIB.

Cuando Icinga va a comprobar el estado de estos *Services*¸ hace uso de un *Command*, que no es más que un conjunto de atributos vacío que contendrá, durante la llamada al *plugin*, aquellos parámetros necesarios para recuperar el dato al que se hace referencia. En el caso de este trabajo, en el que se hace uso de SNMP, esto equivaldrá a cada uno de los campos necesarios para realizar la petición (versión del protocolo, comunidad, etc.). Una vez se ha establecido en Icinga la configuración del *Service* y del *Host*, el servicio de *icinga2* utilizará esos atributos para rellenar los campos vacíos del *Command*. Cuando éste tiene todos los parámetros recopilados referentes a la solicitud SNMP, realizará la llamada al *plugin* (que bien podría ser un *plugin* de Nagios Plugins, como es el caso, o un script que espere datos de entrada). En la solución que se está aplicando, donde SNMP es el medio de comunicación, este papel lo jugaría *check_snmp*. Una vez se ejecuta el *plugin* con todos los parámetros, se envía la solicitud SNMP y el sistema queda a la espera de una respuesta por parte del objeto interrogado.

En la Figura 19, se puede observar que dos de los *services* apuntan a un mismo *command* ya que, al ser ambos objetos comprobados vía SNMP, pueden utilizar el mismo *command* que, por extensión, hace uso del mismo *plugin*, aunque la ejecución del mismo se realizaría con parámetros obviamente diferentes (como mínimo variaría su OID). Por otro lado, *Service1* y *Command1* representan objetos que podrían ser recopilados mediante otro tipo de estrategia o protocolo, y haciendo uso de otros *plugins.*

Los datos que obtiene Icinga de un sistema remoto clasificarían su estado en dos tipos: por un lado, el estado de los *services* (por ejemplo: espacio libre en disco, memoria consumida, etc.), que es la parte que se acaba de describir. Por otro lado, se encuenta la comprobación del sistema remoto en sí mismo, también conocido como el estado del *host*, que haría referencia a la disponibilidad del equipo remoto como tal. El estado del *host* también es comprobado mediante algún *plugin*, *s*in embargo, no se ha añadido a la figura para facilitar su comprensión. La parte parte relacionada con ambos estados se describirá con más detalle a lo largo del apartado 4.5.

Todos los parámetros recogidos en la Figura 19, y otros que serán utilizados a lo largo del proceso de configuración, se pueden dar de alta en los ficheros de configuración de icinga2 que, por defecto, se encuentran en /etc/icinga2/conf.d/.

4.4.1 Identificación de objetos por supervisar

Supóngase un entorno en el que pudiera resultar de especial interés la monitorización de dispositivos móviles (*smartphones o tablets*) con Android, y que éstos, a pesar de no incorporar de forma nativa en su configuración un agente SNMP, fueran motivo suficiente para iniciar una supervisión que englobase decenas o cientos de ellos.

Para solventar la problemática de la escucha por SNMP, se hará uso de un proyecto desarrollado por un usuario de GitHub[14]. Este proyecto se compone de una aplicación (que puede ser descargada en forma de fichero .apk) que se comporta como un agente SNMP (dicho proyecto cuenta también con una aplicación en Java que realiza la función de *manager*, y que **no** será utilizado en este proyecto ya que ese rol lo juega Icinga en este caso), así como una MIB que contiene todos los parámetros monitorizables. Una de las ventajas de este agente es que podría ser ampliado en el futuro con más funcionalidades, desarrollando procedimientos que permitieran recoger más datos mediante la ampliación de su MIB, así como de los mecanismos que lo hacen posible.

El siguiente paso a seguir, con este o con cualquier agente SNMP a controlar, debería ser conocer los parámetros que hay disponibles en el agente, descartando aquellos elementos que no sean de nuestro interés, y anotando aquellos que, por el contrario, sí lo son. Por lo tanto, en primer lugar, es de utilidad familiarizarse con la MIB en cuestión, a fin de conocer las posibilidades que ofrece (un agente puede implementar varias MIBs; en este caso, convendría revisar todas aquellas compatibles). Para el supuesto actual, la MIB de *ANDROID-MIB* se presenta a coninuación:

```
ANDROID-MIB DEFINITIONS ::= BEGIN

IMPORTS
       enterprises, TimeTicks
              FROM RFC1155-SMI
       OBJECT-TYPE
              FROM RFC-1212;

ufrgs                OBJECT IDENTIFIER ::= { enterprises 12619 }
android              OBJECT IDENTIFIER ::= { ufrgs 1 }
system               OBJECT IDENTIFIER ::= { android 1 }
services             OBJECT IDENTIFIER ::= { android 2 }

-- The hardware status.

hardware             OBJECT IDENTIFIER ::= { android 3 }
```

[14] https://github.com/brnunes/Android-SNMP

```
manager             OBJECT IDENTIFIER ::= { android 4 }

srvcNumber  OBJECT-TYPE
       SYNTAX     INTEGER
       ACCESS     read-only
       STATUS     mandatory
       DESCRIPTION
              "The number of services currently running on device."
       ::= { services 1 }

srvcTable  OBJECT-TYPE
       SYNTAX SEQUENCE OF SrvcEntry
       ACCESS     read-only
       STATUS     mandatory
       DESCRIPTION
              "A list of the services currently running on device.
The number of entries is given by the value of srvcNumber."
       ::= { services 2 }

srvcEntry  OBJECT-TYPE
       SYNTAX       SrvcEntry
       ACCESS     read-only
       STATUS     mandatory
       DESCRIPTION
              ""
       INDEX { srvcIndex }
       ::= { srvcTable 1 }

SrvcEntry ::= SEQUENCE {
       srvcIndex
              INTEGER,
       srvcDescr
              OCTET STRING,
       srvcRunningTime
              TimeTicks,
       srvcMemoryUsed
              INTEGER
}

srvcIndex  OBJECT-TYPE
       SYNTAX     INTEGER
       ACCESS     read-only
       STATUS     mandatory
       DESCRIPTION
              ""
       ::= { srvcEntry 1 }

srvcDescr  OBJECT-TYPE
       SYNTAX     OCTET STRING
       ACCESS     read-only
       STATUS     mandatory
       DESCRIPTION
```

```
            ""
      ::= { srvcEntry 2 }

srvcRunningTime  OBJECT-TYPE
      SYNTAX     TimeTicks
      ACCESS     read-only
      STATUS     mandatory
      DESCRIPTION
            ""
      ::= { srvcEntry 3 }

srvcMemoryUsed  OBJECT-TYPE
      SYNTAX     INTEGER
      ACCESS     read-only
      STATUS     mandatory
      DESCRIPTION
            ""
      ::= { srvcEntry 4 }

hwBatteryStatus  OBJECT-TYPE
      SYNTAX     INTEGER (0|1)
      ACCESS     read-only
      STATUS     mandatory
      DESCRIPTION
            "Battery charge level of the device:

            0 - OFF
            1 - ON"
      DEFVAL  { 0 }
      ::= { hardware 1 }

hwBatteryLevel  OBJECT-TYPE
      SYNTAX     INTEGER (0|1)
      ACCESS     read-only
      STATUS     mandatory
      DESCRIPTION
            "Battery level of the device:

            0 - 100 %"
      ::= { hardware 2 }

hwGPSStatus  OBJECT-TYPE
      SYNTAX     INTEGER (0|1)
      ACCESS     read-only
      STATUS     mandatory
      DESCRIPTION
            "The GPS Status:

            0 - OFF
            1 - ON"
      ::= { hardware 3 }
```

```
hwBluetoothStatus  OBJECT-TYPE
       SYNTAX     INTEGER (0|1)
       ACCESS     read-only
       STATUS     mandatory
       DESCRIPTION
              "The Bluetooth Status:

              0 - OFF
              1 - ON"
       ::= { hardware 4 }

hwNetworkStatus  OBJECT-TYPE
       SYNTAX     INTEGER (0|1)
       ACCESS     read-only
       STATUS     mandatory
       DESCRIPTION
              "The Network Status:

              0 - OFF
              1 - ON"
       ::= { hardware 5 }

hwCameraStatus  OBJECT-TYPE
       SYNTAX     INTEGER (0|1)
       ACCESS     read-write
       STATUS     mandatory
       DESCRIPTION
              "The Network Status:

              0 - OFF
              1 - ON"
       ::= { hardware 6 }

sysModelNumber  OBJECT-TYPE
       SYNTAX     OCTET STRING
       ACCESS     read-only
       STATUS     mandatory
       DESCRIPTION
              "Device Model Number"
       ::= { system 1 }

sysAndroidVersion  OBJECT-TYPE
       SYNTAX     OCTET STRING
       ACCESS     read-only
       STATUS     mandatory
       DESCRIPTION
              "Device Android Version"
       ::= { system 2 }

sysUpTime  OBJECT-TYPE
       SYNTAX     TimeTicks
       ACCESS     read-only
```

```
        STATUS     mandatory
        DESCRIPTION
              "Device Up time since last charge"
        ::= { system 3 }

mngMessage  OBJECT-TYPE
        SYNTAX     OCTET STRING
        ACCESS     read-write
        STATUS     mandatory
        DESCRIPTION
              "Some message from manager"
        ::= { manager 1 }

END

-- This MIB was created using NuDesign Team's Visual MIBuilder
(Ver 4.9).
```

Con esta primera lectura, y leyendo los comentarios del autor, ya podemos ver qué objetos son de utilidad. Además, este primer contacto con la MIB ya ofrece una interpretación de aquellos valores que podrían ser de interés. Por ejemplo, algunos flags de estado. Éste sería el caso del objeto *hwBluetoothStatus*, que ya nos indica en su descripción cómo interpretar los valores obtenidos en una lectura vía SNMP (siendo un "1" cuando el Bluetooth está encendido, y un "0" cuando éste se encuentre apagado). El fragmento asociado a esto, sería:

```
hwBluetoothStatus  OBJECT-TYPE
        SYNTAX     INTEGER (0|1)
        ACCESS     read-only
        STATUS     mandatory
        DESCRIPTION
              "The Bluetooth Status:
              0 - OFF
              1 - ON"
        ::= { hardware 4 }
```

Esta tarea puede resultar algo más compleja y minuciosa con una MIB cuyo contenido sea muy extenso, por lo que, otra forma de hacerlo es mediante algún software que permita mostrar gráficamente la jerarquía de valores tabulares y escalares que conforman el árbol.

Para este propósito, se puede hacer uso de herramientas como *iReasonning MIB Browser* (en su versión gratuita permitiría explorar hasta diez MIBs simultáneas). Este software ofrece la posibilidad de leer e interpretar estos ficheros, a la par que permite realizar operaciones vía SNMP con dispositivos que integren estas capacidades y que se encuentren accesibles desde la red donde se encuenta el equipo que ejecuta el programa. Una vez instalado, y habiendo cargado la MIB en el programa, el panel izquierdo nos mostraría la jerarquía de elementos, como se puede apreciar en la Figura 20.

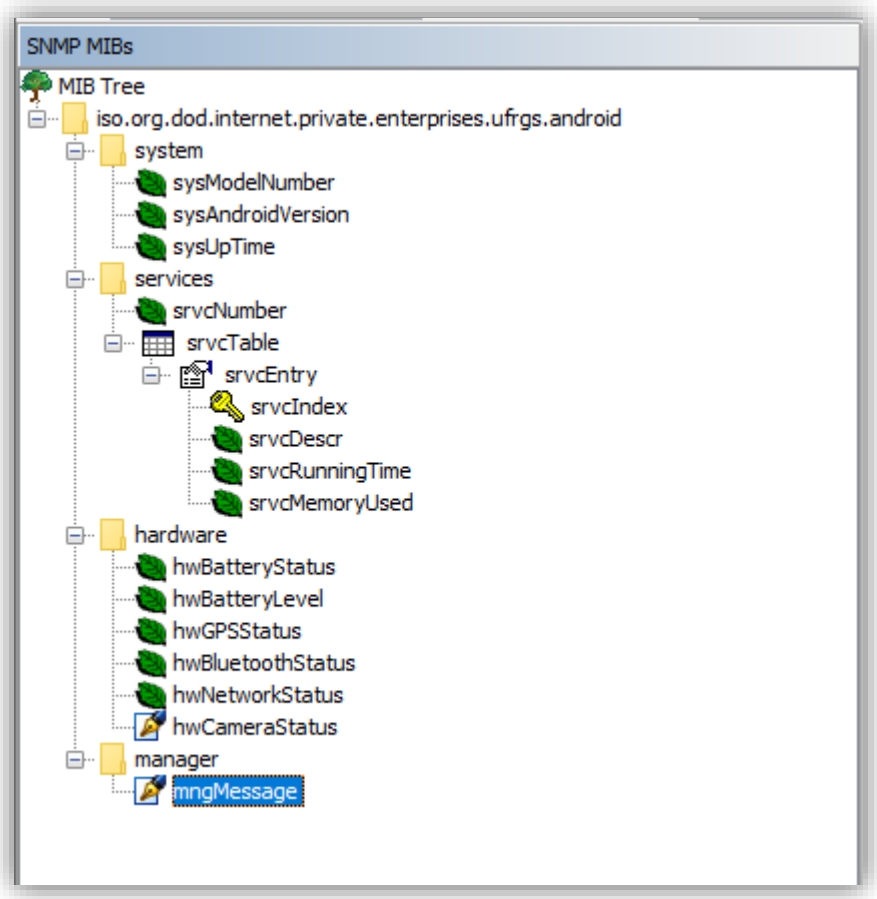

Figura 20. Exploración de la MIB ANDROID-MIB.

En la Figura 20, se puede apreciar que los elementos finales, aquellos representados con forma de hoja guardando la similitud con las hojas de los árboles, serían aquellos susceptibles de ser consultados. También se pueden ver elementos que contienen un icono con forma de pluma, que son aquellos que mediante los permisos adecuados podrían ser modificados a través de una PDU del tipo SETREQUEST. Además de esos elementos, se pueden identificar una tabla que contiene un total de tres columnas que, según su descripción harían referencia a los servicios en funcionamiento dentro del terminal.

Instalada la aplicación del agente SNMP en un terminal con Android, se puede empezar a experimentar en la obtención de valores con este programa. En primer lugar, se debe tener conocimiento de la IP que tiene el terminal dentro de la red, además de leer la documentación del desarrollador con el fin de identificar los parámetros de la cadena de conexión. En el supuesto actual, dicha documentación especifica que la comunidad de lectura es *public*, el puerto es el 32150 y la versión del agente SNMP es la 1. Por lo tanto, la ventana de ajustes avanzados quedaría como se muestra en la Figura 21.

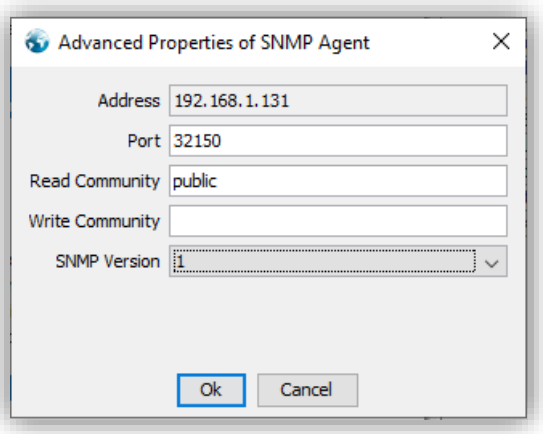

Figura 21. Ajustes de conexión a un terminal Android con MIB Browser.

Una vez establecidos los parámetros de conexión, seleccionando *"Get Subtree"*, se obtienen los valores de todos los objetos por debajo de la rama escogida, representados en una tabla. Si esa rama es la raíz, mostrará todos ellos. Para este primer dispositivo y su integración en Icinga, se hará uso de todos ellos, a excepción de la tabla y los valores de escritura, es decir, aquellos que se encuentran en la Tabla 6.

Objeto SNMP	**Descripción**
sysModelNumber	Devuelve el modelo del dispositivo móvil.
sysAndroidVersion	La versión de *Android* en funcionamiento actualmente.
sysUpTime	El tiempo transcurrido desde el encendido del dispositivo.
srvcNumber	Número de servicios en ejecución actualmente.
hwBatteryStatus	Flag que indica si la batería se encuentra en un estado óptimo.
hwBatteryLevel	Nivel de batería restante. Representado de forma porcentual.
hwGPSStatus	Flag de estado de activación del módulo GPS.
hwBluetoothStatus	Flag de estado de activación del módulo Bluetooth.
hwNetworkStatus	Flag de estado de activación de la interfaz de red.

Tabla 6. Objetos a monitorizar en un terminal Android.

Si se consultan los valores de cada OID escogido, desde MIB Browser con *Get Subtree*, interrogando al host dado de alta en la Figura 21, se obtiene lo mostrado en la Figura 22, obtenida a través de *MIB Browser*.

Name/OID	Value	Type	IP:Port
sysModelNumber.0	LGE Nexus 5	OctetString	192.168.1.131:32150
hwBatteryLevel.0	83	Integer	192.168.1.131:32150
sysUpTime.0	7 hours 8 minutes 39 seconds (2571915)	TimeTicks	192.168.1.131:32150
sysAndroidVersion.0	6.0.1	OctetString	192.168.1.131:32150
srvcNumber.0	51	Integer	192.168.1.131:32150
hwGPSStatus.0	1	Integer	192.168.1.131:32150
hwNetworkStatus.0	1	Integer	192.168.1.131:32150
hwBatteryStatus.0	0	Integer	192.168.1.131:32150
hwBluetoothStatu...	0	Integer	192.168.1.131:32150

Figura 22. Tabla de estados de los objetos escogidos a través de MIB Browser.

Otra de las ventajas de hacer uso de este tipo de programas, como se comentó en el apartado 3.2.3 es que, al recorrer el árbol de objetos SNMP, son capaces de construir el OID necesario para su comprobación. Si se toman como referencia los elementos de la Tabla 6, podemos establecer la correspondencia de objetos de la MIB con respecto a sus OID mostrada en la Tabla 7.

Objeto SNMP	**OID**
sysModelNumber	.1.3.6.1.4.1.12619.1.1.1
sysAndroidVersion	.1.3.6.1.4.1.12619.1.1.2.0
sysUpTime	.1.3.6.1.4.1.12619.1.1.3.0
srvcNumber	.1.3.6.1.4.1.12619.1.2.1.0
hwBatteryStatus	.1.3.6.1.4.1.12619.1.3.1.0
hwBatteryLevel	.1.3.6.1.4.1.12619.1.3.2.0
hwGPSStatus	.1.3.6.1.4.1.12619.1.3.3.0
hwBluetoothStatus	.1.3.6.1.4.1.12619.1.3.4.0
hwNetworkStatus	.1.3.6.1.4.1.12619.1.3.5.0

Tabla 7. Correspondencia de objetos SNMP de la MIB ANDROID-MIB con sus OID.

Una vez obtenidos los parámetros de conexión al *host*, así como el conjunto de OIDs necesarios, es hora de especificar esta configuración en Icinga. No obstante, antes de comenzar con esta parte, es altamente recomendable que el servidor donde se encuentra Icinga cuente con NET-SNMP para poder realizar pruebas rápidas de conectividad y escucha. Para intalar este paquete de herramientas, habría que dirigirse al términal y únicamente ejecutar el siguiente comando:

```
$ sudo apt-get install snmp
```

Una vez termina la instalación del paquete y sus dependencias, estarán disponibles todas las herramientas (descritas en el apartado 2.4.5). Esto es bastante aconsejable para poder realizar algunas pruebas de conexión antes de ponerse a codificar los ficheros de configuración, a fin de descartar posibles dificultades en la comunicación. Una de las herramientas del paquete recién instalado que más información arrojan y que podría haber sustituido al uso de *MIB Browser* en este supuesto, sería *snmpwalk*. Esta herramienta va iterando OIDs, mediante la PDU: GETNEXTREQUEST. Para llevar a cabo esta prueba, se va a iterar sobre los elementos escalares situados en la rama *system* con el siguiente comando:

```
snmpwalk -v1 -c public 192.168.1.131:32150 .1.3.6.1.4.1.12619.1.1
```

Siendo `.1.3.6.1.4.1.12619.1.1` el OID del objeto tabular asociado a *system* dentro de la MIB. En caso de que la conexión sea satisfactoria y los parámetros sean los correctos, este comando devuelve el estado de todos los valores de los registros situados bajo el OID tabular (*system*), de la siguiente forma:

```
iso.3.6.1.4.1.12619.1.1.1.0 = STRING: "LGE Nexus 5"
iso.3.6.1.4.1.12619.1.1.2.0 = STRING: "6.0.1"
iso.3.6.1.4.1.12619.1.1.3.0 = Timeticks: (9059050) 1 day, 1:09:50.50
```

Como puede comprobarse, los OIDs recorridos (situados en el lado izquierdo del resultado), corresponden con los OIDs explorados anteriormente a través de la lectura de la MIB y de la interpretación con MIB Browser. Una vez se configure el resto del entorno, lo que Icinga hará internamente es llamar al comando *snmpget* para recoger de forma individual cada uno de estos valores.

Ahora sí, el paso siguiente sería editar los ficheros de configuración de icinga2 para añadir los elementos que hacen posible la monitorización.

4.4.2 Creación de un *command* de Icinga

Como se ha explicado en el inicio de este capítulo, una de las piezas fundamentales para formular la solicitud SNMP es la creación de un *command*. En el apartado anterior se han recogido los valores de los OID de los que se desea hacer uso (representados en la Figura 19 por los elementos *Dato1*, *Dato2* y *DatoN*). También se cuenta ya con el *plugin* que se ha de utilizar para realizar la llamada SNMP, por lo que, el siguiente paso consiste en crear un *command* que recoja los

parámetros de cada solicitud que se va a enviar a través de la ejecución de un *plugin*. En este supuesto, el *plugin* a utilizar se encuentra por defecto en */usr/lib/nagios/plugins/* y tiene el nombre de *check_snmp*. En el caso de ser ejecutado sin ningún parámetro, tal y como se muestra a continuación, devuelve lo siguiente:

```
/usr/lib/nagios/plugins# ./check_snmp

check_snmp: Could not parse arguments
Usage:
check_snmp -H <ip_address> -o <OID> [-w warn_range] [-c crit_range]
[-C community] [-s string] [-r regex] [-R regexi] [-t timeout] [-e retries]
[-l label] [-u units] [-p port-number] [-d delimiter] [-D output-delimiter]
[-m miblist] [-P snmp version] [-N context] [-L seclevel] [-U secname]
[-a authproto] [-A authpasswd] [-x privproto] [-X privpasswd] [-4|6]
```

Tras esta salida, ya pueden determinarse los parámetros con los que habría que llamar al *plugin* para que devuelva los datos asociados a un OID. En el supuesto planteado en esta memoria serían los siguientes:

- La IP del agente SNMP (-H)
- La comunidad de seguridad (-C)
- La versión de SNMP (-P)
- El puerto de escucha del agente (-p)
- El OID al que se hace referencia (-o)

Es importante señalar que la configuración de cualquiera de los elementos asociados a icinga2 podría estar ubicada en cualquiera de los ficheros de configuración situados en */etc/icinga2/conf.d/*), con independencia del nombre que tengan. Sin embargo, se recomienda mantener cierto orden para evitar confusiones y hacer más mantenible la plataforma. Con este fin, es recomendable posicionarse en el directorio donde se localiza toda la configuración mencionada y editar el fichero *commands.conf*. En éste, se pueden encontrar de manera predefinida algunos *commands* que hacen referencia a las notificaciones, por lo que habría que anexar a su contenido el comando SNMP que se vaya a utilizar. Una vez codificada la elaboración del mismo, tendría la siguiente apariencia:

```
object CheckCommand "check_snmp" {
        import "plugin-check-command"
        command = [ PluginDir + "/check_snmp" ]
```

```
        arguments = {
                "-H" = "$snmp_host$"
                "-o" = "$snmp_oid$"
                "-C" = "$snmp_com$"
                "-P" = "$snmp_vers$"
                "-p" = "$snmp_puerto$"
        }
        vars.snmp_host = "$snmp_host$"
        vars.snmp_oid = "$snmp_oid$"
        vars.snmp_com = "$snmp_com$"
        vars.snmp_vers = "$snmp_vers$"
        vars.snmp_puerto = "$snmp_puerto$"
}
```

Como se puede apreciar en el fragmento de código anterior, se declara el objeto de tipo *CheckCommand* llamado "*check_snmp*". A continuación, se importa la configuración por defecto de los *plugins* de monitorización y se indica que el fichero ejecutable está en el directorio por defecto (*/usr/lib/nagios/plugins*), aunque en lugar de especificar la ruta se utiliza la variable interna de icinga *PluginDir*. Por último, se indican los parámetros que se le pasarán a este *command* y el nombre que recibirán los parámetros en el resto de los ficheros de configuración cuando se haga referencia a ellos:

```
vars.snmp_host
vars.snmp_oid
vars.snmp_com
vars.snmp_vers
vars.snmp_puerto
```

4.4.3 Grupos

Una buena práctica para evitar código redundante y hacer más sencillo el mantenimiento de la configuración es la creación de grupos. Estos grupos permiten asociar objetos que tengan algo en común entre sí, en base a un parámetro configurable en el resto de los elementos. En la configuración de icinga2 existen dos tipos de grupos:

- ***Hostgroups***: permiten configurar ajustes en base a un patrón que todos los *hosts* tengan en común, en lugar de tener que realizar dichos ajustes de manera individual. Por defecto,

se pueden encontrar en la configuración algunos grupos predefinidos de la siguiente manera:

```
object HostGroup "linux-servers" {
  display_name = "Linux Servers"
  assign where host.vars.os == "Linux"
}
object HostGroup "windows-servers" {
  display_name = "Windows Servers"
  assign where host.vars.os == "Windows"
}
```

Una configuración de este estilo permite posteriormente, por ejemplo, asignar una serie de servicios a un grupo de *hosts* que incluyan en su configuración una variable (*vars.os*) cuyo contenido sea el mismo (en este caso, "Linux" o "Windows"). En la configuración que se está estableciendo para este caso práctico, uno de los aspectos que tendrían en común todos los *hosts* serían los elementos de la Tabla 7. De este modo, cualquier dispositivo móvil que pertenezca al grupo de los *móviles* verá reflejada la comprobación de estas variables. Por lo tanto, se ha creado un *Hostgroup* de la siguiente manera:

```
object HostGroup "moviles"{
  display_name = "moviles"
  assign where host.vars.grupo == "moviles"
}
```

Indicando así, que cualquier *host* que contenga la variable *vars.grupo* con valor "moviles" pertenecerá a este nuevo grupo.

- ***Servicegroups:*** su funcionalidad es aglutinar servicios que tengan algo común. Esta característica ayuda a identificarlos una vez están representados en la web. De forma predeterminada, se pueden encontrar algunos ejemplos en la configuración, a modo de sugerencia, que se exponen aquí:

```
object ServiceGroup "ping" {
  display_name = "Ping Checks"
  assign where match("ping*", service.name)
}
```

```
object ServiceGroup "http" {
  display_name = "HTTP Checks"
  assign where match("http*", service.check_command)
}

object ServiceGroup "disk" {
  display_name = "Disk Checks"
  assign where match("disk*", service.check_command)
}
```

Como se puede apreciar, se ofrecen dos metodologías (no son las únicas). Por un lado, se sugiere que todos los servicios cuyo nombre comience por "ping" puedan ser agrupados desde la interfaz web con el fin de ser encontrados más fácilmente (por ejemplo, a través de un *dashboard*). Esto puede resultar útil si, por ejemplo, en una situación en la que hubiera servicios cuyos datos provienen de diferentes protocolos, los procedentes de uno de ellos pudieran ser más fácilmente identificados. En este supuesto, serían los servicios cuyo nombre fuera *snmp_xxxxx* (*snmp_mem*, *snmp_cpu*, etc.).

Por otro lado, el ejemplo anterior también ofrece agrupación de servicios por tipo de *command* empleado, que perfectamente podría ser el creado en el apartado anterior. Dado que en el ejemplo actual no se cuenta con una diversidad significativa de servicios, no se incorpora esta última funcionalidad.

4.4.4 Templates

Una vez más, con el fin de evitar la duplicidad del código, además de simplificar la estructura de ficheros generada para la configuración, se nos ofrece la posibilidad de crear plantillas en las que se podrán determinar configuraciones que vayan a ser comunes a algunos elementos. Esto es algo que debería decidirse durante el diseño inicial de la monitorización y que simplificará posteriormente la agregación de nuevos elementos a esta configuración.

Dichas plantillas existen nuevamente aplicables a *hosts* y *services*. En ambos casos, pueden resultar útiles para unificar parámetros como, por ejemplo:

- Frecuencia de comprobación.
- Número de reintentos.
- Tiempo de espera para reintentos.

- Valores que sean comunes a dispositivos de un grupo como, por ejemplo:
    - La comunidad SNMP.
    - La versión.
    - El puerto
    - Credenciales genéricas, si las hubiera.
    - Etc.

Es por esto que, en el caso del supuesto actual en el que se están configurando *smartphones*, se ha creado una nueva plantilla (*template*), llamada *móviles*, y que responde a la siguiente configuración:

```
template Host "moviles" {
  max_check_attempts = 3
  check_interval = 5m
  retry_interval = 5m
  check_command = "hostalive"
  vars.snmp_com = "public"
  vars.snmp_vers = "1"
  vars.snmp_puerto = "32150"
}
```

Como puede apreciarse en la definición anterior, se especifican los siguientes parámetros para aquellos *hosts* de la tipología "móviles":

- `max_check_attempts:` el número máximo de reintentos para comprobar el estado del *host* (no hay que confundir éste con el estado de cada servicio) en caso de que no responda de manera adecuada. Para este supuesto, una cantidad de 3 reintentos.

- `check_interval:` el tiempo de espera entre cada comprobación satisfactoria del *host*. Configurado actualmente en cinco minutos.

- `retry_interval`: el tiempo de espera entre cada reintento (sólo entra en funcionamiento cuando se produce un cambio de estado). Configurado actualmente en cinco minutos.

- `check_command:` este atributo permite definir a través de qué comando se quiere determinar si el *host* está operativo o no, pues existen varios tipos de *plugins* que podrían prestar este servicio. Para este supuesto se ha escogido "*hostalive*", que internamente hace uso del *ping* (a través del plugin *check_ping* del paquete *monitoring plugins*), y al

cual se le podrían especificar adicionalmente más atributos (como el número de paquetes o el *timeout*). Para este supuesto, se ha escogido *hostalive* en lugar de, por ejemplo, *check_snmp* ya que, UDP, al ser un protocolo no orientado a la conexión, no otorga demasiada fiabilidad como para determinar si el *host* está disponible. Además, no se considera recomendable que todas las comprobaciones se hagan a través de una sola estrategia, dado que podría sesgar la visión en caso de que, por ejemplo, algún elemento de red filtrase por error el tráfico UDP a través de los *firewalls*. Por lo tanto, en el caso hipotético que se plantea en esta memoria, si el *host* devuelve el *ping*, se considera que está vivo.

- Valores SNMP: dado que todos los *smartphones* que se van a configurar en Icinga hacen uso del mismo agente SNMP, todos ellos cuentan con los mismos parámetros de conexión:
    - Comunidad
    - Versión
    - Puerto

Una vez se den de alta los *hosts*, se debe importar la configuración de este *template* para que Icinga cargue la misma configuración en todos los destinos especificados que cumplan la condición de implementar esta plantilla.

Si, por ejemplo, hubiera *hosts* que escuchasen por otro puerto, habría dos opciones: 1) definir el puerto concreto de escucha en cada *host* que se dé de alta; 2) generar varios *templates* en función de la variedad. Si se diera una variedad tal que el que el número de puertos diferentes de escucha se acercase al número de *hosts*, lo más correcto sería renunciar a que dicho parámetro formase parte de plantilla, teniendo que ser especificarlo como un atributo más, perteneciente a la propia definición de cada *host* individualmente.

4.4.5 Services

En este apartado se van a definir los servicios que se van a comprobar en cada uno de los *hosts*, por el momento aquellos pertenecientes a los *móviles*. No se ha generado una plantilla previa para los servicios ya que, en este caso, no ofrece una ventaja clara; es bastante común que los servicios se encuentren configurados con una mayor granularidad en función de factores como la criticidad de los mismos. Esto hace que parámetros como la frecuencia de comprobación cambien con respecto a otros servicios y no se pueda realizar una plantilla común a todos ellos.

En el supuesto actual, en el cual se comprueba, por ejemplo, el nombre del dispositivo a través de uno de los OIDs y, además la batería restante con otro OID diferente, puede que no sea relevante comprobar el nombre del dispositivo cada 5 minutos, pero sí que podría ser decisivo comprobar la batería con una frecuencia mayor, a fin de determinar rápidamente un problema de escasez de alimentación. Es por esto que, para los servicios enumerados en la Tabla 7, se propone la siguiente definición de *services*:

```
apply Service "NombreMovil"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "moviles"
  vars.snmp_oid = "1.3.6.1.4.1.12619.1.1.1.0"
  check_interval=100m
  retry_interval=10m
  max_check_attempts=3
}

apply Service "VersionAndroid"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "moviles"
  vars.snmp_oid = "1.3.6.1.4.1.12619.1.1.2.0"
  check_interval=100m
  retry_interval=10m
  max_check_attempts=3
}

apply Service "Uptime"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "moviles"
  vars.snmp_oid = "1.3.6.1.4.1.12619.1.1.3.0"
  check_interval=10m
  retry_interval=5m
  max_check_attempts=3
}

apply Service "MovilCargando"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "moviles"
  vars.snmp_oid = "1.3.6.1.4.1.12619.1.3.1.0"
  check_interval=5m
  retry_interval=1m
  max_check_attempts=3
}

apply Service "BateriaRestante"{
```

```
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "moviles"
  vars.snmp_oid = "1.3.6.1.4.1.12619.1.3.2.0"
  check_interval=2m
  retry_interval=1m
  max_check_attempts=3
}

apply Service "GPSEncendido"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "moviles"
  vars.snmp_oid = "1.3.6.1.4.1.12619.1.3.3.0"
  check_interval=5m
  retry_interval=3m
  max_check_attempts=3
}

apply Service "BluetoothEncendido"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "moviles"
  vars.snmp_oid = "1.3.6.1.4.1.12619.1.3.4.0"
  check_interval=5m
  retry_interval=3m
  max_check_attempts=3
}

apply Service "AdaptadorRedLevantado"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "moviles"
  vars.snmp_oid = ".1.3.6.1.4.1.12619.1.3.5.0"
  check_interval=10m
  retry_interval=1m
  max_check_attempts=3
}

apply Service "NumeroServicios"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "moviles"
  vars.snmp_oid = ".1.3.6.1.4.1.12619.1.2.1.0"
  check_interval=10m
  retry_interval=1m
  max_check_attempts=3
}
```

Como puede observarse, para cada uno de los servicios, se especifican atributos que son variables en función de una supuesta escala de prioridades. Nótese que en el caso de los servicios, la

definición de *check_command* hace referencia al comando definido en el apartado 4.4.3 y que cada servicio se asigna al grupo de los móviles a través de la comprobación realizada con la siguiente condición:

```
assign where host.vars.grupo == "moviles"
```

Además de los parámetros variables en función de la criticidad, en la definición de cada uno de los *services* se puede apreciar el que nos permite acceder al registro concreto donde se encuentra la información, el OID.

4.4.6 Hosts

Por último, lo único que quedaría para completar el proceso de alta de nuevos elementos en Icinga es el *host* como tal. Una vez definidos el resto de los objetos vistos en los apartados anteriores, la definición de *hosts* es algo bastante sencillo. Además, la casuística más típica de cualquier entorno es la siguiente: alta de equipos nuevos y baja de aquellos que ya han terminado su ciclo de vida.

Para el caso planteado, la definición de un *host* se codificaría de la siguiente manera:

```
object Host "Nexus5" {
        display_name = "Google Nexus 5"
        import "moviles"
        address = "192.168.1.131"
        vars.snmp_host = "192.168.1.131"
        vars.grupo = "moviles"
}
```

En una hipotética situación en la que se desease añadir más objetos del tipo *host*, bastaría con añadir este segmento de código variando el atributo de la IP, así como el nombre con el que se desea reconocer el objeto. En el caso expuesto aquí, podemos ver que la definición de la IP aparece en dos ocasiones, ya que el *check_command* del *host* (*ping*) espera encontrar ese dato en la variable *address*, mientras que el *check_snmp* espera hacerlo en el campo *vars.snmp_host*.

Por último, el host queda incluido en el grupo “móviles” (definido en el apartado 4.4.3), mediante la asignación: `vars.grupo = "moviles".` Y la plantilla (generada en el apartado 4.4.4) queda importada para cargar la configuración a través de: `import "moviles".`

Una vez todos los parámetros del *host* y sus *services* se cargan en Icinga, éste los va utilizando, según la planificación, a través del comando *check_snmp*, definido en el apartado 4.4.2, para realizar las pertinentes solicitudes SNMP.

4.5 Obtención de datos de monitorización

Una vez los elementos a monitorizar están definidos en los ficheros de configuración, el sistema está listo para iniciar la recopilación de los datos. Por lo tanto, para que Icinga reconozca estos cambios, se debe reiniciar o recargar el servicio de icinga2 con el siguiente comando:

```
# systemctl reload icinga2
```

Durante el proceso de recarga o reinicio del servicio, el demonio de Icinga realiza una comprobación de los archivos para verificar que tanto el contenido como la sintaxis es correcta. En el caso de encontrar errores, se avisará al usuario para que revise la configuración. Un comando que puede resultar bastante útil es el siguiente, correspondiente a la invocación manual del compilador de sintaxis. Ello evita forzar la recarga del servicio cada vez que se necesite cambiar la configuración o comprobar que la sintaxis es correcta (también es útil disponer durante este proceso de una salida continua de *syslog* para detectar otro tipo de errores que pudieran producirse):

```
# icinga2 daemon –C
```

En un supuesto en el que la configuración establecida sea correcta, el usuario podría iniciar sesión en la parte web de Icinga y comprobar el estado de los elementos recién dados de alta.

Una vista de lo que se muestra en pantalla tras aplicar la configuración especificada en los apartados anteriores sería lo mostrado en la Figura 23. Como se puede apreciar, en la parte superior se indica el *host* seleccionado y, bajo él, las diez variables asociadas a cada uno de los *services* dados de alta. Nótese que, a pesar de haber dado de alta nueve elementos, hay uno

adicional, el *ping* (sobre IPv4), el cual será crucial para determinar si el *host* entero está operativo o fuera de línea.

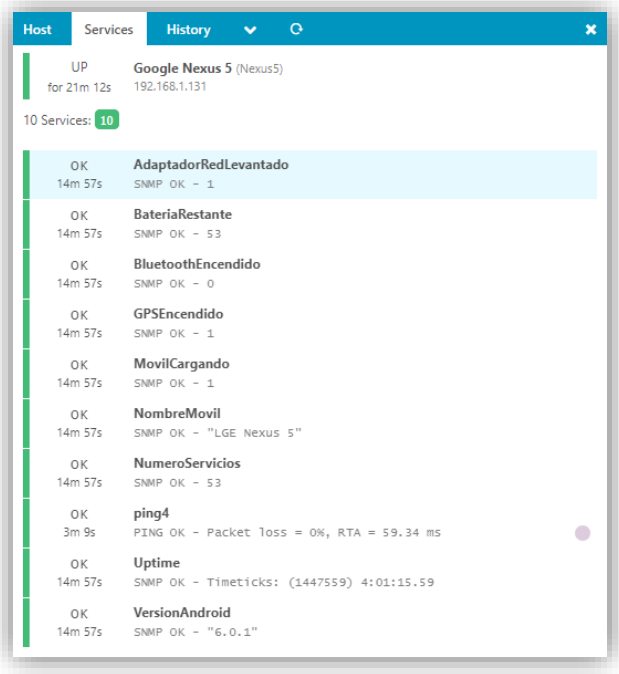

Figura 23. Vista de servicios de un host a través de Icinga Web.

Asimismo, bajo el nombre de cada una de las variables, aparece el dato devuelto por el propio dispositivo, que era uno de los objetivos a alcanzar en el caso de prueba. Si bien los datos devueltos son legibles y comprensibles para un humano, la información podría quedar representada de una forma en la cual fuera más fácilmente interpretable por el personal destinado a la revisión. Habitualmente los operadores de este tipo de plataformas tienen entre sus responsabilidades cantidades mucho más grandes de dispositivos y variables. Por ello, Icinga ofrece posibilidad de categorizar estas señales, dándoles la apariencia de semáforos.

4.5.1 Creación de umbrales para la creación de semáforos.

Como ya se ha comentado ya anteriormente, tanto los *hosts* como los servicios pueden manifestar diferentes estados en función de su situación. Lo que se pretende en este apartado es ampliar la configuración anterior, a fin de que Icinga realice una primera interpretación de los resultados y los clasifique en función de la gravedad. Para ello, es necesario comprender cómo funcionan los estados, y los cambios entre ellos (Icinga Development Team, 2021).

Por su parte los *hosts* tienen una clasificación de estados bastante sencilla. Básicamente o está operativo, o no lo está (Ver Tabla 8).

Estado	Descripción
UP	El *host* está operativo.
DOWN	El *host* no se encuentra disponible.

Tabla 8. Estados posibles de un host.

Sin embargo, los estados de los *services* están más granulados, ya que hay valores intermedios en función de la calidad que ese servicio está prestando (ver Tabla 9).

Estado	Descripción
OK	El servicio está funcionando correctamente.
WARNING	El servicio está experimentando algunos problemas pero aún se considera que está realizando su función correctamente.
CRITICAL	El servicio se encuentra interrumpido.
UNKNOWN	La última comprobación no pudo determinar el estado en que se encuentra el servicio.

Tabla 9. Estados posibles de un service.

Para muchos de los *plugins*, la clasificación del dato puede llevarse a cabo en base a un código de resultado de la operación y, por ende, establecer una correspondencia con los estados dependiendo de si dicho resultado va orientado a ofrecer la situación del *host* o de un *service*, tal y como describe la Tabla 10:

Estado	Estado del host	Estado de un servicio
0	Up	*OK*
1	Up	*Warning*
2	Down	*Critical*
3	Down	*Unknown*

Tabla 10. Correspondencia de código de resultado con los estados.

En este punto, lo único que quedaría por aclarar es la transición entre estados (independientemente de si está asociado al *service* o a un *host*), y aunque puede sonar redundante, esto sería equivalente a definir el "estado del estado".

Supóngase una situación en la que un estado no hubiera cambiado en bastante tiempo. Este estado se encontraría entonces como *hard-state* en Icinga, y por lo tanto, la política de comprobaciones utiliza el parámetro "check_interval" periódicamente. Si, por el contrario, dicho estado se encuentra como *soft-state*, quiere decir que ese cambio de estado se ha producido recientemente, y por tanto, en su lugar, sobre ese elemento se aplicaría el parámetro "retry_interval". Esta es la otra frecuencia establecida en la configuración, utilizada principalmente para verificar más rápidamente si es un suceso puntual o bien, se extiende en el tiempo. En el caso de no ser algo puntual, pasaría de nuevo al *hard-state*, que implica que ese elemento está fallando de manera continuada.

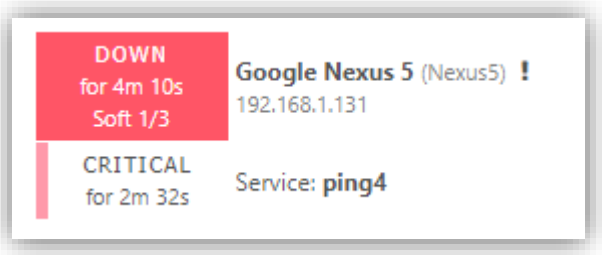

Figura 24. Vista de un soft-state en un servicio.

En la Figura 24 se puede apreciar que se ha perdido la conexión con el equipo vía *ping* recientemente (ya que su *service* se encuentra en *soft-state)*. Icinga acaba de realizar el primero de los 3 reintentos aplicando la frecuencia establecida en el parámetro *retry_interval*.

Una vez definidos estos conceptos, puede procederse con la implementación de umbrales. El *plugin* utilizado en este trabajo para la comprobación de los registros OID de cada equipo (*check_snmp*) también utiliza los códigos representados en la Tabla 10 (de hecho, los scripts personalizados también deben establecer esos códigos de salida para que estos valores puedan ser representados correctamente con semáforos). Sin embargo, Icinga realiza dicha correspondencia internamente en base a unos parámetros recibidos durante la solicitud SNMP. En el apartado 4.4.2, donde se muestra la ayuda del *check_snmp*, se puede apreciar que dos de los parámetros adicionales que se le pueden pasar como referencia son los rangos para los cuales se le devolverá a Icinga uno de los códigos asociados al estado (0, 1, 2 ó 3).

Antes de establecer estos valores conviene aclarar que estos rangos para la creación de umbrales tienen una sintaxis propia heredada de Nagios. Mediante la Tabla 11, con ejemplos, extraída de

la propia documentación oficial (Nagios Enterprises, LLC., 2018), se pretende resumir este tipo de expresiones. La finalidad es facilitar al lector las expresiones que aparecerán más adelante editando la configuración ya realizada a lo largo del apartado 4.4.

Estado	Descripción
10	< 0 o >10, (fuera del rango {0 .. 10})
10:	< 10, (fuera del intervalo {10 .. ∞})
~:10	> 10, (fuera del rango {-∞ .. 10})
10:20	< 10 o > 20, (fuera del rango entre {10 .. 20})
@10:20	≥ 10 and ≤ 20, (dentro del rango {10 .. 20})

Tabla 11. Ejemplos de expresiones en Nagios para la estipulación de umbrales.

Una vez indicado cómo expresar los umbrales, es necesario realizar modificaciones en dos de los ficheros que se han configurado en los pasos anteriores.

Por un lado, está la modificación del *command* codificado en el apartado 4.4.2, ya que, añadir únicamente los valores de los umbrales en cada *service* es inútil, dado que el *command* no espera recibir esos parámetros. Así pues, se debe editar la configuración para que los acepte. Esto puede codificarse de la siguiente manera (las líneas añadidas están marcadas en negrita):

```
object CheckCommand "check_snmp" {
        import "plugin-check-command"
        command = [ PluginDir + "/check_snmp" ]
        arguments = {
                "-H" = "$snmp_host$"
                "-o" = "$snmp_oid$"
                "-C" = "$snmp_com$"
                "-P" = "$snmp_vers$"
                "-w" = "$snmp_warning$"
                "-c" = "$snmp_critical$"
                "-p" = "$snmp_puerto$"
        }
        vars.snmp_host = "$snmp_host$"
        vars.snmp_oid = "$snmp_oid$"
        vars.snmp_com = "$snmp_com$"
        vars.snmp_vers = "$snmp_vers$"
        vars.snmp_warning = "$snmp_warning$"
```

```
        vars.snmp_critical = "$snmp_critical$"
        vars.snmp_puerto = "$snmp_puerto$"
}
```

Los campos "-w" y "-c" son propios de este *plugin*. Al igual que en la situación anterior, dichos campos pueden ser encontrados en la ayuda cuando se invoca sin ningún parámetro.

Tras realizar el paso anterior, conviene establecer los rangos de valores para los cuales se espera encontrar una separación de umbrales. En el caso práctico presente no se van a aplicar en todos los *services* (ya que algunos de ellos simplemente devuelven datos informativos tales como el nombre del dispositivo o la versión de Android). Esto último no implica que estos valores puedan ser de interés para el administrador del entorno. Por tanto, se ha decidido establecer los siguientes valores hipotéticos de corte para los servicios (ver Tabla 12).

Objeto SNMP	Nombre Servicio Icinga	Umbral Warning	Umbral Critical
sysModelNumber	NombreMovil	--	--
sysAndroidVersion	VersionAndroid	--	--
sysUpTime	Uptime	Si hace menos de un día que se encendió (1 día = 8640000 *timeticks*).	--
srvcNumber	NumeroServicios	Entre 30 y 55	Más de 55
hwBatteryStatus	MovilCargando	Flag distinto de 1	--
hwBatteryLevel	BateriaRestante	Entre 21% y 40%	Entre 0% y 20%
hwGPSStatus	GPSEncendido	--	Flag distinto de 1
hwBluetoothStatus	BluetoohEncendido	--	Flag distinto de 1
hwNetworkStatus	AdaptadorRedLevantado	--	Flag distinto de 1

Tabla 12. Valores hipotéticos para la estipulación de umbrales de un host Android.

Por lo tanto, realizando la traducción de estos rangos al formato Nagios, nos queda un código en el que los datos figuran de la la forma que se indica abajo, marcando nuevamente en negrita aquellas líneas que se añaden a la configuración (con el fin de facilitar la lectura del documento se omiten el resto de líneas ya declaradas y que no influyen en los umbrales):

```
apply Service "NombreMovil"{
        ..
```

```
  vars.snmp_warning = ""
  vars.snmp_critical = ""
}

apply Service "VersionAndroid"{
      ..
  vars.snmp_warning = ""
  vars.snmp_critical = ""
}

apply Service "Uptime"{
      ..
  vars.snmp_warning = "8640000:"
  vars.snmp_critical = ""
}

apply Service "MovilCargando"{
      ..
  vars.snmp_warning = "1:1"
  vars.snmp_critical = ""
}

apply Service "BateriaRestante"{
      ..
  vars.snmp_warning = "@21:40"
  vars.snmp_critical = "@0:20"
}

apply Service "GPSEncendido"{
      ..
  vars.snmp_warning = ""
  vars.snmp_critical = "1:1"
}

apply Service "BluetoothEncendido"{
      ..
  vars.snmp_warning = ""
  vars.snmp_critical = "1:1"
}

apply Service "AdaptadorRedLevantado"{
      ..
  vars.snmp_warning = ""
  vars.snmp_critical = "1:1"
}

apply Service "NumeroServicios"{
      ..
  vars.snmp_warning = "@30:55"
  vars.snmp_critical = "~:55"
}
```

Como se puede apreciar, para aquellos servicios en los que no se hace uso de umbrales es necesario especificar un valor vacío.

Una vez realizadas las modificaciones, habría que cargar esta configuración mediante la recarga o reinicio del servicio de icinga2. Tras esto, el usuario podría dirigirse nuevamente a la web de Icinga y esperar a la siguiente comprobación de los servicios, o bien forzar una consulta manual al *host*. De cualquier forma, y si las condiciones interpuestas a los componentes se cumplen en algún caso, la situación quedaría representada como aparece en la Figura 25.

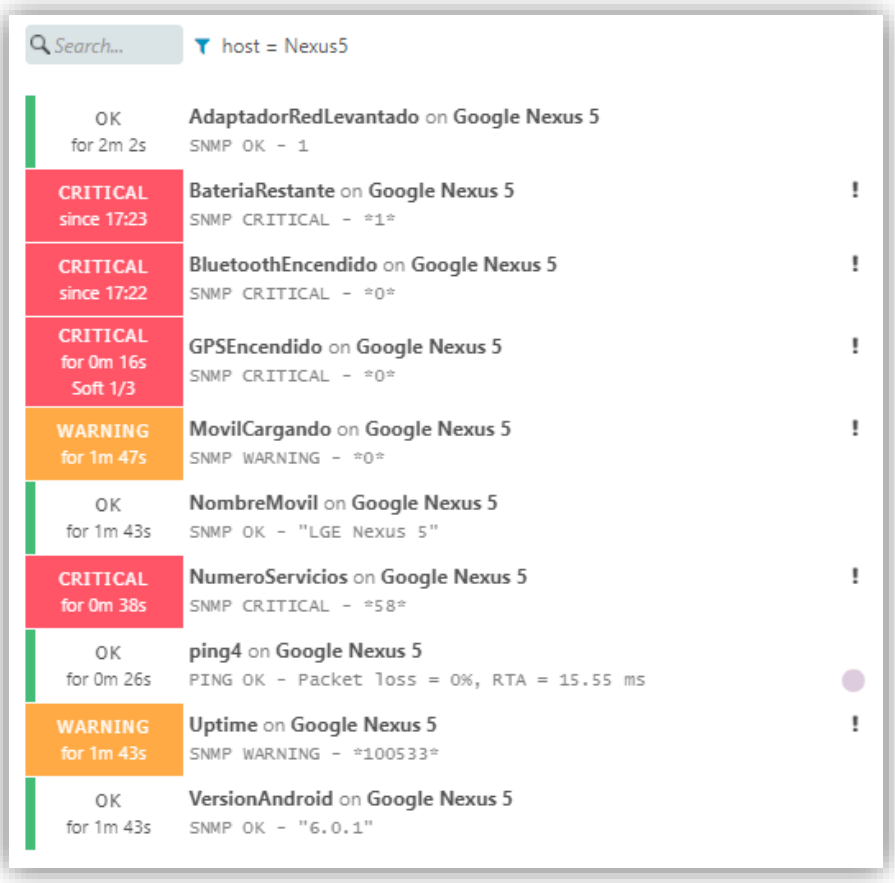

Figura 25. Obtención de datos con umbrales a través de Icinga Web.

Para el caso representado en la Figura 25, se ha propiciado manualmente la situación de alguna de esas variables para que los datos obtenidos muestren los umbrales con los colores representados. Como puede apreciarse, de esta forma es mucho más rápido interpretar la información obtenida a través del navegador, en lugar de tener que comprobar individualmente en cada servicio si los valores son acordes o se salen de la norma.

4.5.2 Ventanas de tiempo

Otra de las circunstancias que pueden darse de manera relativamente frecuente en ámbitos reales es que algunos de los elementos que conforman la infraestructura de monitorización no se encuentren operativos la totalidad del tiempo. Esto conllevaría que los dispositivos remotos o algunos de los valores que se comprueban mediante estos mecanismos, no estén disponibles. La casuística puede darse en cualquiera de las partes, incluyendo aquellos dispositivos que son externos a la monitorización pero que forman parte del entorno.

En este sentido, y desde el punto de vista de la supervisión de dispositivos, podríamos decir que existen dos niveles de corte de servicio: total y parcial. Un ejemplo de la primera categoría podría ser que aquellos elementos de red que conforman el entorno y dan servicio a todo tipo de necesidades (*switches*, *routers*, etc.), tuvieran una franja de tiempo para desplegar parches de seguridad o realizar tareas de mantenimiento. Este tipo de corte es total ya que afecta transversalmente a todos los estratos e impediría tener conocimiento de cualquiera de los dispositivos durante el periodo de tiempo que duren este tipo de intervenciones.

La segunda categoría, cortes parciales, abarcaría aquellas situaciones en las que una intervención no afecta a la visibilidad total. Ejemplos de ello serían un apagado de uno de los *hosts* que se monitorizan, o un apagado de alguno de los sensores que integra, a causa de algún tipo de labor de mantenimiento.

Por último, y esto sería algo determinante, cualquiera de las dos categorías expuestas puede darse de dos maneras: de forma planificada o no planificada. Ésta última modalidad no suele darse en situaciones en las que el elemento impulsor sean labores de mantenimiento, ya que habitualmente suelen estar bien coordinadas para minimizar riesgos.

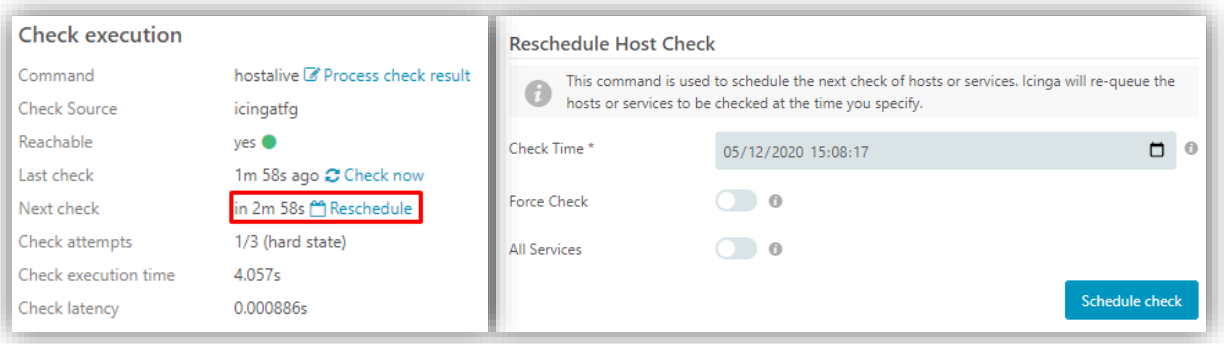

Figura 26. Planificación de la siguiente solicitud SNMP manualmente.

No obstante, la forma no planificada puede existir en ocasiones en las que tengan que ser realizadas por un suceso puntual, como la sustitución de una batería que ha llegado al final de su ciclo de vida. Para esta situación, Icinga incorpora dos funcionalidades que pueden ser interesantes y su gestión es posible desde la propia web (ver Figura 26).

Además de poder lanzar comprobaciones de manera manual, Icinga admite reprogramar las solicitudes (tanto de *hosts* como de *services*), sobrescribiendo la configuración automática especificada en los ficheros de definición. De esta manera, se puede evitar que se interrogue a dispositivos durante un tiempo prudencial en el que se sabe que no va a haber operatividad. Adicionalmente, si no se desea que la plataforma "oculte" la información de ese suceso pero se desea que las notificaciones (en el caso de estar configuradas) no lleguen a las personas responsables de ese elemento para no alarmarlas, se pueden suspender por un periodo de tiempo las comunicaciones asociadas e incluir comentarios que faciliten las revisiones futuras de los históricos (ver Figura 27).

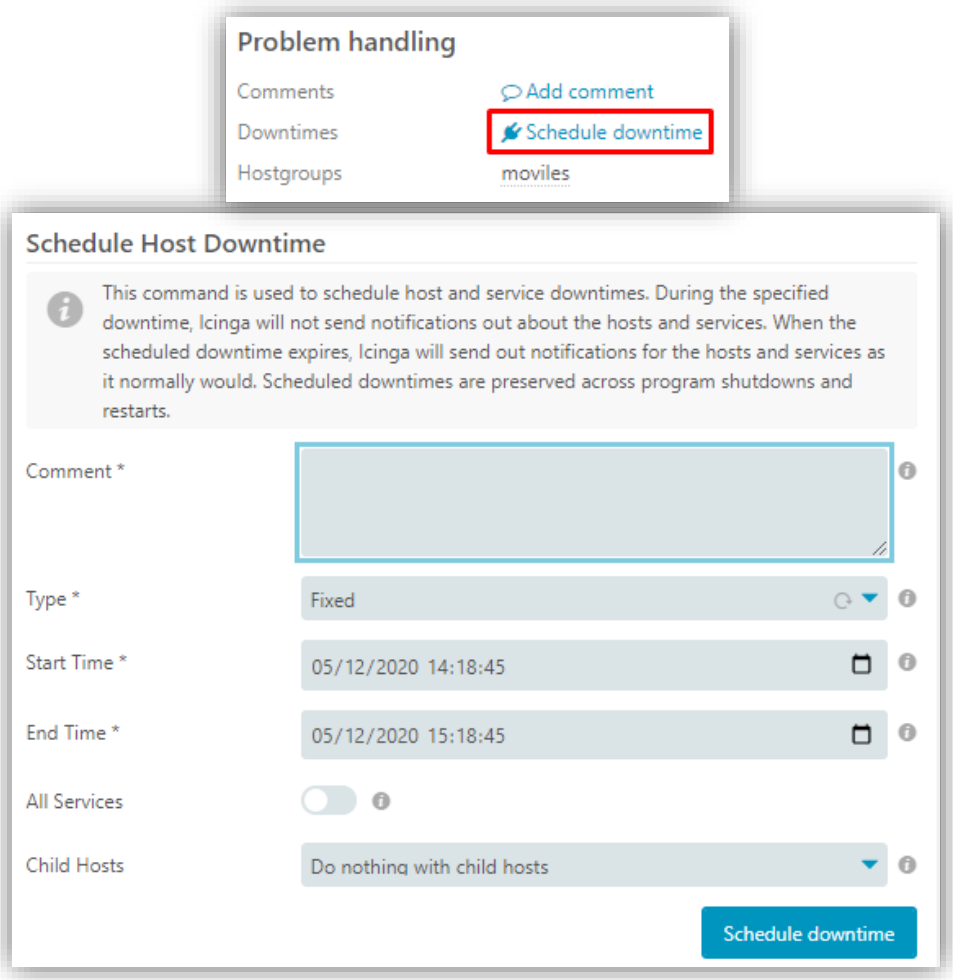

Figura 27. Suspensión de notificaciones y comentarios por intervención de mantenimiento.

Por último, quedarían aquellas situaciones de corte, parcial o total, que integren una programación periódica planificada. En estos supuestos, no tendría sentido que cada día o semana tuvieran que utilizarse las dos estrategias anteriores manualmente para poder manejar este tipo de intervenciones. Para estos casos Icinga incorpora en su configuración interna los periodos de comprobación. Éstos no son más que una planificación temporal en la que se le indica al *service* las horas del día en las que se puede interrogar a los *hosts*.

Los eventos planificados pero no periódicos (como, por ejemplo, un festivo) pueden ser añadidos en la configuración especificando el día del año o del mes en el que tienen lugar. Para poder excluir esas fechas es necesario crear un fragmento de código declarando un objeto de tipo *TimePeriod* en el que figuren las fechas o periodos a excluir. Se recomienda añadir el código referente a esta parte en el fichero *timeperiods.conf* con el fin de mantener la estructura sugerida al principio:

```
object TimePeriod "exclusiones" {
       ranges = {
       "january 1" = "00:00-24:00"
       "december 25" = "00:00-24:00"
       }
}
```

Una vez claras las fechas que desean excluirse y sus horarios, quedaría crear otro objeto del mismo tipo en el que especificar cuándo sí se puede interrogar. Esto se haría de la siguiente manera:

```
object TimePeriod "check_diario" {
   display_name = "Icinga comprobaciones diario"
   excludes = ["exclusiones"]
   ranges = {
    "monday"    = "01:00-23:59"
    "tuesday"   = "01:00-23:59"
    "wednesday" = "01:00-23:59"
    "thursday"  = "01:00-23:59"
    "friday"    = "01:00-23:59"
    "saturday"  = "01:00-23:59"
    "sunday"    = "01:00-23:59"
   }
 }
```

Nótese que en negrita se han especificado el/los módulo(s) (pueden añadirse varios separándolos por comas) cuya planificación va a ser excluida una vez se asigne el objeto *TimePeriod* a algún elemento de la monitorización. Ello puede ser realizado de varias maneras. Una consiste en añadir una línea de asignación en cualquier objeto de la monitorización (*services*, *hosts*, *templates*, *hostgroups*, etc.). En este ejemplo se ha añadido a uno de los *services* dados de alta anteriormente:

```
apply Service "MovilCargando"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "moviles"
  vars.snmp_oid = "1.3.6.1.4.1.12619.1.3.1.0"
  check_interval=10m
  retry_interval=1m
  max_check_attempts=3
  vars.snmp_warning = "1:1"
  vars.snmp_critical = ""
  check_period = "check_diario"
}
```

Una vez recargado el servicio de icinga2, se puede apreciar en la web que el *service* que comprueba si el terminal se encuentra cargando está sujeto a esta planificación recién creada (ver Figura 28).

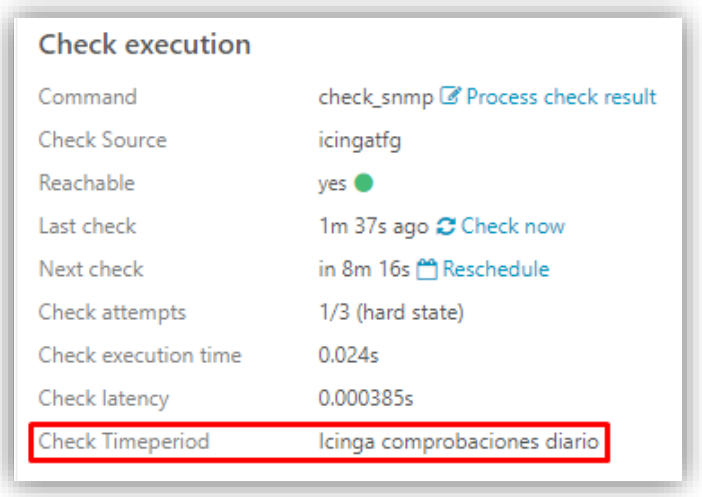

Figura 28. Comprobación de un servicio sujeto a una planificación.

Por lo tanto, este OID será únicamente comprobado periódicamente con la frecuencia configurada en el *service*, de 01:00 a 23:59, todos los días de la semana (dejando libre de peticiones a los *host* que integren en su configuración este *service* en la hora comprendida entre las 24:00 y las 01:00),

a excepción de los días 1 de enero y 25 de diciembre, en los que no se comprobará a ninguna hora.

Una última cuestión a tener en cuenta a este respecto es que Icinga no realiza las comprobaciones de cada elemento en el momento exacto en el que están programadas, sobre todo si la cola de procesamiento del servicio de icinga2 está saturada (puede suceder cuando hay muchos elementos con comprobaciones muy próximas en entre sí). Esto es algo que la documentación de Nagios[15] especifica claramente. Por lo tanto, en el caso de dejar ventanas de ejecución muy pequeñas, hay que considerar que podría no darse la situación deseada durante el diseño de los periodos, quedando quizá algunos servicios sin comprobar o haciéndolo a destiempo (ver Figura 29).

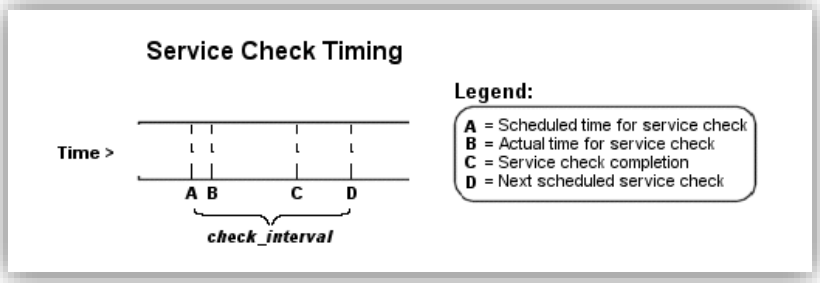

Figura 29. Desvío de planificación en función de la carga de trabajo del servidor (Nagios Enterprises, LLC., 2019).

4.5.3 Creación de Dashboards y filtros

Las funcionalidades de este tipo de herramientas no concluyen con la automatización de la monitorización de los elementos, pues las opciones de representación de la información gráficamente, son también de vital importancia. Cuando la estructura formada por todos los objetos es demasiado grande, estas herramientas son, si cabe, de mayor ayuda porque posibilitan el disponer de resúmenes visuales que vayan clasificando la información obtenida durante el proceso, en función de las necesidades del equipo destinado al control del entorno. Esto último es conocido como filtrar la información, que al igual que otras herramientas (como los *dashboard)* permiten trabajar con cantidades de datos muy grandes, y pueden suponer un gran apoyo.

Los filtros de Icinga admiten parametrizar casi cualquiera de los elementos recogidos en la configuración, con la ventaja de que éstos pueden ser combinados entre sí utilizando puertas lógicas. Todos ellos pueden ser configurados desde la interfaz web, un ejemplo de filtro se muestra en la Figura 30. En este caso, se presenta un filtro en el que se podrían visualizar todos aquellos servicios en estado *WARNING* o *CRITICAL* pertenecientes al grupo de los móviles. Una

[15] https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/checkscheduling.html

vez definido el filtro y pulsando en *Apply*, se mostrará la salida en la parte inferior. A medida que se gana familiaridad con la herramienta, es posible definir estos filtros desde la propia URL, que contiene toda la sintaxis necesaria para la obtención de los datos mostrados en la Figura 30.

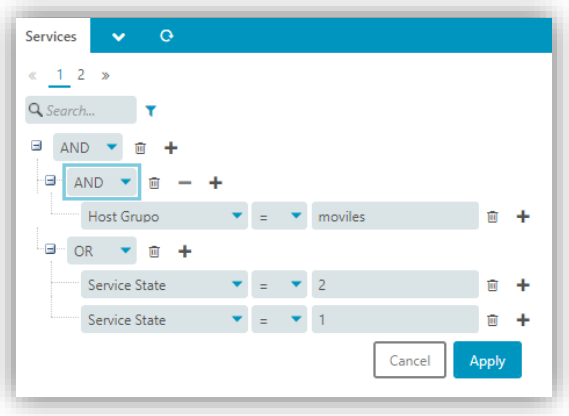

Figura 30. Apariencia de un filtro en Icinga.

Asimismo, un *dashboard* o pizarra, es un lugar en el que visualizar paneles en los que se muestran resúmenes informativos donde tener una vista rápida de los elementos más prioritarios, así como del estado general. La ventaja de esta herramienta es que permite representar cualquiera de los filtros guardados con anterioridad, así como cualquier vista que se haya visitado en los apartados de la web. Esto puede hacerse pulsando el botón con forma de flecha que se encuentra en la parte superior izquierda de la Figura 30 y seleccionando *Add to dashboard*, icono que aparecerá en casi todos los apartados. A cada uno de los paneles de visualización que se encuentran dentro de un *dashboard* se le conoce con el nombre de *dashlet* y pueden ser añadidos tantos como se desee, conviene seleccionar aquellos elementos que sean prioritarios en cada vista (ver Figura 31).

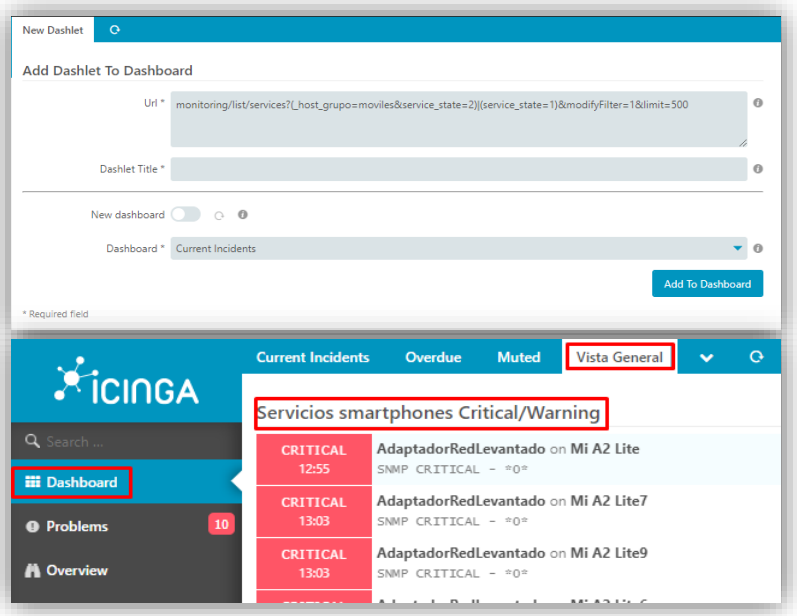

Figura 31. Creación de un dashboard en base a un filtro definido anteriormente.

Icinga también cuenta con paneles predefinidos en los que visualizar la información, los cuales también pueden ser filtrados al gusto, o añadidos en forma de *dashlets* al *dashboard* creado anteriormente. Un ejemplo de ello se presenta en la Figura 32.

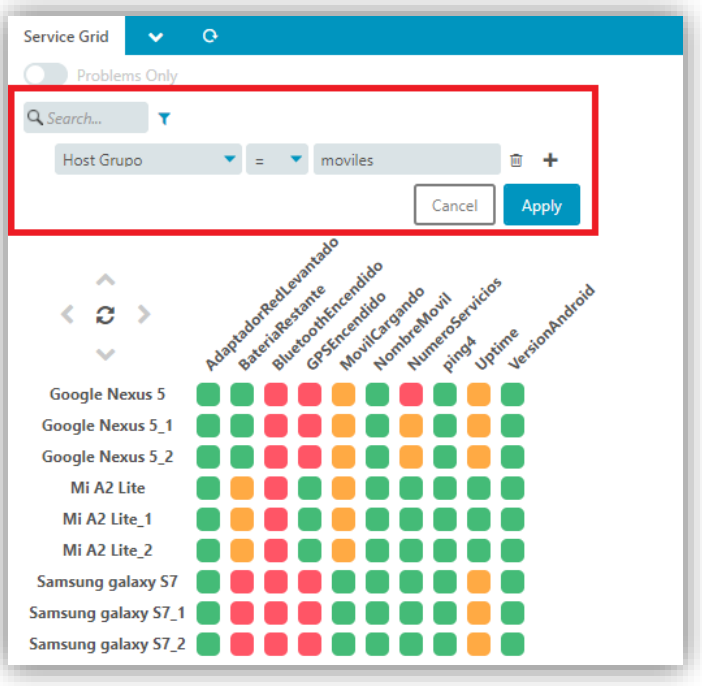

Figura 32. Filtrado de una vista predeterminada de Icinga en un dashboard.

Nuevamente podemos encontrar el icono de la flecha en la parte superior izquierda de la imagen, que nos permitirá añadir la visualización ya filtrada a un *dashboard* de nuestra elección. Para esta captura se han dado de alta los mismos *hosts* varias veces (con nombres distintos), con el fin de mostrar más elementos en una rejilla de *hosts/services*. Integrando este panel, y algunos elementos más en un *dashboard*, obtendríamos una pizarra de resumen similar a la de la Figura 33.

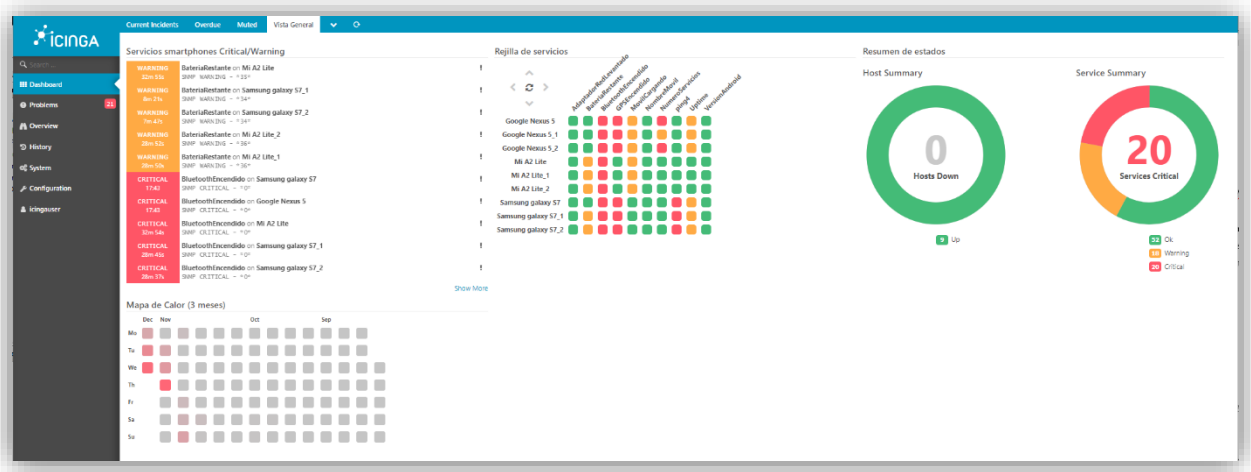

Figura 33. Vista final de un Dashboard con los filtros aplicados anteriormente para el grupo "móviles".

Capítulo 5

Implantación de mejoras al alcance

Este quinto capítulo tiene como propósito, habiendo analizado el proceso descrito anteriormente, encontrar carencias que puedan ser corregidas e integrar nuevas funcionalidades que doten a la plataforma de un valor añadido. Dichas modificaciones pueden estar motivadas por diferentes necesidades pertenecientes cada una de ellas a un ámbito distinto de la monitorización. Por lo tanto, en base a dicho análisis, y teniendo en cuenta las experiencias reales de personal destinado a la implantación y manejo de este tipo de plataformas, se han detectado las siguientes mejoras a aplicar:

- En lo referente al manejo de *plugins*, y habiendo trabajado con SNMP a lo largo del capítulo anterior, se ha detectado una escasa flexibilidad en *check_snmp* para interrogar a los equipos remotos usando distintas estrategias. Asimismo, los datos devueltos por dicho *plugin* no pueden ser tratados de ninguna manera, por lo que la información recuperada no puede ser modificada previamente a su representación a través de la Web de Icinga (por ejemplo: la conversión de unidades, condicionales, representación más amigable, etc.). Por ello, se llevará a cabo el desarrollo de nuevos *plugins* que, haciendo uso de las herramientas proporcionadas por NET-SNMP, cubran esas carencias.

- A pesar de que la obtención de datos vía SNMP sea correcta mediante el *polling*, se ha echado en falta la incorporación de algún mecanismo que permita la recepción y representación de *traps* de notificación en Icinga. Por lo tanto, se llevará a cabo su incorporación haciendo uso de los demonios *snmptrapd*[16] y *snmptt*[17]. Adicionalmente, se incluirá en este apartado el desarrollo de un visor de *traps* que permita consultar y explotar la información obtenida a través de esas notificaciones.

- Desde el punto de vista de la seguridad, se ha considerado que la parte Web de Icinga que funciona sobre HTTP ofrece la representación de datos que podrían ser sensibles dependiendo de la relevancia de los datos y equipos monitorizados. Por lo tanto, se

[16] https://support.nagios.com/kb/article.php?id=88

[17] http://www.snmptt.org/

añadirá al *frontend* de Icinga un juego de certificados autofirmados (generados a través de *OpenSSL*) que permitan cifrar dicho tráfico usando HTTPS.

- Por último, y en materia de alta disponibilidad, se ha determinado que un cometido como el que prestan algunos de los procesos activos en el servidor de Icinga, podrían ser de vital importancia, y contar con una criticidad bastante elevada ya que, en su ausencia se perdería la visibilidad acerca del estado de muchos elementos. Por este motivo, se llevará a cabo la integración del paquete de herramientas *supervisord*[18], a fin de reforzar la ejecución de algunos de esos procesos. La principal motivación de esto es desde el punto de vista de la calidad de la monitorización, ya que ese paquete permite recuperar las pérdidas de servicio puntuales producidas por fallos eventuales derivados de causas reales como saturación de recursos, alta demanda, etc.

5.1 Desarrollo de nuevos *plugins*

Como se ha especificado anteriormente, además del capítulo anterior, el uso de *check_snmp* (un archivo binario incluido en el paquete *monitoring-plugins*, instalado tal y como se indica en el apartado 4.2) hace sencilla la obtención de un dato puntual procedente de equipos remotos vía SNMP; con la creación del *command* asociado, y aportando los parámetros de conexión necesarios, es suficiente para empezar a obtener información de un *host* (o tantos como se desee). Sin embargo, y aunque su desempeño es más que correcto, se echa de menos cierta flexibilidad para realizar otras operaciones que sí que admite el protocolo SNMP.

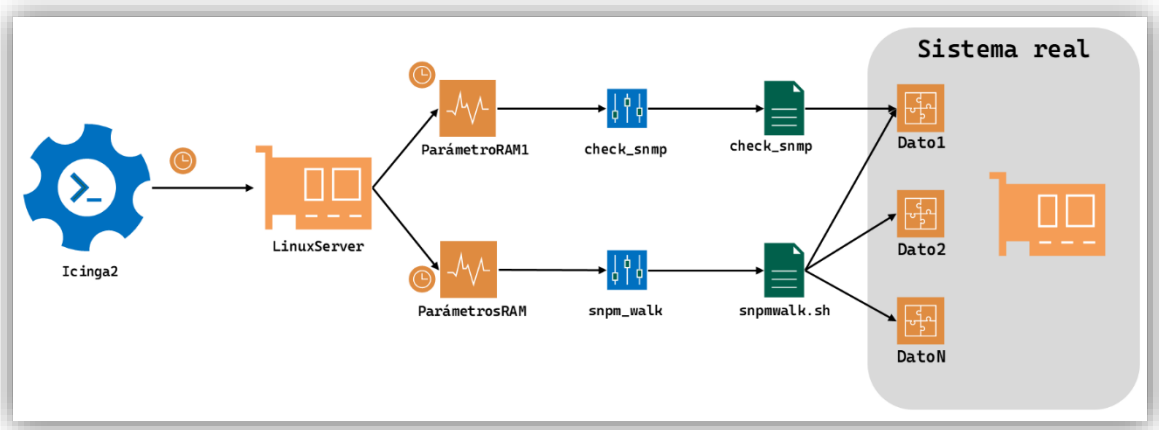

Figura 34. Representación de funcionalidades de plugins. Elaboración propia.

[18] http://supervisord.org/introduction.html

Un requerimiento que podría darse con relativa facilidad es integrar el resultado al utilizar *snmpwalk* en un *service* de Icinga, a fin de evitar la creación de tantos *services* como OID's se quieran recorrer (ver Figura 34).

En la Figura se puede observar cómo, para un host con Linux como Sistema Operativo, y mediante un *service* que utilice el *plugin check_snmp*, sólo podríamos obtener uno de los atributos referentes a la RAM del equipo, haciendo referencia a un único OID. Sin embargo, el desarrollo de un *plugin* que permita utilizar iterativamente la PDU de GETNEXTREQUEST, nos devolvería, mediante la creación de un único *service*, todo el conjunto de atributos asociados a los OID's escalares de la estructura de la MIB (en este caso aquellos relacionados con la RAM del dispositivo).

Para integrar esta nueva funcionalidad, únicamente sería necesaria la creación del *plugin* como tal, así como de un nuevo *command*. Para el primer elemento, se ha desarrollado un *shell script* que, en base a unos parámetros de entrada (entre los que se encontraría un OID), vaya llamando secuencialmente a sus sucesores. A continuación, se puede ver el código del *plugin*:

```
#!/bin/bash
while getopts ":v:c:H:O:" opt; do
          case $opt in
              v) version="$OPTARG"
              ;;
              c) comunidad="$OPTARG"
              ;;
              H) host="$OPTARG"
              ;;
              O) oid="$OPTARG"
              ;;
              \?) echo "Invalid option -$OPTARG" >&2
              ;;
          esac
        done
/usr/bin/snmpwalk -v $version -c $comunidad $host $oid
```

Como se puede apreciar en el código, lo que se pretende es asociar los parámetros de entrada a una instrucción que utiliza *snmpwalk* (del paquete NET-SNMP) para recuperar la información solicitada. Cabe destacar que los parámetros de entrada deben ser nominales, a diferencia de los esperados por *snmpwalk*, que no llevan ningún tipo de etiqueta asociada a los parámetros de *host*

y OID. Una vez esté creado el *command*, Icinga realizaría internamente el equivalente a la siguiente instrucción (obviando que algunos parámetros, como la IP, son simplemente un ejemplo para este supuesto):

```
# ./snmpwalk.sh -v 2c -c public -H 192.168.1.126 -O .1.3.6.1.4.1.2021.4
```

Dicha llamada sería el equivalente a ejecutar manualmente lo siguiente:

```
# snmpwalk -v2c -c public 192.168.1.126 .1.3.6.1.4.1.2021.4
```

A continuación se muestra la codificación del *command* asociado a este *plugin*, que como puede apreciarse, guarda una similitud con el empleado para *check_snmp*:

```
object CheckCommand "snmp_walk" {
        import "plugin-check-command"
        command = [ PluginDir + "/snmpwalk.sh" ]
        arguments = {
                "-H" = "$snmp_host$"
                "-O" = "$snmp_oid$"
                "-c" = "$snmp_com$"
                "-v" = "$snmp_vers$"
        }

        vars.snmp_host = "$snmp_host$"
        vars.snmp_oid = "$snmp_oid$"
        vars.snmp_com = "$snmp_com$"
        vars.snmp_vers = "$snmp_vers$"
}
```

Tal y como puede verse, el *command* nuevo espera la llegada de los cuatro parámetros que serán redirigidos al *plugin.* Dicho *script* iría situado, por comodidad, en */usr/lib/nagios/plugins*, si bien puede emplearse una ruta alternativa para su utilización, siempre que el usuario con el que se ejecuta Icinga tenga permisos de ejecución en ese directorio.

Una vez completada la edición de ambos ficheros, y habiendo creado adicionalmente un *service* que haga referencia a *snmp_walk*, quedaría recargar el servicio de Icinga. La información obtenida con *snmpwalk.sh* referente a la RAM del equipo, ya presentada vía web, sería similar a la mostrada en la Figura 35.

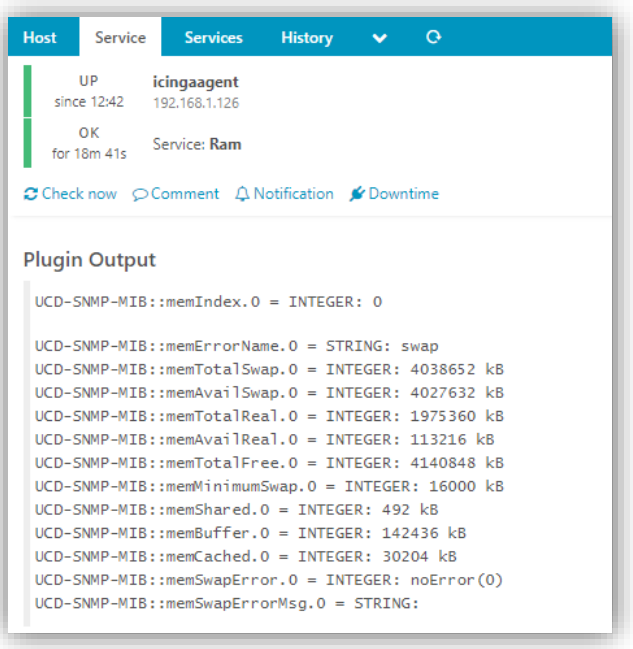

Figura 35. Representación de los OID's de la RAM mediante un plugin que usa snmpwalk en Icinga Web.

La otra necesidad detectada en este ámbito se apoya en la carencia de manejo de los datos obtenidos ya que, mediante el *plugin* por defecto (*check_snmp*), no podrían convertirse unidades, ni tampoco representarse los datos de una manera más intuitiva. Por ello, se ha desarrollado un *plugin* adicional que incorpora en su interior la validación de umbrales y una representación más amigable de la información. En cuanto al proceso de integración del mismo, igual que en el caso anterior, el proceso conlleva la creación de un *command* que actuará como pasarela para interpelar al *host*, así como de un *service* a través del cual efectuar la llamada.

Supóngase el caso del agente desplegado en el apartado 4.4.1 para los terminales con Android, podría resultar de utilidad que, con únicamente un *service*, se pudiera recuperar toda la información del *host*, estableciendo en el propio código del *plugin* aquellos OID's que son de nuestro interés. Permitiendo además, validar internamente los umbrales de los resultados obtenidos, y obviando así establecer dichos umbrales con la nomenclatura de Nagios, la cual es poco intuitiva. A continuación, se expone el código asociado al s*hell script*:

```
#!/bin/bash
while getopts ":v:c:H:p:" opt; do
        case $opt in
                v) version="$OPTARG"
                ;;
                c) comunidad="$OPTARG"
                ;;
```

```
                H) host="$OPTARG"
                ;;
                p) puerto="$OPTARG"
                ;;
                \?) echo "Invalid option -$OPTARG" >&2
                ;;
        esac
done

warning="0"
critical="0"
servs_w=""
servs_c=""

echo "-------------------------------------"
echo "               Resumen               "
echo "-------------------------------------"

#Descripción teléfono
resultado=$(snmpget     -v     $version     -c     $comunidad     $host:$puerto
1.3.6.1.4.1.12619.1.1.1.0 -Oqv) #Oqv saca la salida sin operadores ni tipo de
datos
echo "Modelo: $resultado"

#Versión de Android
resultado=$(snmpget     -v     $version     -c     $comunidad     $host:$puerto
1.3.6.1.4.1.12619.1.1.2.0 -Oqv)
echo "Versión Android: $resultado"

#Tiempo de encendido
resultado=$(snmpget     -v     $version     -c     $comunidad     $host:$puerto
1.3.6.1.4.1.12619.1.1.3.0 -Oqv)
echo "Uptime: $resultado"

echo "-------------------------------------"
echo "       Informacion dispositivo       "
echo "-------------------------------------"

# NUMERO DE SERVICIOS
```

```
resultado=$(snmpget     -v     $version     -c     $comunidad     $host:$puerto
1.3.6.1.4.1.12619.1.2.1.0 -Oqv)
if [[ ($resultado -gt 50) && ($resultado -lt 70)]]
then
        let "warning++"
        servs_w="${servs_w} NumServicios, "
fi
if [[ $resultado -gt 69 ]]
then
        let "critical++"
        servs_c="${servs_c} NumServicios, "
fi
echo "Numero Servicios: $resultado"

#Movil cargando
resultado=$(snmpget     -v     $version     -c     $comunidad     $host:$puerto
1.3.6.1.4.1.12619.1.3.1.0 -Oqv)
if [[ $resultado != 1 ]]
then
        let "warning++"
        servs_w="${servs_w}MovilCargando, "
        echo "Movil cargando: No"
else
        echo "Movil cargando: Si"
fi

#Batería Restante
resultado=$(snmpget     -v     $version     -c     $comunidad     $host:$puerto
1.3.6.1.4.1.12619.1.3.2.0 -Oqv)
if [[ ($resultado -lt 40) && ($resultado -gt 20)]]
then
        let "warning++"
        servs_w="${servs_w}BateriaRestante, "
fi
if [[ $resultado -lt 21 ]]
then
        let "critical++"
        servs_c="${servs_c}BateriaRestante, "
fi
echo "Batería Restante: $resultado"
```

```
#GPS Encendido
resultado=$(snmpget     -v     $version     -c     $comunidad     $host:$puerto
1.3.6.1.4.1.12619.1.3.3.0 -Oqv)
if [[ $resultado != 1 ]]
then
        let "critical++"
        servs_c="${servs_c}GPSEncendido, "
        echo "GPS Activado: No"
else
        echo "GPS Activado: Si"
fi

#Bluetooth Encendido
resultado=$(snmpget     -v     $version     -c     $comunidad     $host:$puerto
1.3.6.1.4.1.12619.1.3.4.0 -Oqv)
if [[ $resultado != 1 ]]
then
        let "critical++"
        servs_c="${servs_c}BluetoohEncendido, "
        echo "Bluetooh Activado: No"
else
        echo "Bluetooh Activado: Si"
fi

#Red Operativa
resultado=$(snmpget     -v     $version     -c     $comunidad     $host:$puerto
1.3.6.1.4.1.12619.1.3.5.0 -Oqv)
if [[ $resultado != 1 ]]
then
        let "critical++"
        servs_c="${servs_c}AdaptadorRedLevantado, "
        echo "Red Disponible: No"
else
        echo "Red Disponible: Sí"
fi


echo "-------------------------------------"
echo "          Resumen de Datos           "
```

```
echo "-------------------------------------"
echo "Warnings: $warning"
echo "Criticals: $critical"
#echo "Servicios en warning: " $servs_w
#echo "Servicios en critical: " $servs_c
echo -e "Servicios en warning:" $servs_w
echo -e "Servicios en critical:" $servs_c
echo "-------------------------------------"

if [[ $critical -gt 0 ]]
then
        exit 2
fi

if [[ $warning -gt 0 ]]
then
        exit 1
fi

if [[ ($warning -eq 0) && ($resultado -eq 0)]]
then
        exit 0
fi
then
       let "warning++"
       servs_w="${servs_w}BateriaRestante, "
fi
if [[ $resultado -lt 21 ]]
then
       let "critical++"
       servs_c="${servs_c}BateriaRestante, "
fi
echo "Batería Restante: $resultado"

#GPS Encendido
resultado=$(snmpget     -v     $version     -c     $comunidad     $host:$puerto
1.3.6.1.4.1.12619.1.3.3.0 -Oqv)
if [[ $resultado != 1 ]]
then
       let "critical++"
```

```
	servs_c="${servs_c}GPSEncendido, "
	echo "GPS Activado: No"
else
	echo "GPS Activado: Sí"
fi

#Bluetooth Encendido
resultado=$(snmpget -v $version -c $comunidad $host:$puerto
1.3.6.1.4.1.12619.1.3.4.0 -Oqv)
if [[ $resultado != 1 ]]
then
	let "critical++"
	servs_c="${servs_c}BluetoohEncendido, "
	echo "Bluetooh Activado: No"
else
	echo "Bluetooh Activado: Sí"
fi

#Red Operativa
resultado=$(snmpget -v $version -c $comunidad $host:$puerto
1.3.6.1.4.1.12619.1.3.5.0 -Oqv)
if [[ $resultado != 1 ]]
then
	let "critical++"
	servs_c="${servs_c}AdaptadorRedLevantado, "
	echo "Red Disponible: No"
else
	echo "Red Disponible: Sí"
fi

echo "-------------------------------------"
echo "          Resumen de Datos           "
echo "-------------------------------------"
echo "Warnings: $warning"
echo "Criticals: $critical"
#echo "Servicios en warning: " $servs_w
#echo "Servicios en critical: " $servs_c
echo -e "\e[33mServicios en warning: " $servs_w
echo -e "\e[31mServicios en critical: " $servs_c
echo "-------------------------------------"
```

```
if [[ $critical -gt 0 ]]
then
      exit 2
fi

if [[ $warning -gt 0 ]]
then
      exit 1
fi

if [[ ($warning -eq 0) && ($resultado -eq 0)]]
then
      exit 0
fi
```

Este script validaría todos los OIDs del agente (aquellos que ya aparecían durante la configuración detallada a lo largo del apartado 4.4), pero esta vez mediante el comando *snmpget* (aplicación del paquete NET-SNMP, y que es lo que utiliza realmente *check_snmp* internamente).

Como se puede apreciar, al final del código del nuevo *plugin*, se incluye una última aprobación, por la cual, si alguno de los OID's ha devuelto un valor que no se considera como "OK", todo el estado del *service* pasaría a estar en la situación del peor de los casos. Por ejemplo, si su ejecución devuelve varios OID's cuyo resultado es considerado "OK" pero, adicionalmente, hay un "WARNING" y un "CRITICAL", el *service* asociado a la ejecución de este *plugin* pasaría a estar en estado "CRITICAL". Nótese que los umbrales establecidos en el código son los mismos que los expuestos en el apartado 4.4.

A continuación, se presenta el *command* asociado con la ya conocida estructura (véase apartado 4.4.2):

```
object CheckCommand "check_moviles" {
        import "plugin-check-command"
        command = [ PluginDir + "/check_moviles.sh" ]
        arguments = {
                "-H" = "$snmp_host$"
                //"-O" = "$snmp_oid$"
                "-c" = "$snmp_com$"
```

```
            "-v" = "$snmp_vers$"
            "-p" = "$snmp_puerto$"
        }

        vars.snmp_host = "$snmp_host$"
        //vars.snmp_oid = "$snmp_oid$"
        vars.snmp_com = "$snmp_com$"
        vars.snmp_vers = "$snmp_vers$"
        vars.snmp_puerto = "$snmp_puerto$"
}
```

Una vez reiniciado o recargado el servicio de Icinga, y habiendo asignado el *service* a un host, podría contarse con la ejecución del nuevo *plugin*, que tendría una salida, vía web, similar a la mostrada en la Figura 36.

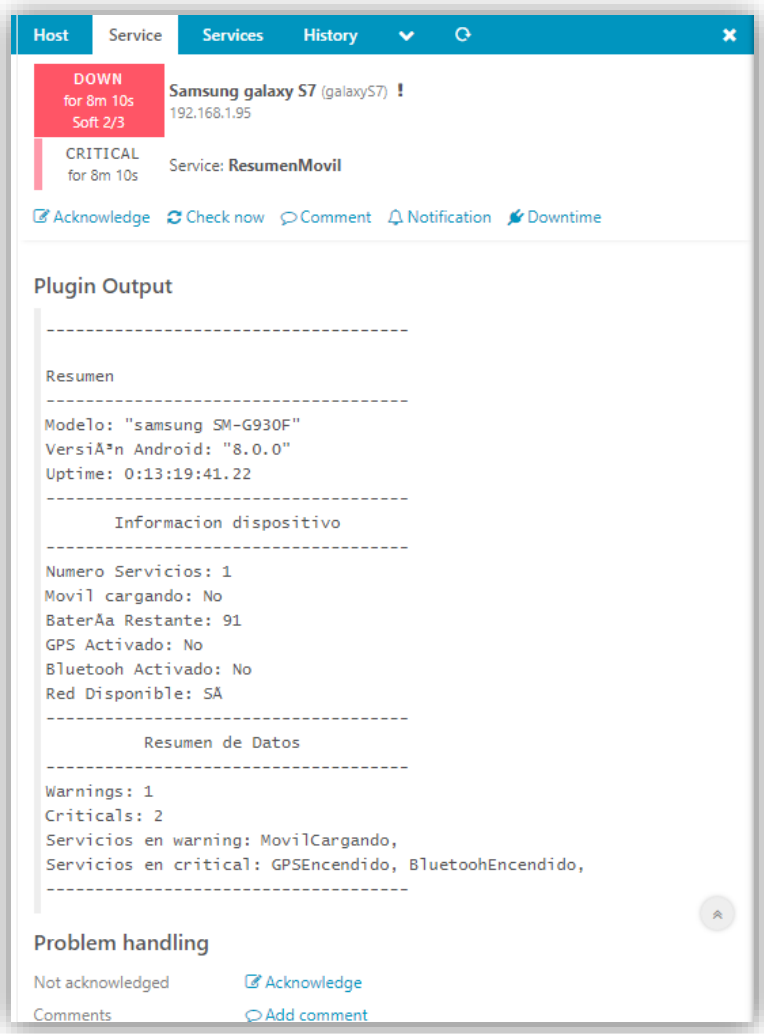

Figura 36. Representación vía web de la información recogida un plugin de validación.

Como puede apreciarse en la Figura 36, en la sección "Resumen de Datos" se recoge qué OID's han devuelto información cuyo estado es considerado como "WARNING" y "CRITICAL" mediante los umbrales. Dado que un OID está con estado "CRITICAL", el *service*, que resume todo el estado del *host*, estaría también en estado "CRITICAL".

5.2 Demonio para la recepción y visualización de *traps*

Como se mencionaba al comienzo del capítulo, otra insuficiencia detectada en Icinga desde un enfoque SNMP (ya que, probablemente, de no haber determinado esta estrategia de monitorización se detectarían carencias de otra naturaleza), es la ausencia de monitorización en su variante pasiva. A lo largo de los últimos apartados, la recolección de datos se ha realizado siempre de forma activa, es decir, el propio servidor es el que interroga a los *hosts* haciendo *polling*, y una vez tiene la información en su poder, la valida y la representa en la web. Sin embargo, no hay que olvidar que, a través de SNMP, los agentes pueden enviar notificaciones de forma asíncrona para notificar ciertos eventos, y es en esta modalidad donde se centra la nueva mejora a introducir.

Para llevar esta idea a término, se necesita poder recibir el *trap* vía UDP en el servidor, procesarlo con un manejador y, por último, hacer coincidir el identificador del equipo que envía la notificación con el *host* correspondiente, dado de alta en Icinga previamente. Por esta razón, este proceso se va a apoyar en los siguientes tres pilares:

- Un demonio de recepción de *traps*: en este caso se utilizará *snmptrapd*[19] para recibir los *traps* en el sistema; es el punto de entrada de la información. Tal y como se especifica en la documentación de Nagios, este servicio trabajará en conjunción con un manejador que permita el procesado de las notificaciones.

- El manejador de traps: *SNMPTT*[20] será el servicio destinado al procesado e interpretación del *trap* para servírselo al *script.*

- Un *script* de procesado: cruzará la información del equipo remoto (ubicada en la notificación) con el identificador que tiene Icinga guardado, a fin de conocer su

[19] https://support.nagios.com/kb/article.php?id=88
[20] http://www.snmptt.org/docs/snmptt.shtml#LoggingUnknown

procedencia y destinar el dato recogido a Icinga. Este tendrá configurado un servicio como “pasivo” para representar la información del *trap* a través de la web.

Como primer paso, habrá que instalar *snmpd* en el servidor de Icinga, ya que será necesario cambiar algunos parámetros de configuración para indicar que el paquete encargado de la recepción de *traps* será el proceso *snmptrapd*, además de detallar sus opciones. Por ello se ejecutan los siguientes comandos:

```
# apt-get update
# apt-get install snmpd
```

Una vez se ha completado el proceso, hay que cambiar el fichero que determina el comportamiento por defecto de *snmpd* (el fichero está ubicado en */etc/default/snmpd*). En concreto, se deberán modificar las líneas necesarias para que el fichero cuente con la siguiente configuración:

```
…
SNMPDRUN=yes
TRAPDRUN=yes
TRAPDOPTS='-On -Lsd -p /var/run/snmptrapd.pid'
…
```

Como se puede apreciar, en esta configuración se especifica que el proceso encargado de la recepción de traps será *snmptrapd*, con ese juego de parámetros, que indican lo siguiente (NET-SNMP, SourceForge, 2004):

- **On**: No se traducen los OIDs numéricos a su correspondencia en la MIB.
- **Lsd**: Los mensajes se volcarán en *syslog* para tener mejor trazabilidad a la hora de hacer pruebas.
- **p**: Guarda el ID del proceso en el fichero ubicado en */var/run/snmptrapd.pid*

El siguiente paso es instalar *snmptrapd* con el siguiente comando:

```
# apt-get install snmptrapd
```

A continuación se deberá cambiar su fichero de configuración para que, cuando se reciban los *traps*, éstos sean procesados por un manejador (*snmptt*). Para ello, es necesario editar el fichero

/etc/snmp/snmptrapd.conf para especificar la ubicación de *smptt*, ya que por defecto viene todo el fichero comentado:

```
traphandle default /usr/sbin/snmptthandler
disableAuthorization yes
```

La modificación efectuada en la segunda línea daría lugar a la admisión de todos los paquetes entrantes, sin importar su control de acceso. Esta es una manera de aceptar todas las notificaciones.

En este punto, y para continuar con el proceso, habría que instalar las MIB básicas, a fin de poder interpretar muchos de los *traps* de notificación que puedan llegar. Para ello se ejecutaría el siguiente comando:

```
# apt-get-install snmp-mibs-downloader
```

Por defecto, en la distribución Ubuntu, todas las MIB descargadas se encuentran en los archivos cuyas rutas son las siguientes:

- */usr/share/snmp/mibs*
- */var/lib/snmp/mibs*

Es común que estas MIBs sean eliminadas en los casos en los que no se necesiten todas. Por esto, si se eliminan algunas de forma accidental o se modifican, puede utilizarse el siguiente comando para forzar una nueva descarga de las mismas:

```
/usr/bin/download-mibs
```

El siguiente paso es instalar *snmptt*, ejecutando el siguiente comando:

```
# apt-get-install snmp
```

Posteriormente, se procede a modificar su configuración en el archivo */etc/snmp/snmptt.ini*. En este fichero hay que asegurarse de que contiene los siguientes ajustes entre su configuración:

- `mibs_environment = ALL`: para poder usar todas las MIBs desplegadas en el sistema.

- `net_snmp_perl_enable = 1`: habilita el procesado con el módulo de Perl que integra el paquete NET-SNMP.

- Todas las líneas que comiencen por "`translate_`" deben tener asignado el valor numérico "2", de esta forma, todos los *traps* que lleguen al sistema se traducirán con los nombres encontrados en las MIB y estarán representados de forma corta, indicando en su llegada la MIB a la que pertenece la notificación.

Una vez hecho esto, se debe modificar el fichero */etc/snmp/snmptt.conf*, el cual integra bastante configuración en su interior. Por ello, se recomienda generar una copia del original y borrar todo el contenido, dejando únicamente la siguiente configuración:

```
EVENT CatchAll .1.* SNMP Traps Critical
FORMAT $o
EXEC /usr/lib/nagios/plugins/submit_check_result $A snmp_traps 1 "$O@$+*"
```

De esta forma, se aceptan todos los *traps* que vayan llegando a través de *snmptrapd*. Por último, con esa información recogida acerca del *host* a partir de la notificación, se realiza una llamada al *script* de envío a Icinga, con los siguientes parámetros (SNMPTT Documentation, 2020):

- **$A**: será el host del que procede la notificación.
- **snmp_traps**: es el nombre del *service* que se dará de alta en Icinga para depositar la información.
- **1**: es la criticidad con la que se tratarán los *traps* en Icinga, tal y como se explicó en el apartado 4.5.1, el "1" se usa para indicar un valor de estado "WARNING".
- **"$O@$+*"**: la cadena de texto que contendrá la notificación del trap.

El directorio donde se ubica el *script* es arbitrario, pudiendo estar en cualquier ubicación que permita su ejecución. No obstante, como se han depositado en ese directorio otros ejecutables que tienen que ver con Icinga, se ha almacenado en éste por comodidad. El siguiente paso es crear el *script* de envío a Icinga. Dicho script está basado en el creado por Ethan Galstad[21] (personal de Nagios), para redirigir información a la cola de procesamiento de Nagios. En este caso será modificado para permitir el envío del *trap* de la siguiente manera:

21 https://github.com/NagiosEnterprises/nagioscore/blob/master/contrib/eventhandlers/submit_check_result

```
#!/bin/sh
# SUBMIT_CHECK_RESULT
# Written by Ethan Galstad (egalstad@nagios.org)
# Last Modified: 26 Oct 15
# . . .
# Arguments:
#  $1 = host_name (Short name of host that the service is associated with)
#  $2 = svc_description (Description of the service)
#  $3 = return_code (An integer that determines the state of the service check,
0=OK,
# 1=WARNING, 2=CRITICAL, 3=UNKNOWN).
#  $4 = plugin_output (A text string that should be used as the plugin output
for the service check)
#
echocmd="/bin/echo"
CommandFile="/var/run/icinga2/cmd/icinga2.cmd"
# get the current date/time in seconds since UNIX epoch
datetime=`date +%s`
identificador=$(mysql --password=tfg2020 -u root -r -e "select H.alias from
icinga_customvariables C join icinga_hosts H on H.host_object_id = C.object_id
where C.varname = 'snmp_host' and C.varvalue = '$1' limit 1;" -N -B icinga2)
# create the command line to add to the command file
cmdline="[$datetime] PROCESS_SERVICE_CHECK_RESULT;$identificador;$2;$3;$4"
# append the command to the end of the command file
`$echocmd $cmdline >> $CommandFile`
```

Como puede apreciarse en el código aquí expuesto, se establece la ruta donde se redirigirá la información una vez sea procesada (*/var/run/icinga2/cmd/icinga2.cmd*). Después, se realiza una consulta a la base de datos (en este caso MySQL) a fin de comprobar que el equipo remoto coincide con el guardado en la configuración de Icinga (es de crucial importancia que ambos nombres coincidan exactamente, ya que de otra manera el *trap* no será procesado).

Posteriormente, se almacena en la variable "cmdline" una cadena de texto con el formato que espera encontrar Icinga para procesar la información (separada por ";"):

- **$datetime:** Contiene la marca de tiempo recogida más arriba en el propio *script*.

- **PROCESS_SERVICE_CHECK_RESULT**: es la cadena que indica a Icinga que es un envío de un *service* pasivo.

- **$identificador**: el *snmp_host* del *host* concreto configurado en Icinga, ya validado a través de la consulta a la base de datos.

- **$2, $3, $4**: son los parámetros ya especificados anteriormente, y que también vienen comentados por el autor del script genérico en la parte inicial. Éstos son, respectivamente:
    - Nombre del *service*: en este caso será siempre "*snmp_traps*" como se ha podido ver en el fichero de configuración de *snmptt.*
    - Estado: el "1" indicado en la llamada del fichero de configuración (WARNING).
    - Cadena de texto que contiene el *trap*.

Una vez definido todo lo anterior, se deposita en la cola de procesamiento de Icinga (*/var/run/icinga2/cmd/icinga2.cmd*). En el caso de existir un *service* llamado "snmp_traps" para ese *host* de Icinga, la información será representada. Por ahora, dicho servicio no existe, será creado más adelante. El siguiente paso para continuar, es reiniciar todos los servicios asociados a estos ficheros de configuración y dar permisos de ejecución al *script* con la siguiente secuencia de comandos:

```
# service snmpd restart
# service snmptrapd restart
# service snmptt restart
# chmod +x /usr/lib/nagios/submit_check_result
```

Para que esta entrada de datos en la cola de procesamiento de Icinga sea posible, hay que facilitar los permisos de escritura en el fichero */var/run/icinga2/cmd/icinga2.cmd* con el siguiente comando:

```
# chmod 660 /var/run/icinga2/cmd/icinga2.cmd
prw-rw---- 1 nagios www-data 0 Dec 25 12:58 /var/run/icinga2/cmd/icinga2.cmd
```

Tras estas modificaciones, ya se podría dar de alta en Icinga el *service* correspondiente. Para este fin se ha creado adicionalmente un nuevo *template* (como los mostrados en el apartado 4.4.4) que usará el *service* pasivo que mostrará los *traps* (para este supuesto, tanto la codificación del *template* como la del *service* se ha depositado en un nuevo fichero llamado "traps.conf" dentro del directorio de configuración de Icinga). A continuación se presenta la codificación de ambos objetos:

```
template Service "snmp-trap-service" {
```

```
import "generic-service"
check_command = "passive"
enable_notifications = 1
enable_active_checks = 1
enable_passive_checks = 1
enable_flapping = 0
volatile = 1
max_check_attempts = 1
check_interval = 87000
enable_perfdata = 0
}

apply Service "snmp_traps" {
  import "snmp-trap-service"
  assign    where    (host.vars.grupo==    "agente_linux"    ||
host.vars.grupo== "agente_windows")
}
```

Como se puede apreciar, el *command* utilizado en esta ocasión es de tipo "passive" ya que, al estar a la espera de la información, no se necesita de la ejecución de un *plugin*. Otro parámetro que se ha añadido a la configuración es la comprobación del estado cada 24 horas (87000 segundos). De esta manera, un *trap* que haya llegado al sistema permanecerá visible en la web durante 24 horas. Una vez transcurra este tiempo, el sistema quedaría nuevamente a la espera de una nueva notificación, mostrando "No Passive Check Result" en la salida del *service*. Sin embargo, esa información no desaparece; pueden consultarse los *traps* enviados por ese agente en la pestaña "History". Por último, ahora sí, dentro de la configuración del *service*, se ha asignado este servicio a los *hosts* pertenecientes a los agentes Linux y Windows, respectivamente (el despliegue y configuración de estos agentes puede encontrarse en el Apéndice A, al final de esta memoria).

Una vez completado todo este proceso, y habiendo reiniciado los servicios pertinentes para aplicar la configuración, pueden realizarse pruebas con la nueva configuración del sistema enviando un *trap* manualmente (o forzando un evento) desde un nodo externo que exista en la configuración de Icinga. Para ello, se pueden utilizar órdenes de envío como la siguiente:

```
# snmptrap -v1 -c public 192.168.1.130 .1.3.6.1.6.3.1.1.5.2 0 0 "" ""
.1.3.6.1.4.1 s "hello"
```

Otro ejemplo, pero en SNMPv2 y usando nombres de la propia MIB, sería el siguiente:

```
# snmptrap -v 2c -c public 192.168.1.130 '' NET-SNMP-EXAMPLES-
MIB::netSnmpExampleHeartbeatNotification netSnmpExampleHeartbeatRate i 123456
```

Si se comprueba el *syslog* en la máquina virtual donde se encuentra Icinga, puede observarse que dichos *traps* están llegando correctamente y que pasan tanto por *snmptrapd* como por *snmptt*:

```
Dec 25 14:16:19 icingatfg snmptrapd[2954]: 2020-12-25 14:16:19 0.0.0.0(via UDP:
[192.168.1.126]:52256->[192.168.1.130]:162) TRAP, SNMP v1, community
public#012#011SNMPv2-MIB::warmStart Cold Start Trap (0) Uptime:
1:18:06.61#012#011SNMPv2-SMI::enterprises = STRING: "hello"
Dec 25 14:16:23 icingatfg snmptt[2655]: SNMPv2-MIB::coldStart Traps "SNMP"
192.168.1.126 - .1.3.6.1.6.3.1.1.5.1
```

Y por último, si el usuario inicia sesión desde la web de Icinga, pueden verse dichos *traps* en Icinga buscando el servicio asociado a ese *host* (ver Figura 37).

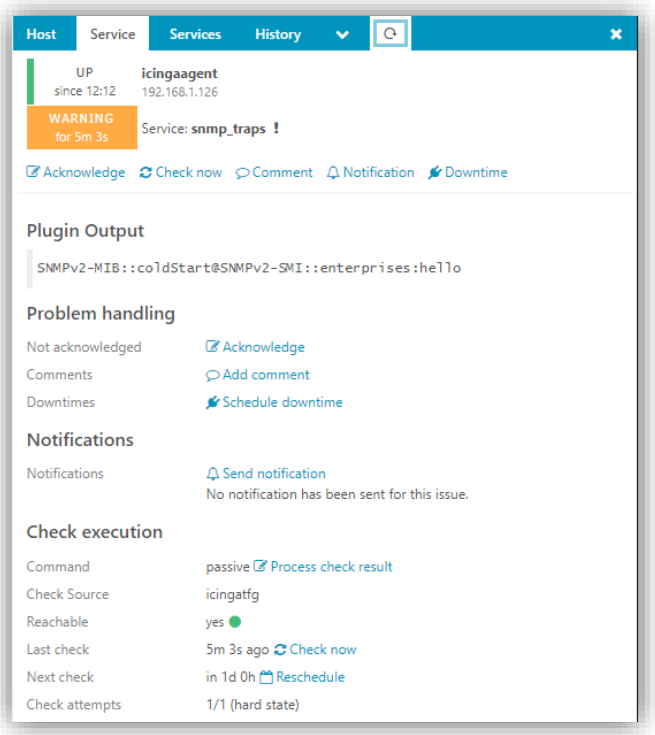

Figura 37. Salida mostrada por Icinga ante la recepción de un trap.

Si se desea comprobar los *traps* recibidos, previos al que aparece en la Figura 37, puede hacerse pulsando la pestaña "History", la cual listará por antigüedad todos los mensajes recibidos desde ese agente SNMP. Los dos agentes configurados con posibilidad de enviar *traps* (Windows y

Linux), enviarán notificaciones de forma autónoma ante ciertos eventos (ejemplos: cuando sean reiniciados, cuando reciban solicitudes SNMP a través de una comunidad que no corresponde con la establecida o ante otros sucesos que tengan relevancia, si bien esto puede ser configurado con mayor granularidad por el administrador de esos sistemas dentro de los propios agentes).

5.2.1 Explotación y consulta de datos mediante un visor de *traps*

A pesar de que el demonio de recepción de *traps* es, en sí mismo, un avance, al permitir visualizar la llegada de notificaciones procedente de equipos remotos a través de la web de Icinga, a su vez entraña algunas carencias. Una de ellas es que los *traps* se visualizan por separado, es decir, pueden ser consultados revisando el *service* de cada agente SNMP presente en Icinga, pero no pueden verse juntos, a no ser que se realice a través de la creación de un *dashboard* que lo permita. No obstante, incluso con esta solución, en la propia pizarra en la que se incluyese el servicio de *traps* de cada agente, únicamente figuraría la última notificación recibida por cada uno de ellos, y no los *traps* anteriores. Esto tampoco es un problema en sí mismo, dado pueden consultarse vía web pulsando en la pestaña "History" de cada *service*, si bien es algo laborioso. El hecho de que por defecto no se pueda mezclar el histórico de *traps* de un *host* con el de los demás hace que sea un poco más tedioso ordenar en una línea temporal los sucesos, a fin de determinar las causas y el origen de un problema. Algo que puede resultar muy útil cuando quiere hacerse análisis forense de un suceso que ha provocado una serie de problemas en cadena.

Por todo lo mencionado, de la mejora del demonio de *traps* nace una nueva necesidad, y ésta puede ser cubierta gracias a que dichas notificaciones quedan guardadas en los históricos de Icinga. Estos datos pueden ser consultados de una forma más intuitiva en una base de datos si, en la configuración de IDO-MySQL se le indica al servicio de Icinga que utilice determinadas tablas para alojar esos datos. Este proceso implica modificar el fichero */etc/icinga2/features-available/ido-mysql.conf*, ampliando el campo "categories" con el siguiente código:

```
library "db_ido_mysql"
object IdoMysqlConnection "ido-mysql" {
  user = "icinga2",
  password = "tfg2020",
  host = "localhost",
  database = "icinga2"
  categories = [DbCatConfig , DbCatState , DbCatAcknowledgement , DbCatComment
, DbCatDowntime , DbCatEventHandler , DbCatExternalCommand , DbCatFlapping ,
```

```
DbCatCheck , DbCatNotification , DbCatProgramStatus , DbCatRetention ,
DbCatStateHistory]
  cleanup = {
     servicechecks_age = 365d
  }
}
```

Adicionalmente, se ha establecido que la información contenida se vaya eliminando de esas tablas pasados 365 días, para no saturar la base de datos de contenido en caso de que ésta creciera sin control. Una vez reiniciado el servicio de icinga2, así como el de mysql, los datos comenzarían a guardarse en las tablas situadas dentro de la base de datos "icinga2", en un formato más manejable.

Antes de codificar el visor, conviene familiarizarse con la estructura que mantiene la base de datos de Icinga. Por ello, se ha realizado una exploración de su contenido con clientes gráficos como *Heidi*, a fin de determinar la forma en la que se almacena la información. Tras este análisis, se ha determinado que, para identificar inequívocamente un *trap*, se necesitan 3 factores: el *host* del que procede, la fecha y hora de su recepción y su contenido. Por ello, se ha resuelto que los *traps* se almacenan en la tabla "`icinga_servicechecks`", pero en ésta no figuran los nombres de los *hosts*, ni tampoco del nombre del *service*; únicamente un ID que hace referencia a cada objeto por separado. Por ello, es necesario cruzar los datos de tres tablas para filtrar y mostrar las 3 columnas:

- `icinga_servicechecks`
- `icinga_services`
- `icinga_hosts`

Por lo tanto, una vez analizada su estructura, y habiendo comprendido la lógica en la que se guardan los datos, se ha procedido con la codificación del visor de *traps*. Dicho visor es una aplicación codificada en *Powershell*, que se apoya en *WinForms* para ofrecer la interfaz de usuario. Esta utilidad permite realizar las siguientes operaciones:

- Visibilizar la entrada de los últimos 20 *traps* que han entrado al sistema (cuya procedencia sea cualquier *host* configurado previamente en Icinga).
- Consultar todos los *traps* recibidos en las últimas 24 horas.
- Visualizar todos los *traps* procedentes de un *host* bajo elección del usuario.
- Visualizar todos los *traps* recibidos en una fecha concreta, bajo elección del usuario.

- Exportar los *traps* a un fichero *csv* que permita su explotación posterior con alguna herramienta como *Microsoft Excel*. Dicha exportación puede realizarse con tantos días de antigüedad como considere el usuario teniendo en cuenta que los datos, con la configuración actual, pueden ser consultados con un máximo de 365 días de antigüedad.

En base a estas funcionalidades, y tratando de mantener la mayor organización y eficiencia posibles del código, se han creado las siguientes funciones de Powershell mediante las cuales se llevan a cabo las operaciones anteriores:

- **`function ConsultaTraps{ param ( [string[]]$query) }:`**

  Esta función recibe una cadena de texto que contiene una consulta SQL ya formada en formato *string*, y devuelve el resultado de dicha *query* en forma de un *DataSet*. Dicha función hace uso del conector .NET/MySQL para llevar a término la consulta y contempla el manejo de errores en caso de que ésta no suceda como se espera.

- **`function PintarRejilla{ param ($res_consulta) }:`**

  Esta función espera como parámetro de entrada un *DataSet* con un conjunto de valores (en general, obtenidos tras realizar una consulta a la base de datos). Una vez esta función recibe la información, transforma el contenido del objeto en un *ArrayList* cuyo contenido será representado en un *DataGridView* que contendrá las 3 columnas mencionadas anteriormente para identificar unívocamente un *trap*.

- **`function VerUltimos20{}:`**

  Esta función realiza una llamada a las dos funciones anteriores con la consulta. Ésta devolvería los veinte últimos *traps* recogidos en todo el sistema; ordenados por fecha, independientemente de su momento de entrada o *host* de procedencia. Dicha información puede recargarse pulsando el botón de "Refrescar".

- **`function Traps24h{}:`**

  Esta función realiza una llamada a las dos primeras funciones con la consulta que devolvería todos los *traps* recogidos en todo el sistema en las 24 horas anteriores al instante actual, por orden de llegada. Esto suele resultar bastante útil para revisar rápidamente lo acontecido en las últimas horas.

- **`function TrapsHost{ param ( [string[]]$hostbuscar)}:`**

  Esta función permite visualizar todos los *traps* procedentes de un *host* a elección del usuario. Recibe como parámetro de entrada la cadena de texto recogida por teclado a

través de la interfaz gráfica. Una vez hecho esto, se conforma la consulta a base de datos, se ejecuta y se representa el resultado en la rejilla.

- **`function TrapsFecha{param ( [string[]]$day, [string[]]$month, [string[]]$year )`**

  Esta función permite buscar y visualizar todos los *traps* cuya recepción sea en una fecha introducida manualmente por el usuario a través de la interfaz gráfica. Una vez se tienen los tres parámetros (día, mes y año), se conforma la consulta, se ejecuta y se representa su resultado en el *DataGridView*, a través de las dos primeras funciones.

- **`function ExportTraps{}:`**

  Esta función solicita al usuario, a través de la interfaz, la antigüedad con la que quiere recuperar el histórico de *traps* (a partir del día y hora actuales) y, en base a ello, se realiza una consulta a la base de datos, volcando dicho resultado en un fichero *csv* que se guarda en el Escritorio del usuario que ejecuta el visor. Adicionalmente, se representa el resultado en la rejilla.

El resto del código del visor se corresponde con la representación de opciones del menú principal, parámetros de conexión a base de datos y funcionalidad de los botones que conforman la interfaz. Además, al conjunto de botones que relaciona cada opción con las funciones vistas anteriormente, se incorpora un botón para abandonar el visor en cualquier momento.

Como requisitos de uso para el visor tendríamos los siguientes:

- Previa instalación del conector de base de datos .NET/MySQL.
- Recomendado el uso de Powershell 5.0 (o superior).
- Recomendado tener habilitada la política de ejecución de *scripts* de Powershell.
- Configuración correcta de la cadena de conexión al servidor. Se ha determinado que, para aumentar su portabilidad, el destino de la conexión será la DNS "icingasnmp.intranet.es" y las credenciales serán las de un usuario con permisos para conectar con ese esquema de base de datos.

A continuación, se presenta el código completo del visor de *traps*. Como puede apreciarse, al comienzo de la codificación se especifica que el diseño gráfico de la interfaz se ha realizado a través de *Poshgui*[22], un editor web para interfaces de usuario con *Winforms*, entre otras funcionalidades:

[22] https://poshgui.com/

```
<# This form was created using POSHGUI.com a free online gui designer for PowerShell
.NAME
    VisorTraps #>

#Declaración de todos los objetos visuales que se muestran en la interfaz gráfica
#con WinForms así como la definición de sus propiedades y coordenadas.
Add-Type -AssemblyName System.Windows.Forms
[System.Windows.Forms.Application]::EnableVisualStyles()

$Form                            = New-Object system.Windows.Forms.Form
$Form.ClientSize                 = New-Object System.Drawing.Point(1278,671)
$Form.text                       = "Visor de Traps"
$Form.TopMost                    = $false

$Ultimos20                       = New-Object system.Windows.Forms.Button
$Ultimos20.text                  = "Ultimos 20 Traps"
$Ultimos20.width                 = 164
$Ultimos20.height                = 30
$Ultimos20.location              = New-Object System.Drawing.Point(86,579)
$Ultimos20.Font                  = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$Traps24h                        = New-Object system.Windows.Forms.Button
$Traps24h.text                   = "Traps Ultimas 24hrs"
$Traps24h.width                  = 164
$Traps24h.height                 = 30
$Traps24h.location               = New-Object System.Drawing.Point(269,579)
$Traps24h.Font                   = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$TrapsHost                       = New-Object system.Windows.Forms.Button
$TrapsHost.text                  = "Traps de un Host"
$TrapsHost.width                 = 164
$TrapsHost.height                = 30
$TrapsHost.location              = New-Object System.Drawing.Point(456,579)
$TrapsHost.Font                  = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$ExportarHistoricos              = New-Object system.Windows.Forms.Button
$ExportarHistoricos.text         = "Exportar Historicos"
$ExportarHistoricos.width        = 164
$ExportarHistoricos.height       = 30
$ExportarHistoricos.location     = New-Object System.Drawing.Point(828,579)
$ExportarHistoricos.Font         = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$UAH                             = New-Object system.Windows.Forms.PictureBox
$UAH.width                       = 152
$UAH.height                      = 78
```

```
$UAH.location                    = New-Object System.Drawing.Point(29,22)
$UAH.imageLocation               = "https://www.uah.es/export/sites/uah/.galleries/im
agenes-estructura/logopie.png"
$UAH.SizeMode                    = [System.Windows.Forms.PictureBoxSizeMode]::zoom
$Salir                           = New-Object system.Windows.Forms.Button
$Salir.text                      = "Salir"
$Salir.width                     = 164
$Salir.height                    = 30
$Salir.location                  = New-Object System.Drawing.Point(1015,579)
$Salir.Font                      = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$PictureBox1                     = New-Object system.Windows.Forms.PictureBox
$PictureBox1.width               = 155
$PictureBox1.height              = 74
$PictureBox1.location            = New-Object System.Drawing.Point(181,17)
$PictureBox1.imageLocation       = "https://encrypted-
tbn0.gstatic.com/images?q=tbn:ANd9GcRLYHYyPScrN2S8VPBDaLnmxtYbnjZ3goQG_w&usqp=CAU"
$PictureBox1.SizeMode            = [System.Windows.Forms.PictureBoxSizeMode]::zoom
$Label1                          = New-Object system.Windows.Forms.Label
$Label1.text                     = "label"
$Label1.AutoSize                 = $true
$Label1.visible                  = $true
$Label1.enabled                  = $true
$Label1.width                    = 25
$Label1.height                   = 30
$Label1.location                 = New-Object System.Drawing.Point(101,113)
$Label1.Font                     = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$DataGridView1                   = New-Object system.Windows.Forms.DataGridView
$DataGridView1.width             = 1095
$DataGridView1.height            = 390
$DataGridView1.location          = New-Object System.Drawing.Point(86,151)

$TextoPregunta                   = New-Object system.Windows.Forms.Label
$TextoPregunta.AutoSize          = $true
$TextoPregunta.visible           = $false
$TextoPregunta.enabled           = $false
$TextoPregunta.width             = 25
$TextoPregunta.height            = 10
$TextoPregunta.location          = New-Object System.Drawing.Point(626,113)
$TextoPregunta.Font              = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$Opcion                          = New-Object system.Windows.Forms.TextBox
$Opcion.multiline                = $false
$Opcion.width                    = 143
$Opcion.height                   = 30
```

```
$Opcion.visible                  = $false
$Opcion.enabled                  = $false
$Opcion.location                 = New-Object System.Drawing.Point(940,105)
$Opcion.Font                     = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)
$Opcion.ForeColor                = [System.Drawing.ColorTranslator]::FromHtml("")

$Buscar                          = New-Object system.Windows.Forms.Button
$Buscar.text                     = "Buscar"
$Buscar.width                    = 73
$Buscar.height                   = 30
$Buscar.visible                  = $false
$Buscar.enabled                  = $false
$Buscar.location                 = New-Object System.Drawing.Point(1106,100)
$Buscar.Font                     = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$Exportar                        = New-Object system.Windows.Forms.Button
$Exportar.text                   = "Exportar"
$Exportar.width                  = 73
$Exportar.height                 = 30
$Exportar.visible                = $false
$Exportar.enabled                = $false
$Exportar.location               = New-Object System.Drawing.Point(1106,100)
$Exportar.Font                   = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$TrapsFecha                      = New-Object system.Windows.Forms.Button
$TrapsFecha.text                 = "Traps por Fecha"
$TrapsFecha.width                = 164
$TrapsFecha.height               = 30
$TrapsFecha.location             = New-Object System.Drawing.Point(644,579)
$TrapsFecha.Font                 = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$Dia                             = New-Object system.Windows.Forms.Label
$Dia.text                        = "Dia (dd):"
$Dia.AutoSize                    = $true
$Dia.visible                     = $false
$Dia.enabled                     = $false
$Dia.width                       = 25
$Dia.height                      = 30
$Dia.location                    = New-Object System.Drawing.Point(426,113)
$Dia.Font                        = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$TextoDia                        = New-Object system.Windows.Forms.TextBox
$TextoDia.multiline              = $false
$TextoDia.width                  = 36
```

```
$TextoDia.height                = 30
$TextoDia.visible               = $false
$TextoDia.enabled               = $false
$TextoDia.location              = New-Object System.Drawing.Point(487,109)
$TextoDia.Font                  = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$Mes                            = New-Object system.Windows.Forms.Label
$Mes.text                       = "Mes (mm):"
$Mes.AutoSize                   = $true
$Mes.visible                    = $false
$Mes.enabled                    = $false
$Mes.width                      = 25
$Mes.height                     = 30
$Mes.location                   = New-Object System.Drawing.Point(552,113)
$Mes.Font                       = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$TextoMes                       = New-Object system.Windows.Forms.TextBox
$TextoMes.multiline             = $false
$TextoMes.width                 = 36
$TextoMes.height                = 30
$TextoMes.visible               = $false
$TextoMes.enabled               = $false
$TextoMes.location              = New-Object System.Drawing.Point(625,109)
$TextoMes.Font                  = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$Anyo                           = New-Object system.Windows.Forms.Label
$Anyo.text                      = "Anyo (aaaa):"
$Anyo.AutoSize                  = $true
$Anyo.visible                   = $false
$Anyo.enabled                   = $false
$Anyo.width                     = 25
$Anyo.height                    = 30
$Anyo.location                  = New-Object System.Drawing.Point(687,113)
$Anyo.Font                      = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

$TextoAnyo                      = New-Object system.Windows.Forms.TextBox
$TextoAnyo.multiline            = $false
$TextoAnyo.width                = 57
$TextoAnyo.height               = 30
$TextoAnyo.visible              = $false
$TextoAnyo.enabled              = $false
$TextoAnyo.location             = New-Object System.Drawing.Point(773,109)
$TextoAnyo.Font                 = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)
```

```
$BuscaFecha                      = New-Object system.Windows.Forms.Button
$BuscaFecha.text                 = "Buscar"
$BuscaFecha.width                = 60
$BuscaFecha.height               = 30
$BuscaFecha.visible              = $false
$BuscaFecha.enabled              = $false
$BuscaFecha.location             = New-Object System.Drawing.Point(854,105)
$BuscaFecha.Font                 = New-
Object System.Drawing.Font('Microsoft Sans Serif',10)

#Incorporación de todos los objetos anteriores al Form
$Form.controls.AddRange(@($Ultimos20,$Traps24h,$TrapsHost,$ExportarHistoricos,$UAH,$S
alir,$PictureBox1,$Label1,$DataGridView1,$TextoPregunta,$Opcion,$Buscar,$Exportar,$Tr
apsFecha,$Dia,$TextoDia,$Mes,$TextoMes,$Anyo,$TextoAnyo,$BuscaFecha))
#FIN de la codificación de la interfaz gráfica

# Parámetros de conexión a la base de datos
$IPMySQL = "icingasnmp.intranet.es"
$PuertoMySQL = "3306"
$UsuarioMySQL = "dbadmin"
$ContrasenaMySQL = "tfg2020"
$BDIcinga = "icinga2"

######################
# F U N C I O N E S #
######################

<# Esta función recibe  como parámetro de entrada un string que contiene
la consulta a la BD para recuperar la información que se utilizará
en cada una de las funcionalidades. Dicha información se devuelve
como un DataSet que podrá representarse más tarde en la interfaz #>
function ConsultaTraps { param ( [string[]]$query)
    try{ # captura de errores que puedan producirse al conectar a la BD
            #se carga el complemento de .NET para MySQL y se utiliza la cadena de con
exión con parámetros anteriores
            [void][System.Reflection.Assembly]::LoadWithPartialName("MySql.Data")
            $Connection = New-Object MySql.Data.MySqlClient.MySqlConnection
            $ConnectionString = "server=$IPMySQL" + ";port=$PuertoMySQL;uid=$UsuarioM
ySQL" + ";pwd=$ContrasenaMySQL" + ";database=$BDIcinga"
            $Connection.ConnectionString = $ConnectionString
            $Connection.Open() #se abre la conexión con el destino
            #a continuación se lanza la query
            $Command = New-
Object MySql.Data.MySqlClient.MySqlCommand($query, $Connection)
            $DataAdapter = New-
Object MySql.Data.MySqlClient.MySqlDataAdapter($Command)
            #se declara el objeto DataSet
            $DataSet = New-Object System.Data.DataSet
            #se cuenta el número de filas devueltas en la consulta
```

```
            $RecordCount = $dataAdapter.Fill($dataSet, "data")
            #En caso de que que la consulta no devuelva ninguna fila para esa consult
a se muestra el siguiente mensaje
            if ($RecordCount -eq 0) {
                LimpiarInterfaz
                $Label1.Text = "Vaya... parece que no se ha encontrado ningun trap pa
ra esta consulta..."
            } else{
                $Label1.Text = "Hay un total de $RecordCount traps en el sistema:"
            }
            #se vuelca el resultado de la consulta en la variable y se efectúa el ret
orno de la información
            $resultado = $DataSet.Tables[0]
            $Connection.Close() #cierre de conexión a la BD
    } catch {$Label1.Text = "Vaya, parece que algo no ha ido bien en la consulta..."}
    return $resultado
}

<# Esta función recibe un DataSet con el resultado de una consulta a la BD
y representa su contenido en el DatagridView (la rejilla de visualización).#>
function PintarRejilla { param ($res_consulta)
    #se crea un objeto de tipo ArrayList
    $array= New-Object System.Collections.ArrayList
    #Se convierte el DataSet en un ArrayList y se cambia el nombre de las cabeceras d
e la tabla.
    #A continuación se integra también el control de errores en el caso de que el res
ultado sea nulo.
    try {
        $array.AddRange( @($res_consulta | Select-
Object end_time, display_name, output | Select-
Object @{Label='Fecha y hora'; Expression={$_.end_time}},@{Label='Host'; Expression={
$_.display_name}},@{Label='Trap'; Expression={$_.output}}) )
    } catch {
        Write-Debug "Consulta vacía"
    }
    #A continuación se rellena la tabla con el contenido del ArrayList
    $DataGridView1.AutoSizeColumnsMode = 'Fill'
    $dataGridView1.DataSource=$Null
    $dataGridView1.AutoGenerateColumns = $True
    $DataGridView1.DataSource = $array
    $dataGridView1.Refresh
}

<# Esta función muestra los últimos 20 traps recibido en todo el sistema,
independientemente de la fecha de su recepción#>
function VerUltimos20  {
    #se guarda la query que devuelve ese resultado en una variable
    $query='
    SELECT icinga_servicechecks.end_time,
```

```
           icinga_servicechecks.output,
           icinga2.icinga_hosts.display_name

    FROM icinga2.icinga_servicechecks
    JOIN icinga2.icinga_services ON icinga2.icinga_servicechecks.service_object_id =
icinga2.icinga_services.service_object_id
    JOIN icinga2.icinga_hosts ON icinga2.icinga_hosts.host_object_id = icinga2.icinga
_services.host_object_id

    WHERE (icinga2.icinga_services.display_name = "snmp_traps" AND icinga2.icinga_ser
vicechecks.return_code <> 3)
    ORDER BY icinga2.icinga_servicechecks.end_time desc
    LIMIT 20;'
    #se realiza la llamada a la función que realiza la consulta a la BD
    $res_consulta = ConsultaTraps($query)
    #llamada a la función que pinta el resultado en la rejulla de visualización
    PintarRejilla($res_consulta)
    $Label1.Text = "Mostrando los ultimos 20 traps recibidos en el sistema:"
}

<# Esta función muestra los traps recibidos en todo el sistema durante las últimas 24
hrs#>
function Traps24h  {
    #se guarda la query que devuelve el resultado deseado en una variable
    $query='
    SELECT icinga_servicechecks.end_time,
         icinga_servicechecks.output,
         icinga2.icinga_hosts.display_name
    FROM icinga2.icinga_servicechecks

    JOIN icinga2.icinga_services ON icinga2.icinga_servicechecks.service_object_id =
icinga2.icinga_services.service_object_id
    JOIN icinga2.icinga_hosts ON icinga2.icinga_hosts.host_object_id = icinga2.icinga
_services.host_object_id

    WHERE (icinga2.icinga_services.display_name = "snmp_traps" AND icinga2.icinga_ser
vicechecks.return_code <> 3 AND icinga_servicechecks.end_time >= NOW() - INTERVAL 1 D
AY )
        ORDER BY icinga2.icinga_servicechecks.end_time desc;'
    #se realiza la llamada a la función que realiza la consulta a la BD
    $res_consulta = ConsultaTraps($query)
    #llamada a la función que pinta el resultado en la rejulla de visualización
    PintarRejilla($res_consulta)
}

<# Esta función recibe como parámetro el nombre de un agente SNMP del cuál
se desean extraer los traps que se encuentren en la BD. #>
function TrapsHost  { param ( [string[]]$hostbuscar)
    #se guarda la query en una variable, nótese que se incluye la variable 'hostname'
```

```
    #con la cadena recibida como parámetro
    $query='
    SELECT icinga_servicechecks.end_time,
         icinga_servicechecks.output,
         icinga2.icinga_hosts.display_name

    FROM icinga2.icinga_servicechecks
    JOIN icinga2.icinga_services ON icinga2.icinga_servicechecks.service_object_id =
icinga2.icinga_services.service_object_id
    JOIN icinga2.icinga_hosts ON icinga2.icinga_hosts.host_object_id = icinga2.icinga
_services.host_object_id
    WHERE (icinga2.icinga_services.display_name = "snmp_traps" AND icinga2.icinga_ser
vicechecks.return_code <> 3 AND icinga2.icinga_hosts.alias="' + $hostbuscar + '")
        ORDER BY icinga2.icinga_servicechecks.end_time desc;'
    #se realiza la llamada a la función que realiza la consulta a la BD
    $res_consulta = ConsultaTraps($query)
    #llamada a la función que pinta el resultado en la rejulla de visualización
    PintarRejilla($res_consulta)
}

<# Esta función recibe como parámetro una fecha (día, mes y año) y efectúa
una búsqueda en la BD para devolver todos los traps recibidos en ese día#>
function TrapsFecha  { param ( [string[]]$day, [string[]]$month, [string[]]$year )
    #se guarda la query en una variable, nótese que se incluyen las variables
    #que contienen la fecha a buscar.
    $query='
    SELECT icinga_servicechecks.end_time,
           icinga_servicechecks.output,
           icinga2.icinga_hosts.display_name

    FROM icinga2.icinga_servicechecks
    JOIN icinga2.icinga_services ON icinga2.icinga_servicechecks.service_object_id =
icinga2.icinga_services.service_object_id
    JOIN icinga2.icinga_hosts ON icinga2.icinga_hosts.host_object_id = icinga2.icinga
_services.host_object_id

    WHERE (icinga2.icinga_services.display_name = "snmp_traps" AND icinga2.icinga_ser
vicechecks.return_code <> 3 AND icinga2.icinga_servicechecks.end_time BETWEEN "' + $y
ear + '-' + $month + '-' + $day + ' 00:00:00" AND "' + $year + '-' + $month + '-
' + $day + ' 23:59:59")
    ORDER BY icinga2.icinga_servicechecks.end_time desc;'
    #se realiza la llamada a la función que realiza la consulta a la BD
    $res_consulta = ConsultaTraps($query)
    #llamada a la función que pinta el resultado en la rejulla de visualización
    PintarRejilla($res_consulta)
}

<# Esta función recibe una antigüedad (en días) con la extraer
un histórico de todos los traps que se encuentran en el sistema,
```

```
independientemente del host de origen. La fecha de partida es el
momento en el que se realiza la consulta. El resultado de esa consulta
se exporta a un fichero csv que se deposita en el Escritorio del usuario
con el que se ejecute el Visor #>
function ExportarHistoricos { param ( [string[]]$numdias)
    #si la cantidad de días es cero se devuelven los traps del día actual
    if ($numdias -eq 0) {
        $query='
        SELECT icinga_servicechecks.end_time,
             icinga_servicechecks.output,
             icinga2.icinga_hosts.display_name
        FROM icinga2.icinga_servicechecks

        JOIN icinga2.icinga_services ON icinga2.icinga_servicechecks.service_object_i
d = icinga2.icinga_services.service_object_id
        JOIN icinga2.icinga_hosts ON icinga2.icinga_hosts.host_object_id = icinga2.ic
inga_services.host_object_id

        WHERE (icinga2.icinga_services.display_name = "snmp_traps" AND icinga2.icinga
_servicechecks.return_code <> 3)
            ORDER BY icinga2.icinga_servicechecks.end_time desc;'
    } else { #en otro caso se utiliza la antigüedad recibida como parámetro en la fun
ción.
        $query='
        SELECT icinga_servicechecks.end_time,
             icinga_servicechecks.output,
             icinga2.icinga_hosts.display_name
        FROM icinga2.icinga_servicechecks

        JOIN icinga2.icinga_services ON icinga2.icinga_servicechecks.service_object_i
d = icinga2.icinga_services.service_object_id
        JOIN icinga2.icinga_hosts ON icinga2.icinga_hosts.host_object_id = icinga2.ic
inga_services.host_object_id

        WHERE (icinga2.icinga_services.display_name = "snmp_traps" AND icinga2.icinga
_servicechecks.return_code <> 3 AND icinga_servicechecks.end_time >= NOW() - INTERVAL
 "' + $numdias + '" DAY )
            ORDER BY icinga2.icinga_servicechecks.end_time desc;'
    }
    #Se guarda en una variable la ruta al escritorio del usuario
    $RutaEscritorio = [Environment]::GetFolderPath("Desktop")
    #Tratamiento del nombre de fichero en función de la antigüedad seleccionada
    $fechahoy = Get-Date -Format "dd_MM_yyyy-HH_mm"
    if ($numdias -eq 0) {
        $FicheroSalida = "$RutaEscritorio\traps-todos-dias-$fechahoy.csv"
    } else {
        $FicheroSalida = "$RutaEscritorio\traps-$numdias-dias-$fechahoy.csv"
    }
    #llamada a la función que ejecuta la query a la BD
```

```
    $res_consulta = ConsultaTraps($query)  | Select-
Object "end_time","output","display_name"
    #se exporta el resultado de la consulta al fichero CSV modificando las cabeceras
de la tabla.
    $res_consulta | Select-Object end_time, display_name, output | Select-
Object @{Label='Fecha y hora'; Expression={$_.end_time}},@{Label='Host'; Expression={
$_.display_name}},@{Label='Trap'; Expression={$_.output}} | Export-
Csv $FicheroSalida -Delimiter ';'
    #se deshabilitan los botones y cajas de texto de la interfaz
    $TextoPregunta.Enabled = $false
    $TextoPregunta.Visible = $false
    $Opcion.Enabled = $false
    $Opcion.Visible = $false
    $Buscar.Enabled = $false
    $Buscar.Visible = $false
    $DataGridView1.Enabled = $True
    $DataGridView1.Visible = $True
    #se muestran los traps exportados en la rejilla de visualización y se da feedback
 al usuario
    PintarRejilla($res_consulta)
    $Label1.Text = "Estos son los traps exportados. Tu fichero CSV, se encuentra en:
$Ficherosalida"
}

<# Función que limpia los objetos de la interfaz cuando se realiza un cambio de secci
ón al pulsar los botones de la parte inferior del Form#>
function LimpiarInterfaz {
    $Label1.Text = ""
    $TextoPregunta.Enabled = $false
    $TextoPregunta.Visible = $false
    $Opcion.Enabled = $false
    $Opcion.Visible = $false
    $Buscar.Enabled = $false
    $Buscar.Visible = $false
    $Exportar.Visible = $false
    $Exportar.Enabled = $false
    $Dia.Visible = $false
    $Dia.Enabled = $false
    $Mes.Visible = $false
    $Mes.Enabled = $False
    $Anyo.Visible = $false
    $Anyo.Enabled = $false
    $TextoDia.Visible = $false
    $TextoDia.Enabled = $false
    $TextoMes.Visible = $false
    $TextoMes.Enabled = $false
    $TextoAnyo.Visible = $false
    $TextoAnyo.Enabled = $false
    $BuscaFecha.Visible = $false
```

```
    $BuscaFecha.Enabled = $false
    $dataGridView1.DataSource=$Null
}

<# A continuación la declaración de las acciones que desencadena
la pulsación de cada uno de los botones. Incluyen la llamada a las
funciones declaradas anteriormente. #>

        #Botón para salir del Visor
        $Salir.DialogResult  = [System.Windows.Forms.DialogResult]::OK

        #Botón para visualizar los últimos 20 traps
        $Ultimos20.Add_Click( {
            LimpiarInterfaz
            $Ultimos20.text = "Refrescar Traps"
            $DataGridView1.visible = $true
            $DataGridView1.enabled = $true
            VerUltimos20
        })

        #Botón para visualizar los traps de las últimas 24h
        $Traps24h.Add_Click( {
            LimpiarInterfaz
            $Ultimos20.text = "Ultimos 20 Traps"
            $DataGridView1.visible = $true
            $DataGridView1.enabled = $true
            Traps24h
        })

        #Botón para acceder a la búsqueda de traps por host
        $TrapsHost.Add_Click( {
            LimpiarInterfaz
            $Ultimos20.text = "Ultimos 20 Traps"
            $DataGridView1.visible = $false
            $DataGridView1.enabled = $false
            $TextoPregunta.Enabled = $true
            $TextoPregunta.Visible = $true
            $Opcion.Enabled = $true
            $Opcion.Visible = $true
            $Buscar.Enabled = $true
            $Buscar.Visible = $true
            $TextoPregunta.text = "Escribe un host del que deseas obtener los traps:"
            $Opcion.text = "Nombre del Host"
        })

        #Boton para la búsqueda de traps cuando se desea buscar traps por host
        $Buscar.Add_Click({
            $DataGridView1.visible = $true
            $DataGridView1.enabled = $true
```

```
    TrapsHost $Opcion.Text
})

#Boton para acceder a la búsqueda de traps por fecha
$TrapsFecha.Add_Click({
    LimpiarInterfaz
    $Label1.Text = "Busca por fecha con los siguientes parametros:"
    $Ultimos20.text = "Ultimos 20 Traps"
    $DataGridView1.visible = $false
    $DataGridView1.enabled = $false
    $Dia.Visible = $true
    $Dia.Enabled = $true
    $Mes.Visible = $true
    $Mes.Enabled = $true
    $Anyo.Visible = $true
    $Anyo.Enabled = $true
    $TextoDia.Visible = $true
    $TextoDia.Enabled = $true
    $TextoDia.Text = ""
    $TextoMes.Visible = $true
    $TextoMes.Enabled = $true
    $TextoMes.Text = ""
    $TextoAnyo.Visible = $true
    $TextoAnyo.Enabled = $true
    $TextoAnyo.Text = ""
    $BuscaFecha.Visible = $true
    $BuscaFecha.Enabled = $true
})

#Botón para la búsqueda desde la sección de búsqueda de traps por agente SNMP
$BuscaFecha.Add_Click({
    $DataGridView1.visible = $true
    $DataGridView1.enabled = $true
    TrapsFecha $TextoDia.Text $TextoMes.Text $TextoAnyo.Text
})

#Función para acceder a la sección exportación de históricos traps
$ExportarHistoricos.Add_Click( {
    LimpiarInterfaz
    $Label1.Text = ""
    $Ultimos20.text = "Ultimos 20 Traps"
    $DataGridView1.visible = $false
    $DataGridView1.enabled = $false
    $Opcion.Enabled = $true
    $Opcion.Visible = $true
    $Opcion.text = "Num. dias"
    $TextoPregunta.Enabled = $true
    $TextoPregunta.Visible = $true
    $TextoPregunta.text = "Numero de dias hacia atras (0 para todos traps): "
```

```
        $Exportar.Enabled = $true
        $Exportar.Visible = $true
    })

    #Botón para la búsqueda de históricos en base a una antigüedad configurada
    $Exportar.Add_Click({
        ExportarHistoricos $Opcion.Text
    })

# M A I N #
#se oculta la rejilla
$DataGridView1.visible = $false
$DataGridView1.enabled = $false
#Bienvenida al usuario
$Label1.text = "
Bienvenido al visor de traps, puedes navegar entre las diferentes opciones con los bo
tones que se encuentran en la parte inferior de la pantalla."
$Null = $form.ShowDialog()
```

Para una mayor facilidad de uso, se ha empaquetado el visor en un fichero binario (.exe), que funciona de manera portable haciendo doble click sobre él. Una vez se ejecuta el programa, se muestra el menú de opciones. En la Figura 38 pueden verse dos capturas que recogen el diseño de la interfaz de inicio y prestando servicio a través de alguna de sus opciones, respectivamente.

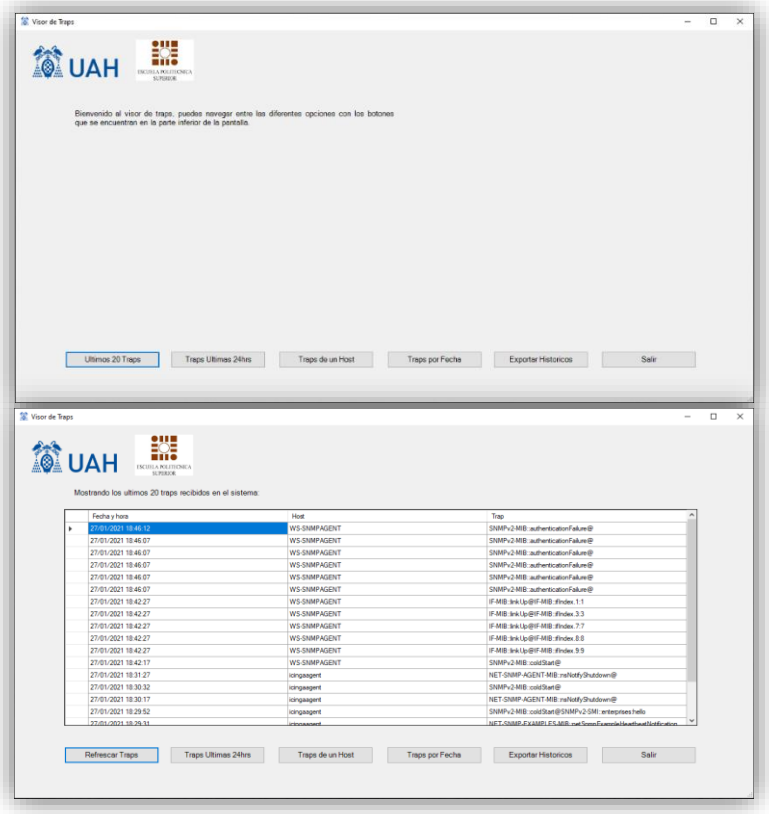

Figura 38. Interfaces del Visor de Traps propuesto como mejora.

5.3 Icinga sobre SSL

Esta mejora tiene como motivación principal securizar el tráfico http. Actualmente, cuando se consulta la web de Icinga, a pesar de que haya credenciales de acceso a la plataforma, los datos se transfieren sin ningún tipo de cifrado. Esto podría provocar que alguien con el conocimiento suficiente filtrase dicho tráfico y obtuviera la información que contiene sin demasiado problema. En consecuencia, se produce una brecha de seguridad en el sistema ya que, en muchas ocasiones, este tipo de herramientas tienen como cometido monitorizar elementos que se encuentran en entornos críticos, y cuya situación requiere de cierto nivel de confidencialidad.

No obstante, y aunque durante el despliegue de Icinga no se ofrece la posibilidad de funcionar sobre HTTPS, el hecho de que esta herramienta funcione sobre servidores web como *Apache* o *Nginx* facilitan mucho este tipo de mejoras.

El hecho de cifrar el tráfico web no sólo consiste en colgar un certificado del servidor HTTP, ya que, si el certificado no se considera fiable por el equipo cliente, aparecerá en el navegador del usuario un mensaje que le notificará (con bastante criterio) que el certificado en uso no es seguro porque no se conoce la entidad firmante. Es por esto que existen ciertas autoridades consideradas fiables, las cuales expiden certificados SSL firmados por lo que se conoce como un "CA Root" (certificados raíz de confianza). Muchos de estos certificados "CA Root" vienen incorporados en nuestros navegadores para que, cuando se visite una web (u otro servicio) que utilice un certificado SSL firmado por estas entidades, nuestro equipo confíe directamente en ese destino.

Por lo tanto, la solución más sencilla para llevar a cabo esta mejora sería solicitarle a uno de estos organismos un certificado SSL para esta plataforma. Sin embargo, muchas de estas instituciones no permiten escoger entre demasiados tipos de cifrado. A veces tampoco es un proceso inmediato, y conllevan un coste asociado a factores como la validez del certificado o sus características de seguridad. Por ello, y como también es frecuente que muchas compañías utilicen su propio certificado *CA Root* para crear certificados de uso privado, se va a crear un juego de certificados (*CA Root* y SSL) para llevar a cabo esta mejora, algo que se traduce en una seguridad adicional y en un ahorro en materia de costes. Este proceso se apoya en la *suite* de herramientas de código abierto OpenSSL[23], disponible para muchos sistemas. En varias distribuciones de Linux viene instalada por defecto. Su licencia permite el uso de esta herramienta para uso comercial y personal, por lo que tampoco se incumpliría ninguna cláusula en caso de utilizarse en un entorno profesional.

[23] https://www.openssl.org/

Por lo tanto, el primer paso del proceso reside en la creación de un certificado *CA Root* que pueda firmar certificados SSL. Como cualquier certificado, lleva previamente asociada la creación de una clave privada. Cabe destacar que, aunque el *CA Root* puede permanecer importado en los equipos clientes de miles de personas, esta clave únicamente la posee la entidad certificadora. De este modo, nadie que no sea parte de este organismo podría firmar certificados SSL con el certificado raíz. Por esta razón, es conveniente que dicha clave privada tenga cierto nivel de seguridad cuando se crea un nuevo *CA Root*, a fin de evitar que alguien pueda dar con la contraseña y hacerse pasar por una autoridad.

Teniendo en cuenta lo anterior, el primer paso para crear el certificado con OpenSSL será la creación previa de un fichero *.key*:

```
# openssl genrsa -des3 -out ROOTCA.key 2048
```

En esta instrucción se especifica el algoritmo de cifrado y el tamaño de la clave (en bits). Acto seguido a su ejecución, el sistema solicita al usuario una palabra con la que se cifrará el fichero de clave. Una vez introducida dos veces, el sistema deposita el fichero en el directorio actual. Ya con la clave disponible, el siguiente paso es crear el certificado raíz, para ello se introduce este nuevo comando:

```
# openssl req -x509 -new -nodes -key ROOTCA.key -sha256 -days 1825 -out
ROOTCA.pem
```

Nuevamente se especifica el algoritmo de cifrado, así como el nombre del fichero *.key* creado en el paso anterior, el cual vinculará inequívocamente el certificado con su clave privada. Adicionalmente, debe indicarse la validez de dicho certificado en días. Acto seguido, OpenSSL solicita al usuario la introducción de ciertos parámetros que se alojarán en el *CA Root* para que cualquiera pueda ver su procedencia. A continuación, se ofrece la salida de consola con la que se ha configurado el certificado (los fragmentos en rojo son los introducidos manualmente):

```
Enter pass phrase for ROOTCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
```

```
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Guadalajara
Locality Name (eg, city) []:Guadalajara
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UAH
Organizational Unit Name (eg, section) []:Monitorización
Common Name (e.g. server FQDN or YOUR name) []rootcamonitorizacion.intranet.es
Email Address []:santiago.marcos@edu.uah.es
```

Como puede apreciarse, lo primero que solicita OpenSSL es la contraseña introducida durante la creación de la clave privada, a fin de verificar que somos el autor de dicha clave. Una vez se han completado los campos solicitados, la herramienta deposita el fichero *.pem* (que es el que contiene el certificado) en el directorio de trabajo y ya estaría disponible el certificado raíz que, junto con la clave privada, nos permitiría firmar certificados SSL.

Por lo tanto, el siguiente paso es la creación del certificado SSL para la web de Icinga. Al igual que en el caso anterior, todo parte de la creación de un nuevo fichero de clave privada que sólo tendrá el servidor HTTP donde se ubique el certificado para que, aunque alguien sustraiga el certificado de la web, no pueda hacerse pasar por ese destino creando un clon malicioso que suplante la identidad. Esta tarea se realiza a través del siguiente comando:

```
# openssl genrsa -out icingassl/icinga.key 2048
```

Para el almacenaje ordenado de los elementos se ha creado un directorio llamado *icingassl*, en el cual OpenSSL deposita el fichero de clave.

Cuando se solicita a una entidad raíz de confianza la creación de un nuevo certificado, hay un requisito para su obtención: el envío por parte del solicitante de un CSR (*Certificate Signing Request*), que no es más que un fichero en el que se encuentra toda la información necesaria que contendrá el certificado una vez se expida, y en el que debe aparecer el nombre exacto que tendrá la herramienta donde se instale el certificado, *Common Name* (CN). Por ello, en este caso también habrá que crear un CSR, aunque seamos nosotros mismos los que validemos su información. El siguiente comando servirá para realizar este paso:

```
# openssl req -new -key icingassl/icinga.key -out icinga.csr
```

Como puede observarse en esta orden, se debe indicar la ubicación del fichero de clave privada así como el nombre que tendrá el fichero de salida (CSR). Tras esto se solicita al usuario la

introducción manual de los parámetros mencionados. En compañías donde es algo muy frecuente, este proceso puede encontrarse automatizado mediante scripts que lo hagan de forma más automatizada. Se ofrece aquí al lector la sucesión de parámetros introducidos:

```
…
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Guadalajara
Locality Name (eg, city) []:Guadalajara
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UAH
Organizational Unit Name (eg, section) []:Monitorización
Common Name (e.g. server FQDN or YOUR name) []:icingasnmp.intranet.es
Email Address []:santiago.marcos@edu.uah.es

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:******
An optional company name []:UAH
```

Una vez introducidos los campos, se obtiene el fichero *icinga.csr y*, después, se enviaría a la entidad certificadora para que ésta expida el certificado correspondiente. No obstante, antes de su obtención, el organismo que lo expide haría probar al solicitante que el dominio *icingasnmp.intranet.es* es de su posesión (lo que lleva asociado otro trámite, bien haciendo colgar del propio servidor web un fichero, o bien haciendo uso de un correo electrónico que sea parte de ese dominio, entre otras técnicas). En equipos que se encuentran en redes aisladas, esto puede suponer un problema, dado que la entidad tendría más complicaciones para verificar el dominio. Por ello, realizar este proceso con certificados auto-firmados es otra ventaja.

El siguiente paso para la creación del certificado SSL es un proceso que realizaría internamente la entidad firmante. Consiste en la creación de un fichero de configuración que especifica los parámetros y condiciones para los que será válido el certificado. A continuación se presenta un ejemplo de de este proceso que enmarcará la validez del futuro certificado SSL (especificación del FQDN, la firma digital…). Pueden encontrarse algunos ejemplos similares en internet que ofrecen diferentes combinaciones. El fichero creado a continuación recibe el nombre de *config_cert.ext*:

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment,dataEncipherment
```

```
subjectAltName = @alt_names
[alt_names]
DNS.1 = icingasnmp.intranet.es
```

Por último, y ya en posesión de todo lo anterior: el CSR, el Certificado CA Root, su clave privada y el fichero de configuración, puede llevarse a cabo la expedición del certificado SSL de Icinga con la siguiente orden:

```
# openssl x509 -req -in icingassl/icinga.csr -CA ROOTCA.pem -CAkey ROOTCA.key
-CAcreateserial -out icingassl/icingasnmp.intranet.es.crt -days 1825 -sha256
-extfile icingassl/config_cert.ext
```

Como puede verse en el comando anterior, se indica el mecanismo de cifrado, la ubicación de los ficheros necesarios para su creación, la validez del certificado SSL y el nombre de salida que tendrá el fichero del certificado. Una vez realizado todo ello, OpenSSL mostrará los parámetros introducidos en el CSR y solicitará la introducción de la clave privada del certificado *CA Root*, creada al inicio de este apartado:

```
Signature ok
subject=C = ES, ST = Guadalajara, L = Guadalajara, O = UAH, OU =
Monitorizaci\C3\83\C2\B3n, CN = icingasnmp.intranet.es, emailAddress =
santiago.marcos@edu.uah.es
Getting CA Private Key
Enter pass phrase for ROOTCA.key: ******
```

En este punto ya se dispone del juego de certificado y clave necesario para integrar en el servicio de apache. No obstante, antes de afrontar esta parte, no hay que olvidar que *icingasnmp.intranet.es* es un nombre de dominio, por lo que habrá que dar una entrada *Host* de tipo A en el servidor DNS de la red donde se ubiquen los clientes que utilizarán Icinga. En el caso de este supuesto, al ser un entorno de pruebas y ante la ausencia de un rol de DNS, se ha considerado oportuno introducir esta entrada en el fichero *hosts* del equipo que visitará la web. En Windows, dicho fichero se ubica en *C:\Windows\System32\drivers\etc*.

Una vez llevadas a cabo todas las acciones previas, se debe habilitar en apache el módulo que permite trabajar con SSL, en caso de no estar habilitado. Esto puede hacerse a través de la orden:

```
# a2enmod ssl
```

Se deben ubicar los dos ficheros (certificado SSL y clave privada) en una ubicación a la que apache pueda acceder para recuperarlos. En este caso se han depositado en un subdirectorio de la carpeta */home/administrador*. Después, se lleva a cabo la creación de un nuevo fichero de configuración de apache (ubicado por defecto en */etc/apache2/sites-enabled*), que se ha llamado *icingassl.conf* y que contendrá el *VirtualHost* que redirigirá el tráfico que iba al puerto 80 hacia la URL de Icinga por el puerto 443. El contenido del fichero de configuración contiene lo siguiente (se marcan en negrita los fragmentos relevantes):

```
<VirtualHost *:80>
      Redirect "/" "https://icingasnmp.intranet.es/icingaweb2"
</VirtualHost>
<IfModule mod_ssl.c>
      <VirtualHost 192.168.1.130:443>
            ServerAdmin santiago.marcos@edu.uah.es
            ServerName icingasnmp.intranet.es
            DocumentRoot /var/www/html
            ErrorLog ${APACHE_LOG_DIR}/error.log
            CustomLog ${APACHE_LOG_DIR}/access.log combined
            SSLEngine on
            SSLCertificateFile
/home/administrador/certificados/icingassl/icingasnmp.intranet.es.crt
            SSLCertificateKeyFile
/home/administrador/certificados/icingassl/icinga.key
            <FilesMatch "\.(cgi|shtml|phtml|php)$">
                  SSLOptions +StdEnvVars
            </FilesMatch>
                  <Directory /usr/lib/cgi-bin>
                        SSLOptions +StdEnvVars
                  </Directory>
                  BrowserMatch "MSIE [2-6]" \
                  nokeepalive ssl-unclean-shutdown \
                  downgrade-1.0 force-response-1.0
      </VirtualHost>
</IfModule>
```

Ahora se debe reiniciar el servicio de apache2 para importar el nuevo funcionamiento en la configuración. El proceso aún no habría acabado del todo, dado que el equipo cliente aún no confía en el certificado SSL que expondrá el servidor de Icinga, dado que no conoce su entidad certificadora raíz de confianza. Para establecer esta confianza, el equipo cliente debe instalar el

fichero PEM que contiene el *CA Root*. Esto puede hacerse de forma sencilla depositando el certificado en algún directorio de dicho equipo e importándolo en el almacén de Entidades Certificadoras Raíz de Confianza de Windows (recomendable hacerlo a nivel de equipo, no de usuario, para que se aplique la configuración por igual a todos los usuarios locales de esa estación de trabajo) o, también, a través de la siguiente instrucción de *Powershell* (haciéndolo como administrador del equipo local):

```
> Import-Certificate .\ROOTCA.pem -CertStoreLocation
'Cert:\LocalMachine\Root'
```

Una vez hecho esto, el navegador del equipo cliente confiará en el certificado expedido para Icinga[24], y nos indicará que la conexión es segura, tal y como muestra la Figura 39:

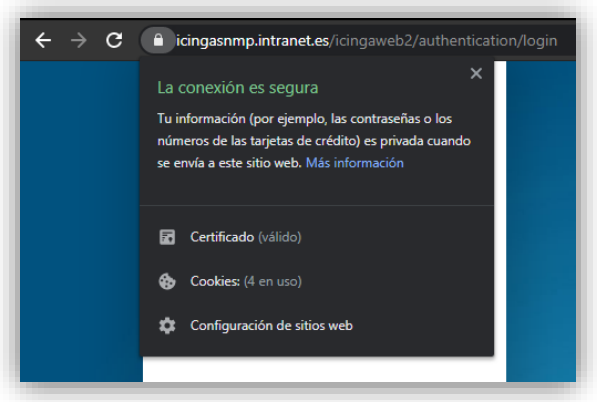

Figura 39. Muestra de confianza de Google Chrome para el certificado SSL de Icinga.

Como puede verse en la Figura 40, si se hace click sobre "Certificado", se ofrece la ruta de certificación por la cual se ha procedido a la firma:

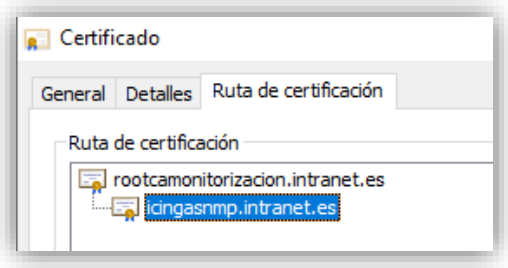

Figura 40. Ruta de certificación del certificado SSL para Icinga.

A partir de este momento, todo el tráfico web generado por navegación a través de Icinga irá cifrado mediante HTTPS, impidiendo la filtración de información y elevando la seguridad del entorno.

[24] En caso de utilizar Mozilla Firefox puede ser necesario importar el certificado raíz en el almacén de dicho navegador.

5.4 Supervisord

Esta última mejora del entorno se origina por un análisis proveniente de la alta disponibilidad de este tipo de herramientas (hablando en general de las plataformas de monitorización). Herramientas como Icinga existen para dar visibilidad del estado de elementos remotos ya que, como se explicó en el apartado 1.1, hacerlo de manera individualizada es cuanto menos impensable dadas las cantidades de sistemas desplegados en muchos entornos. Por ello, en caso de que alguno de los servicios que componen esta plataforma tuviera una parada, afectaría no a uno de los elementos bajo supervisión, sino a toda la infraestructura, lo que desembocaría en la detención de la monitorización; en una ceguera de todo lo que puede estar pasando.

En el supuesto de que la máquina virtual completa sufriera una pérdida de servicio, actualmente existen multitud de medidas para paliar este escenario: copias de seguridad del entorno virtualizado, replicación de servicios o de recursos virtuales, etc. Sin embargo, existe la posibilidad de que haya un fallo granular, es decir, un fallo de alguno de los procesos que llevan a término la monitorización, y esto no lo cubren ese tipo de soluciones. Por ejemplo, una copia de seguridad de la máquina virtual albergaría también esa vulnerabilidad ubicada en el equipo original. En este supuesto, se estaría hablando de una caída de un servicio provocada, por ejemplo, por una saturación de recursos ante un excesivo número de solicitudes de algún tipo.

En un entorno como el que se ha venido configurando, el servicio de Icinga no sería tan vulnerable a este aspecto, ya que la cola de procesamiento de Nagios tiene mecanismos de control para evitar el colapso si, por ejemplo, se mandan desde el servidor muchas solicitudes SNMP a los *hosts* remotos. Sin embargo, procesos como la escucha de *traps* podrían ser susceptibles a este respecto, dado que pueden recibirse muchos *traps* simultáneos, sobre todo en caso de que el número de elementos remotos sea muy grande. Es más, la caída de este servicio sería sumamente silenciosa ya que, al ser un servicio pasivo, el personal no sabría si la ausencia de notificaciones se debe a un correcto funcionamiento o a una parada del demonio de *traps*. Obviamente, una solución que se podría adoptar es monitorizar que exista en ejecución una instancia de este proceso, lo cual podría ser otra mejora adicional. No obstante, se ha considerado llevar a cabo un refuerzo del propio servicio de recepción de *traps* ya que, ante un fallo puntual, el servicio sería restablecido de forma automática aunque el personal destinado a la monitorización no se encuentre disponible en ese instante. En cualquier caso, podrían adoptarse ambas soluciones, ya que una sería mejora complementaria de la otra (validar que exista el proceso en ejecución, frente a proteger la ejecución de ese servicio mediante algún mecanismo), no excluyentes entre sí.

Para llevar a cabo esta corrección se dispone de *Supervisor*[25], también conocido por su nombre interno, *supervisord*, sistema que permite controlar procesos alojados en sistemas basados en Unix. Cuenta con varias funciones, siendo una de ellas la de iniciar una instancia a modo de subproceso de un programa de nuestra elección. Ese papel lo representa el proceso que sería susceptible de sufrir una caída (en este supuesto, el servicio que desea reforzarse es *snmptrapd*) y que será comprobado por *supervisord* continuamente. En el caso de que el primero muera, el segundo se encargaría de levantar una nueva instancia del mismo según una configuración previa, que se verá más adelante.

Supervisor está disponible en los gestores de paquetes de muchas distribuciones de Linux. Sin embargo, desde la documentación oficial se advierte que estas adaptaciones están llevadas a cabo por terceros y que, en muchas ocasiones, podrían no incorporar los mecanismos integrados en las últimas actualizaciones. Por este motivo, para realizar la presente mejora se ha seguido la guía de configuración oficial. Para instalarlo, se necesitará disponer del instalador de paquetes de Python. Con este fin, se ejecutan los siguientes comandos:

```
# sudo apt-get update
# sudo apt-get install python3-pip
```

Una vez concluya el proceso anterior, ya puede instalarse el paquete asociado a *Supervisor*:

```
# pip3 install supervisor
```

Tras lo anterior, puede procederse con la configuración de la herramienta. Una de las maneras de comenzar y entender las posibilidades que ofrece este paquete de herramientas es generar un fichero de configuración, a modo de ejemplo, que pueda ser retocado a posteriori para adaptarlo a las necesidades del supuesto. Este fichero de configuración puede ser generado invocando el ejecutable de ayuda para creación de ficheros y redirigiendo la salida con el siguiente comando:

```
# echo_supervisord_conf > /etc/supervisord.conf
```

El siguiente paso será identificar con qué parámetros tiene que ejecutarse el proceso de la escucha de *traps* para seguir funcionando de manera correcta. Para ello, se puede consultar el fichero donde está la configuración de arranque por defecto del servicio, o bien filtrar en los procesos

[25] http://supervisord.org/

activos los parámetros con los que se ejecuta actualmente, tal y como se indica con la ejecución de la siguiente orden compuesta:

```
# ps -ax | grep snmptrapd
 1367 ?        S      0:00 /usr/sbin/snmptrapd -Lsd –f
```

Una vez se determinan los parámetros con los que se ejecuta el demonio de *traps*, conviene detener el servicio y deshabilitarlo del inicio. El objetivo es evitar que empiece de forma autónoma sin ser lanzado por *Supervisor*, lo que provocaría una ejecución simultánea, en el próximo reinicio. Esta tarea puede hacerse con los siguientes comandos:

```
# systemctl stop snmptrapd
# systemctl disable snmptrapd
```

Ya podría modificarse el fichero de configuración de *supervisord*, descartando las otras funcionalidades sugeridas por el fichero de ejemplo obtenido previamente (mediante la invocación a *echo_supervisord_conf* y la redirección). La configuración quedaría de la siguiente forma:

```
[program:snmptrapd]
    command= /usr/sbin/snmptrapd -Lsd -f

[supervisord]
    priority=999
    autostart=true
    autorestart=true
    startsecs=0
    startretries=3
    stopsignal=TERM
    stopasgroup=false
    killasgroup=true
    user=root
    redirect_stderr=false
    stdout_logfile=/home/administrador/snmptrapd.log
    stdout_logfile_maxbytes=10MB
    stdout_logfile_backups=10
    stdout_capture_maxbytes=1MB
    stderr_logfile_maxbytes=1MB
    stderr_logfile_backups=10
```

```
stderr_capture_maxbytes=1MB
stderr_events_enabled=false
serverurl=AUTO
```

Como puede comprobarse, el parámetro *command* sería la orden que lanzaría *Supervisor* tanto para iniciar como para rehabilitar el proceso de escucha de *traps*. El resto de parámetros que se encuentran en la segunda parte hacen referencia a aspectos de configuración adicional tales como: el número de reintentos para ejecutar el comando en caso de que no sea posible en el primer intento, la rotación de fichero de *log*, su ubicación, etc.

Una vez se completa la modificación del fichero, queda iniciar el proceso de recepción de *traps* a través de *supervisord*. Lo más apropiado sería automatizar esta función para que, tras los reinicios de la máquina virtual, la ejecución de *Supervisor*, se inicie de manera desatendida. Para lograrlo se ha creado un script de lanzamiento y se ha asociado a un nuevo servicio. Puede verse a continuación el contenido de dicho script con la orden de invocación a *supervisord* seguida de la ubicación del fichero de configuración creado como parámetro de entrada:

```
#!/bin/bash
supervisord -c /etc/supervisord.conf
```

Una vez se ha realizado el script, se le otorgan permisos de ejecución y se crea el demonio al que irá asociado. Para esto, se ha dispuesto un fichero llamado *supervisord.service* localizado en */etc/systemd/system/*, donde el sistema espera encontrar nuevos servicios. El contenido de este archivo sería el siguiente:

```
[Unit]
Description=DaemonServSupervisord
[Service]
Type=forking
ExecStart=/home/administrador/start_supervisord.sh
WorkingDirectory=/home/administrador
User=root
[Install]
WantedBy=multi-user.target
```

En la configuración aquí expuesta se puede observar la llamada al script, una descripción del servicio, y el usuario con el que se lanzará dicho proceso, entre otros parámetros. Una vez se

guarda el fichero, se le otorgan permisos de ejecución y se recargan los servicios disponibles del sistema para, posteriormente, habilitar su inicio automático con los siguientes comandos:

```
# systemctl daemon-reload
# systemctl enable supervisord.service
```

Ya puede iniciarse el nuevo servicio y comprobarse que se encuentra en ejecución:

```
# service supervisord status
● supervisord.service - DaemonServSupervisord
   Loaded: loaded (/etc/systemd/system/supervisord.service; enabled; vendor
preset: enabled)
   Active: active (running) since Sat 2021-01-12 18:36:30 CET; 1 day 3h ago
  Process: 1188 ExecStart=/home/administrador/supervisord-snmptrapd.sh
(code=exited, status=0/SUCCESS)
 Main PID: 1325 (supervisord)
    Tasks: 2 (limit: 4659)
   CGroup: /system.slice/supervisord.service
           ├─1325 /usr/bin/python3 /usr/local/bin/supervisord -c
/etc/supervisord.conf
           └─1728 /usr/sbin/snmptrapd -Lsd -f
```

No obstante, la mejor manera de probar que el sistema funciona correctamente es intentando matar el proceso asociado a *snmptrapd* forzosamente a través de su PID. Para ello, se vuelve a ejecutar `ps -ax | grep snmptrapd` y se mata el proceso con el siguiente comando:

```
# kill -9 1728
```

Posteriormente, puede comprobarse que hay en ejecución una nueva instancia del proceso con un PID diferente ejecutando el siguiente comando compuesto:

```
# ps -ax | grep snmptrapd
 1832 ?        S      0:00 /usr/sbin/snmptrapd -Lsd –f
```

Capítulo 6

Conclusiones

En este apartado se pretende hacer referencia a aquellos aspectos reseñables, tanto ventajas como inconvenientes, hallados durante el estudio del protocolo SNMP, en la implantación de Icinga y su desempeño mediante dicho protocolo, así como en la construcción de las mejoras al alcance.

Respecto al primero de los aspectos, se ha encontrado sumamente interesante el estudio del protocolo, dado que a pesar de su antigüedad sigue teniendo su hueco en un mundo donde la monitorización de recursos juega un papel importante. Por esta razón, su conocimiento puede ayudar a comprender algunos de los procesos de supervisión vigentes aún a día de hoy en muchos entornos. No obstante, sorprende que, a pesar de los avances producidos en su implementación, aún exista cierta reticencia por parte de los desarrolladores y fabricantes en integrar la última versión, ya que durante la fase de estudio se han descargado manuales y documentación de algunos dispositivos que integrarían MIB's cuya versión de protocolo está, en su mayoría, asociada a la segunda versión de SNMP, a pesar de que la última cuenta ya con bastante recorrido.

Una de las ventajas más claras que se han encontrado durante el proceso de despliegue de Icinga es que existe multitud de información asociada a la aplicación, ya que al ser algo bastante extendido, derivado de Nagios y que, además, cuenta con una licencia de código abierto, son muchos los que se suman a colaborar y ofrecer sus conocimientos a la comunidad. Adicionalmente, se ha encontrado en Icinga un sistema robusto y con muchas capacidades, incluso en términos de flexibilidad, lo que se traduce finalmente en la posibilidad de cubrir distintos tipos de necesidades, algunas incluso muy específicas, con un mismo producto. Asimismo, este proceso ha suscitado un interés en ahondar en las capacidades que ofrece, tanto éste como otros sistemas y estrategias de monitorización.

Sin embargo, se ha encontrado alguna dificultad en términos de incompatibilidad con versiones de los paquetes especificados en los prerrequisitos que, a pesar de haber cumplido los mínimos detallados en la documentación, han dificultado inicialmente la experiencia, ya que algunos errores inicialmente achacados al funcionamiento no se debían a una configuración incorrecta, sino a un despliegue con versiones incompatibles. También se ha hecho notar la falta de algunas

opciones adicionales durante la configuración inicial de la herramienta. Sorprende especialmente la ausencia relacionada con SSL o el cambio de puertos, políticas de contraseñas, etc. de forma nativa, si bien algunas de estas opciones han podido cubrirse mediante mejoras, a posteriori.

Una de las posibles mejoras de Icinga podría ser la incorporación de un entorno más amigable para la edición de configuración, la cual informase de los fallos de sintaxis o de especificación durante la edición de los ficheros. Asimismo, otro elemento que necesitaría ser mejorado es el compilador del que hace uso el motor de Icinga para validar las modificaciones de sus ficheros, ya que, a menudo ofrece información confusa acerca de los errores detectados durante la validación de ficheros. Un ejemplo de ello es que, durante la creación del *command*, Icinga parece indicar que hay una recursión infinita en la llamada a la operación cuando en realidad había un parámetro mal nombrado. Por ello, un *feedback* más específico mejoraría la experiencia del administrador.

En lo relacionado con las mejoras introducidas en el Capítulo 5, y esto converge con lo mencionado acerca de la flexibilidad de Icinga, se ha podido comprobar de forma empírica que muchas de las ideas que surgen en base a una necesidad detectada tienen bastantes posibilidades de ser implementadas en el entorno, aunque haya que apoyarse en herramientas y paquetes de terceros para llevarlo a término. Lo óptimo sería poder utilizar módulos nativos de Icinga para tal finalidad, algo que podría cambiar en un futuro a medida que la herramienta vaya ganando notoriedad y expansión. No obstante, algunas de las mejoras, como el demonio de *traps*, han supuesto un reto en materia de dificultad, si bien, parte del proceso de integración consiste en familiarizarse con el entorno, el lenguaje y la lógica de sus componentes. A medida que se completa la curva de aprendizaje, algunas de las dificultades iniciales son suavizadas y, en general, queda la visión de que es una gran solución para este tipo de supuestos, así como de otros muchos que pudieran plantearse. Adicionalmente, se ha encontrado en *Powershell* una herramienta que ofrece muchas posibilidades y que no entraña una dificultad excesiva para obtener unos resultados de calidad, a pesar del desconocimiento inicial del que se partía.

En lo concerniente al cumplimiento de objetivos, se ha cubierto la puesta en marcha de una herramienta funcional que aplicase los conocimientos introducidos durante la fase del análisis teórico y de la documentación, obteniendo como resultado una herramienta que podría ser perfectamente adaptable a un entorno real con estas necesidades, más aún contando con que el proyecto se encuentra desplegado en una máquina virtual. Dicho entorno podría ser desplegado de manera sencilla y accedido por el personal destinado a la monitorización de forma segura, simplemente con un navegador web. Otra de las ventajas añadidas es que toda la configuración interna de la plataforma puede ser realizada en remoto.

Capítulo 7.

Líneas de Trabajo Futuro.

Desde una perspectiva de ampliación e investigación, el proyecto expuesto en esta memoria podría ser continuado en base a los siguientes puntos:

- Nuevas mejoras que permitieran dotar al entorno de una mayor funcionalidad desde un punto de vista de navegación web. La parte Web de Icinga está construida con PHP, lo que facilita enormemente la creación de nuevas secciones o mejorar partes ya existentes. Algunas de ellas podrían ser a nivel de seguridad. Por ejemplo, la inclusión de una política de contraseñas, o el uso de "Olvidé mi contraseña" dando de alta un *e-mail* por cada usuario.

- Sería interesante el desarrollo de nuevos módulos instalables de Icinga que añadan funcionalidades a la plataforma tales como la gestión de tareas de supervisión rutinarias.

- A nivel de implantación de recursos vía SNMP, sería interesante introducir algún aspecto más de la versión 3 del protocolo y comprobar cómo se comportaría la plataforma, analizando su compatibilidad, así como los aspectos diferenciadores que separan a la última versión de sus predecesoras. En esta línea, por ejemplo, contemplar la recepción de *traps* a través de la nueva versión a fin de comprobar si es necesaria la incorporación de nuevas medidas.

- El desarrollo de una MIB que pudiera ser utilizada por algún *software* o sistema que pueda ser añadido más tarde como agente SNMP en esta u otra infraestructura.

- Exploración de otras estrategias diferentes de monitorización de recursos (tanto pasivas como activas) como NSCA, u otras que puedan ser una alternativa y que permitan realizar una análisis comparativo.

- Explorar la capacidad de integración de Icinga con el directorio activo y su cabida en el mundo profesional. Asimismo, se propone la inclusión de una infraestructura de mayor tamaño, propuesta en la documentación de Icinga, mediante el uso de los *satellites* de Icinga. En ella podría haber diferentes plataformas desplegadas de forma distribuida,

centralizando todos los datos en un único lugar. Esto es algo realmente útil cuando se cuenta, por ejemplo, con varias sedes.

- Un desarrollo de clientes pesados de Icinga, de escritorio o para móviles, que permitan consultar los datos e interactuar con la herramienta de manera distinta, a fin de facilitar su manejo con otro tipo de dispositivos. Algo similar a lo que implica el Visor de Traps propuesto en el capítulo 5. Para este u otros propósitos, sería interesante explotar las capacidades que ofrece la API de Icinga.

En general, podría decirse que este proyecto parte de una línea de aprendizaje acerca de la monitorización de recursos, cuya variedad de posibilidades ofrece un gran abanico de vías por las que transcurrir.

Parte III

Apéndices

Apéndice A.

Despliegue de agentes en Linux y Windows, y su configuración en Icinga.

En este primer apéndice se pretende ofrecer un punto de vista con algo más de detalle del procedimiento a seguir para monitorizar equipos cuyo sistema operativo sea Windows o Linux. Sin duda, ésta es una de las necesidades que estarían más extendidas, dada su presencia en el ámbito profesional). Dicho proceso será el equivalente al mostrado en el Capítulo 4, donde se detallaba este proceso para un agente SNMP que funcionaba sobre Android.

Cabe destacar que el hecho de incluir un nuevo *host* en la infraestructura no implica la instalación de un agente *per se*, ya que muchos de los fabricantes integran el agente en el propio *firmware* del dispositivo (es el caso de equipos destinados a la gestión de la red, impresoras, NAS, etc.). Lo único que haría el administrador en este caso sería habilitar el servicio, configurar los parámetros de seguridad de SNMP y familiarizarse con la MIB que detalla los parámetros que puedan resultar de interés. Una vez realizado esto, el último paso sería seguir el procedimiento mostrado en el Capítulo 4 para la creación de *services*, umbrales, etc.

En el caso de los sistemas Linux y Windows, el proceso tiene el añadido de la instalación/despliegue del propio agente, además de los pasos descritos. Por lo tanto, este apéndice se dividirá en dos apartados que muestren dicha configuración por separado.

Configuración de un agente en Linux

En este supuesto, el punto de partida sería nuevamente una instalación limpia de Ubuntu Server 18.04. En estos sistemas, el agente SNMP tiene la estructura de un demonio; el código referente a su ejecución pertenece al paquete de software NET-SNMP y para su despliegue lo primero que debe hacerse es instalar el paquete donde se encuentra dicho demonio, tal y como se haría para cualquier otro, con los siguientes comandos:

```
$ sudo apt update
$ sudo apt install snmpd
```

Una vez completado el proceso de instalación del paquete, lo siguiente sería dirigirse al directorio de configuración y editar el fichero *snmpd.conf* que permite ajustar los parámetros del agente (por defecto se encuentra en */etc/snmp/snmpd.conf*). De manera predeterminada, el agente instalado en este servidor sólo escucha las peticiones SNMP procedentes de sí mismo (*localhost*), por lo que dentro del fichero habrá que comentar la línea donde se especifica esta configuración y quitar el comentario donde se encuentra la configuración que permite la interrogación a través de SNMP desde dispositivos remotos, quedando el fichero de la siguiente manera:

```
…
#agentAddress udp:127.0.0.1:161
agentAddress udp:161,udp6:[::1]:161
…
```

Desde este fichero se puede configurar tanto la política de seguridad como el uso de comunidades diferentes para propósitos de lectura/escritura, o la autenticación en el caso de utilizar SNMPv3. En cualquier caso, y para en entorno de pruebas como el del supuesto actual, se mantendrá la comunidad "public" para ambos propósitos.

Adicionalmente, se permite la configuración de parámetros referentes al uso de recursos, como serían la CPU o el disco (pertenecientes a los parámetros ofrecidos por la MIB: UCD-SNMP-MIB). Por último, en el fichero figura la configuración de *traps*. En este supuesto, se quedan habilitados los de la versión 2c apuntando al servidor de Icinga (que es la IP que aparece en el fragmento siguiente):

```
…
# trapsink     localhost public
trap2sink    192.168.1.130 public
#informsink   localhost public
…
```

Finalmente, se debe reiniciar el servicio de *snmpd* para aplicar los cambios y permitir los puertos asociados a SNMP a través del *firewall* con los siguientes comandos:

```
sudo service snmpd restart
```

```
sudo ufw allow 161/udp
sudo ufw allow 162/udp
```

Tras esto, y para familiarizarse con los parámetros que ofrece, es conveniente lanzar desde algún equipo un comando *snmpwalk* apuntando al nuevo agente SNMP, el cual muestre las posibilidades que ofrece a nivel de datos monitorizables. En este punto sería útil familiarizarse con la MIB, al igual que se hizo en el estudio efectuado en el Capítulo 4, para detectar aquellos OIDs que puedan resultar de relevancia:

```
snmpwalk -v2c -c public 192.168.1.126
```

Una vez completado todo el proceso de exploración, se crearían los recursos en Icinga. Estos mantienen una estructura similar a los que se han podido ver anteriormente, tal como se puede apreciar en la configuración siguiente:

- **Commands.conf:**

  Se utilizan los mismos *commands* diseñados en los Capítulos 4 y 5.

- **Groups.conf:**

```
object HostGroup "agente_linux"{
  display_name = "Agentes Linux"
  assign where host.vars.grupo == "agente_linux"
}
```

- **Templates.conf:**

```
template Host "agente_linux" {
  max_check_attempts = 3
  check_interval = 5m
  retry_interval = 5m
  /*check_command = "snmp"*/
  check_command = "hostalive"
  vars.snmp_com = "public"
  vars.snmp_vers = "2c"
  vars.snmp_puerto = "161"
  vars.servicios_definidos=[""]
```

```
}
```

- **Servicios.conf**:

```
apply Service "uptime"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "agente_linux"
  vars.snmp_oid = "1.3.6.1.2.1.1.3.0"
  check_interval=10m
  retry_interval=3m
  max_check_attempts=3
  vars.snmp_warning = "8640000:"
  vars.snmp_critical = ""
}

apply Service "DescripcionSistema"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "agente_linux"
  vars.snmp_oid = "1.3.6.1.2.1.1.1.0"
  check_interval=100m
  retry_interval=5m
  max_check_attempts=3
  vars.snmp_warning = ""
  vars.snmp_critical = ""
}

apply Service "PorcentajeDiscoUsado"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "agente_linux"
  vars.snmp_oid = ".1.3.6.1.4.1.2021.9.1.9.1"
  check_interval=100m
  retry_interval=5m
  max_check_attempts=3
  vars.snmp_warning = "@60:80"
  vars.snmp_critical = "~:80"
}

apply Service "Ram"{
```

```
  import "generic-service"
  check_command = "snmp_walk"
  assign where host.vars.grupo == "agente_linux"
  vars.snmp_oid = ".1.3.6.1.4.1.2021.4"
  check_interval=100m
  retry_interval=5m
  max_check_attempts=3
  vars.snmp_warning = ""
  vars.snmp_critical = ""
}

apply Service "CPULoad5min"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "agente_linux"
  vars.snmp_oid = "1.3.6.1.4.1.2021.10.1.3.2"
  check_interval=100m
  retry_interval=5m
  max_check_attempts=3
  vars.snmp_warning = "@0.50:0.60"
  vars.snmp_critical = "~:60"
}
```

- **Hosts_tfg.conf:**

```
object Host "icingaagent"{
        display_name = "icingaagent"
        import "agente_linux"
        address = "192.168.1.126"
        vars.snmp_host = "192.168.1.126"
        vars.grupo = "agente_linux"
}
```

Ya establecida toda la configuración especificada en este proceso y recargado el servicio de Icinga, el *host* mostraría los siguientes atributos monitorizables para el agente SNMP de Linux (ver Figura 41).

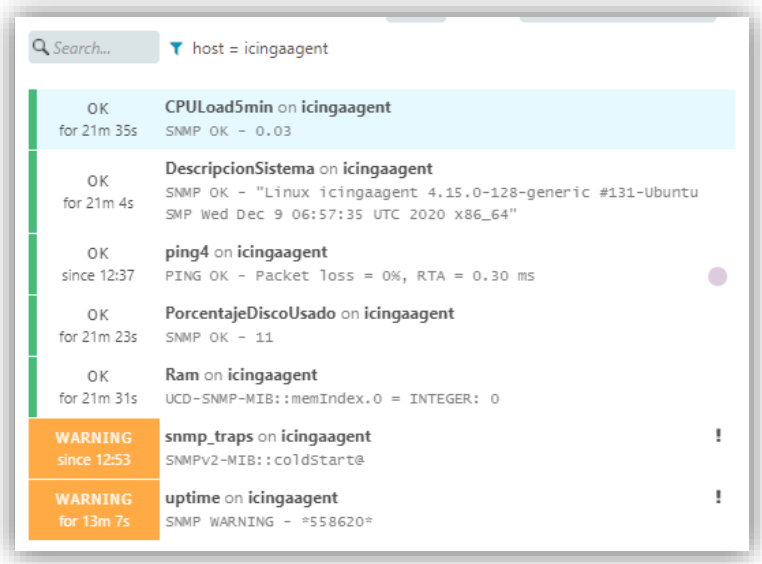

Figura 41. Vista de los servicios de un agente SNMP Linux en Icinga Web.

Configuración de un agente en Windows

En estos sistemas cabe diferenciar que nos encontraríamos ante una disyuntiva pues, dependiendo de la distribución de Windows empleada (Server o PC), su instalación sigue un proceso u otro, si bien la instalación del agente SNMP es definida por Microsoft en ambos casos como una característica.

En equipos personales, la característica ha sido prácticamente desechada, dado que en la actualidad (utilizando la última versión de Windows 10) ya no puede ser instalada a través de la interfaz de instalación de características proporcionada por el sistema. Sin embargo, todavía puede completarse el proceso de manera oficial a través de Powershell (con elevación de permisos) a través de la orden:

```
> Add-WindowsCapability -Online -Name "SNMP.Client~~~~0.0.1.0"
```

Una vez completado el proceso, el agente quedaría a la espera de configuración. No obstante, y por complementar este anexo, se expone a continuación su instalación a través de Windows Server. En esta ocasión el proceso parte de una instalación limpia de Windows Server 2019 y, para completar el proceso, únicamente habría que dirigirse al Panel de Administración del Servidor e indicar que se va a realizar una instalación en el servidor local. Una vez se inicia el

proceso, habría que obviar la instalación de roles y buscar entre las características aquella que integra el agente SNMP, tal y como muestra la siguiente secuencia de imágenes:

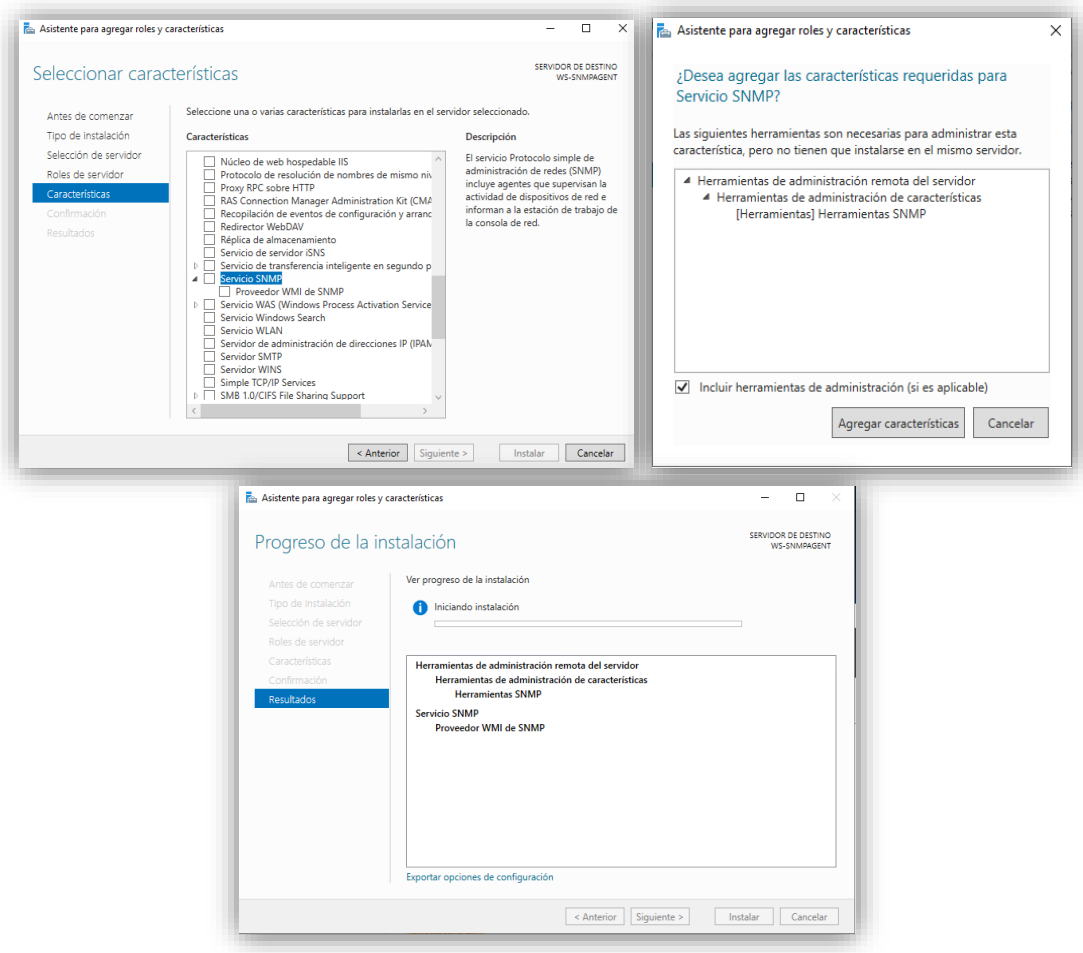

Figura 42. Instalación de la característica de SNMP en un equipo Windows.

Una vez la característica se establece en el equipo, la configuración del agente es igual en las dos distribuciones. Por lo tanto, en cualquier caso, lo siguiente sería dirigirse a los servicios de Windows y buscar entre ellos aquel cuyo nombre es: "Servicio SNMP". Pulsando con el botón derecho sobre él, y entrando en las propiedades, se accede al módulo de configuración. Cabe destacar que, por motivos de seguridad, inicialmente el servicio viene configurado de la misma manera que el procedente del paquete NET-SNMP en Linux. Este sólo acepta peticiones de *localhost*, por lo que habría que cambiar esa opción. Para modificar ese parámetro habría que dirigirse a la pestaña "Seguridad". Al tratarse de un entorno de pruebas, se ha dejado configurado como "Aceptar paquetes SNMP de cualquier Host", y se ha dado de alta la comunidad "wserver" para consulta SNMP. Adicionalmente, se ha dado de alta una comunidad para las notificaciones (*traps*). Para concluir con la configuración en la parte del agente, únicamente habría que ir a la pestaña "Capturas" y establecer con qué comunidad y a qué host(s) dirigir dichas notificaciones. Toda esta configuración puede apreciarse en la siguiente secuencia de figuras:

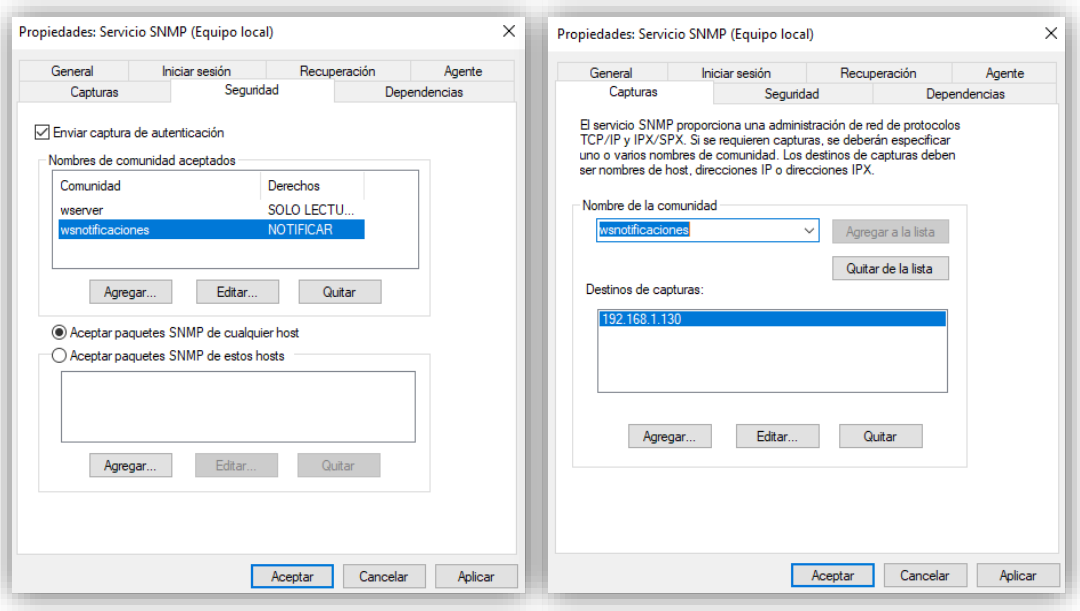

Figura 43. Configuración del agente SNMP de Windows.

Tras ello, al igual que en ocasiones anteriores, debería realizarse una iteración sobre los OID que implementan las MIB compatibles con Windows, además de explorar dichas MIB con alguna herramienta para familiarizarse con su estructura, así como con aquellos OIDs que puedan resultar relevantes. Para ello, desde un terminal (recomendable que sea desde el servidor de Icinga), podría introducirse la orden:

```
# snmpwalk -v2c -c wserver 192.168.1.145
```

Este comando devuelve todos los OID y sus estados. Una vez estudiado el compendio de registros, y habiendo detectado aquellos que son de interés, se puede completar el proceso en la configuración de Icinga. Al igual que en el apartado anterior se ofrece una propuesta de servicios y umbrales para la integración de este servidor en la infraestructura:

- **Commands.conf**:

  No se ha modificado su estructura para esta nueva integración ya que ambos comandos (expuestos anteriormente) son compatibles con este tipo de *hosts*.

- **Groups.conf**:

```
object HostGroup "agente_windows"{
  display_name = "Agente Windows"
```

```
  assign where host.vars.grupo == "agente_windows"
}
```

- **Templates.conf**:

```
template Host "agente_windows" {
  max_check_attempts = 3
  check_interval = 5m
  retry_interval = 5m
  /*check_command = "snmp"*/
  check_command = "hostalive"
  vars.snmp_com = "wserver"
  vars.snmp_vers = "2c"
  vars.snmp_puerto = "161"
  vars.servicios_definidos=[""]
}
```

- **Services.conf**:

```
apply Service "DescripcionSistema"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "agente_windows"
  vars.snmp_oid = ".1.3.6.1.2.1.1.1.0"
  check_interval=1000m
  retry_interval=1m
  max_check_attempts=3
  vars.snmp_warning = ""
  vars.snmp_critical = ""
}

apply Service "UptimeWindows"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "agente_windows"
  vars.snmp_oid = ".1.3.6.1.2.1.25.1.1.0"
  check_interval=10m
  retry_interval=1m
  max_check_attempts=3
```

```
  vars.snmp_warning = "8640000:"
  vars.snmp_critical = ""
}

apply Service "NumeroProcesos"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "agente_windows"
  vars.snmp_oid = ".1.3.6.1.2.1.25.1.6.0"
  check_interval=5m
  retry_interval=1m
  max_check_attempts=3
  vars.snmp_warning = "@90:110"
  vars.snmp_critical = "~:110"
}

apply Service "Core1Load"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "agente_windows"
  vars.snmp_oid = ".1.3.6.1.2.1.25.3.3.1.2.7"
  check_interval=10m
  retry_interval=1m
  max_check_attempts=3
  vars.snmp_warning = "@70:85"
  vars.snmp_critical = "~:85"
}

apply Service "Core2Load"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "agente_windows"
  vars.snmp_oid = ".1.3.6.1.2.1.25.3.3.1.2.8"
  check_interval=10m
  retry_interval=1m
  max_check_attempts=3
  vars.snmp_warning = "@70:85"
  vars.snmp_critical = "~:85"
}
```

```
apply Service "FallosParticionC"{
  import "generic-service"
  check_command = "check_snmp"
  assign where host.vars.grupo == "agente_windows"
  vars.snmp_oid = ".1.3.6.1.2.1.25.2.3.1.7.2"
  check_interval=10m
  retry_interval=1m
  max_check_attempts=3
  vars.snmp_warning = ""
  vars.snmp_critical = "0:0"
}

apply Service "puertos_escuchando"{
  import "generic-service"
  check_command = "snmp_walk"
  assign where host.vars.grupo == "agente_windows"
  vars.snmp_oid = ".1.3.6.1.2.1.6.19.1.7.1"
  check_interval=10m
  retry_interval=1m
  max_check_attempts=3
}
```

- **Hosts_tfg.conf**:

```
object Host "WS-SNMPAGENT" {
        display_name = "WS-SNMPAGENT"
        import "agente_windows"
        address = "192.168.1.145"
        vars.snmp_host = "192.168.1.145"
        vars.grupo = "agente_windows"
}
```

Ya establecida toda la configuración especificada en este proceso y recargado el servicio de Icinga, el *host* mostraría los siguientes atributos monitorizables para el agente SNMP de Windows (ver Figura 44).

Figura 44. Vista de los servicios de un agente SNMP Windows en Icinga Web.

Apéndice B

Manejo de la interfaz de Icinga Web

Gran parte del proceso descrito a lo largo de esta memoria hace referencia al apartado web de Icinga, ya que es mediante este módulo con el que interactuaría el operador/administrador de la infraestructura. Por ello, se ha considerado relevante realizar un breve manual con las secciones que contiene y sus apartados más significativos.

Una vez se accede a Icinga, la pantalla de inicio que se muestra tras el inicio de sesión es el "*Dashboard*", como se ha especificado anteriormente. Estas pantallas contienen pizarras de resumen personalizadas por el usuario, si bien Icinga trae consigo alguna de ellas a modo de ejemplo que permiten representar la información general más relevante.

Una vez dentro, se pueden dividir las funcionalidades que ofrece la herramienta según las secciones que encontramos en las dos partes más diferenciadas de la pantalla: el panel de navegación lateral y la pantalla de central (ver Figura 45).

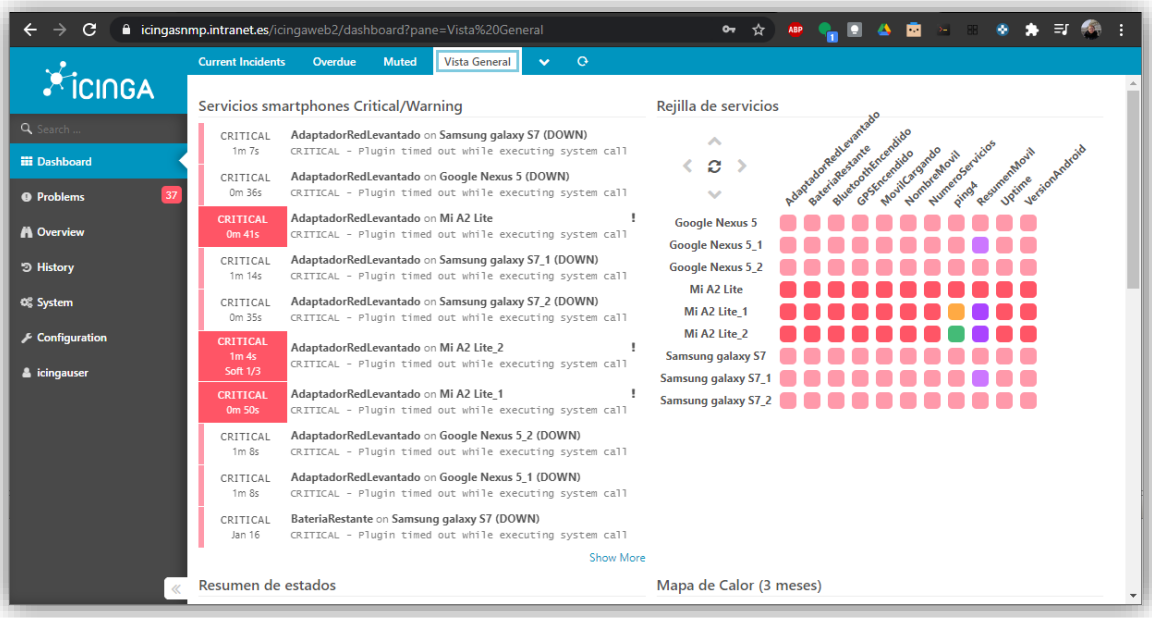

Figura 45. Vista general de Icinga Web.

Como puede apreciarse en la Figura 45, el panel de navegación es la sección que se encuentra en el lado izquierdo, mientas que la pantalla central es todo aquello que contiene un fondo de color blanco.

En el panel de navegación (situado a la izquierda) se encuentran las siguientes opciones:

- **Dashboard**: es la sección en la que se encuentra el usuario tras iniciar sesión. En ella se encuentran todos los *dashlets* (o pizarras de resumen explicadas con mayor profundidad en el apartado 4.5.3). Desde esta zona de la interfaz se puede acceder a los *dashboards* creados por el propio usuario, éstos a su vez, contienen todos los filtros y paneles con la información que suele ser considerada de relevancia por el personal. Para cambiar de un *dashboard* a otro se utilizan las pestañas que se encuentran en la parte superior del panel central.

- **Problems**: Una vez seleccionada esta categoría del panel de navegación, ésta se desplegará y el usuario verá bajo ella, nuevas secciones relacionadas con problemas relacionados con los elementos que se están monitorizando. Dos de estos subapartados que el usuario puede seleccionar son el de *Hosts Problems* y el de *Sevice Problems.* Pulsando en cada uno de ellos, Icinga nos mostrará en el panel central un listado de cada uno de los elementos que requieren atención, *hosts* y *services*, respectivamente. Como última sección de este apartado, se proporciona el "Service grid", rejilla en la que se representan todos los servicios de todos los equipos remotos con su correspondiente operación mediante semáforos (para que esta pantalla tenga sentido, es crucial tener bien configurados los umbrales de cada uno de los elementos). Como se ha mencionado con anterioridad, en la parte superior de cada sección podemos encontrar un icono que permite incluir cada categoría visitada en un *Dashboard* de nuestra elección, así como filtrar el contenido de la vista actual (Ver Figura 46).

Figura 46. Service Grid de Icinga Web.

- **Overview**: Pulsando sobre esta categoría, el panel lateral vuelve a desplegarse, mostrando nuevas secciones que hacen referencia a los elementos dados de alta en el sistema (vistos en el Capítulo 4). Pulsando en cada una de ellas, el panel central mostrará en detalle el contenido que se detalla a continuación:

    - *Tactical Overview:* Una vista táctica en forma de diagramas circulares en los que figuran, a modo de resumen, los estados de los *hosts* y de los *services.*
    - *Hosts:* un listado de todos los dispositivos físicos existentes en el sistema, y sus respectivos estados.
    - *Services*: un listado de todas las señales existentes en el sistema, así como sus respectivos estados.
    - *Hostgroups*: un listado de aquellos grupos de hosts configurados.
    - *Servicegroups.* un listado de aquellos grupos de *services* establecidos en la configuración.
    - *Contacts*: un listado con aquellos contactos configurados para el envío de notificaciones.
    - *Comments:* una sección resumen donde aparecerían pequeñas descripciones dispuestas por el operador de Icinga (u otros usuarios), en servicios o equipos remotos, a modo de notas.
    - *Downtimes:* aquí se encuentran aquellas paradas de servicio planificadas para elementos monitorizados.

- **History**: Pulsando en este apartado del panel de navegación, encontraremos nuevas secciones relacionados con sucesos acontecidos en las últimas horas. Por ejemplo, una rejilla de eventos (a modo de mapa de calor), un resumen cronológico de todo lo sucedido en toda la plataforma, las notificaciones enviadas y un apartado con diagramas circulares que, en función de su tamaño, indicarían la cantidad de eventos recogidos en cada instante.

- **System**: Este aparado permite encontrar en el panel de navegación opciones relacionadas con la propia plataforma de Icinga tales como un equivalente al “Acerca de”, así como una pantalla de control que muestra si, internamente, todo el sistema está funcionando correctamente. Por último se puede localizar una pantalla para configurar mensajes que se mostrarán en la plataforma, independientemente del usuario que inicie sesión, los cuales informarían, por ejemplo, de tareas programadas relacionadas con el mantenimiento del propio servidor.

- **Configuration**: pulsando en esta categoría del panel de navegación se despliegan nuevas opciones, desde ellas, el usuario es dirigido a pantallas en las que se pueden modificar parámetros relacionados con los accesos de la plataforma, ya sea de la propia herramienta a la base de datos, como en lo concerniente a la creación de usuarios y roles. También se encuentra aquí la sección que permite la instalación de módulos de Icinga, creados por terceros, y que puedan añadirse a la interfaz web (siempre y cuando se cuente con aquellos prerrequisitos establecidos por los desarrolladores del módulo).

- **Usuario:** Esta sección, que tendría el nombre del usuario que ha iniciado sesión, dirige a dicho usuario a una pantalla en la que pueden realizarse operaciones sobre el perfil con el que se ha iniciado sesión (cambio de contraseña, idioma, apariencia, etc.).

El panel central, en el que se representa toda la información, contiene múltiples opciones que varían en función del apartado por el que se esté navegando. Cada una de estas pantallas contiene en su interior múltiples enlaces que nos permiten dirigirnos a elementos a los que referencian. Éstos pueden ser distinta naturaleza, aunque generalmente hacen referencia a los *hosts* y a los *services.* Como en definitiva lo más relevante son esas dos categorías, se procede a resumir la navegación por cada una de ellas:

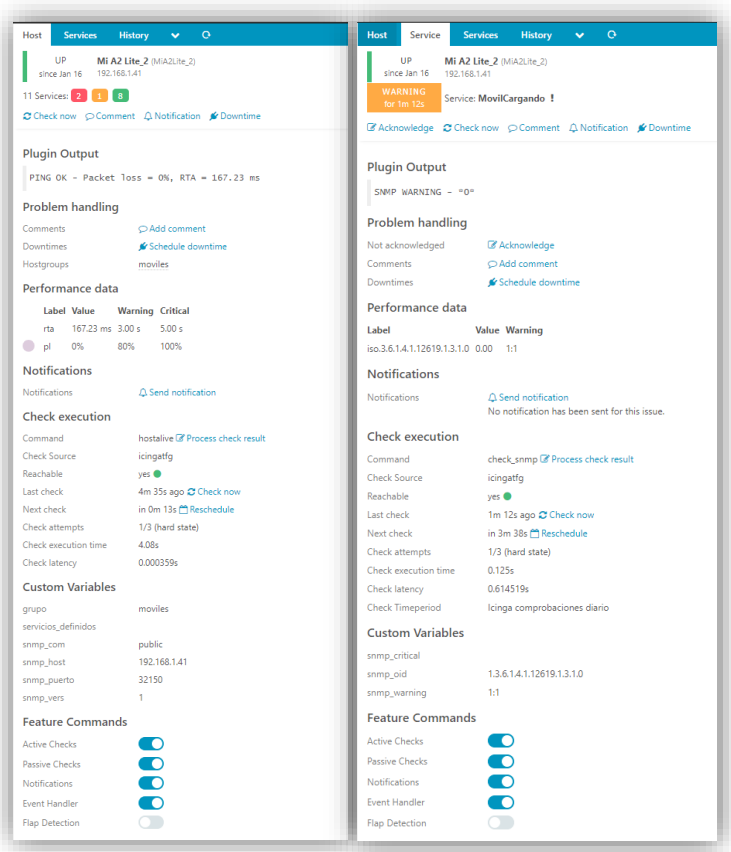

Figura 47. Vista de un host y un service respectivamente en Icinga Web.

Tal y como puede apreciarse en la Figura 47, ambas pantallas guardan entre sí una similitud bastante acentuada, dado que comparten bastantes de sus funciones. En un análisis de arriba abajo, podemos ver que en la parte superior son las propias pestañas las que indican en qué tipo de elemento estamos navegando (*host* o *service)* y, justo debajo, el estado de lo que estamos viendo (nótese que en el caso del *host*, únicamente se indica el nombre del dispositivo, mientras que en el caso del *service*, la pantalla muestra el dispositivo al que hace referencia el estado de este servicio).

Algo más abajo se encuentran una serie de opciones rápidas que son prácticamente comunes a ambas pantallas:

- "**Acknowledge**": es la opción no común a ambas pantallas. Con ella se puede reconocer y aceptar el estado de un problema relacionado con un *host.* De esta manera, se podría detener el envío de notificaciones temporalmente.

- "**Check now**": en *hosts* y *services* de tipo activo permiten interrogar manualmente mediante el uso de la característica *command* de Icinga, sin necesidad de esperar a la siguiente iteración. En aquellos elementos de tipo pasivo, si se pulsa este botón, se fuerza al sistema a marcarlo como "No passive check result", y éste cambiará de estado una vez reciba la siguiente trama.

- "**Comment**": permite poner una nota descriptiva acerca del elemento que se está viendo. Estos son los elementos descritos en el subapartado de "*Overview*" del panel lateral.

- "**Notification**": permite enviar manualmente una notificación al destinatario configurado, algo que puede ser útil para que el responsable de un elemento revise la configuración del dispositivo.

- "**Downtime**": Con esta opción pueden planificarse paradas de servicio que mitiguen el envío de notificaciones y muestran alertas de un nivel inferior, ya que son reconocidas por el sistema como un periodo en el que se realiza el mantenimiento.

Más abajo (ver Figura 47), se pueden observar pequeñas secciones que contienen más información del propio elemento a visualizar. Algunas de las opciones que contienen ya han sido reflejadas en las opciones rápidas recién mencionadas:

- "Plugin Output", salida que Icinga obtiene cuando ejecuta el *plugin* al que hace referencia el *command* establecido en su configuración.

- “Problem Handling” permite reaccionar a los problemas encontrados en ese elemento con el uso de algunas de las opciones rápidas ya descritas.

- “Performance Data” contiene información relacionada con la obtención del dato, así como los umbrales establecidos, el OID, etc.

- “Check Execution” contiene información sobre el *command* utilizado para la obtención del dato, así como referencias acerca de los tiempos y periodos (si los hubiese).

- “Custom variables” hace referencia a los parámetros necesarios para la recolección de la información. En el caso de SNMP se estaría hablando de la comunidad, la versión, etc.

- “Feature Commands” contiene una serie de interruptores que permiten activar o desactivar diferentes funciones relacionadas con el proceso de monitorización (notificaciones, manejador de eventos, etc.).

Apéndice C

Pliego de Condiciones.

A lo largo de todo el proceso descrito en la memoria (dejando al margen el estudio teórico de la supervisión, así como del protocolo SNMP), se mencionan diversos componentes que conforman la infraestructura de monitorización. Con el fin de garantizar el buen funcionamiento de todo el conjunto, se ofrecen a continuación una serie de condiciones técnicas que deberían cumplirse en caso de un despliegue futuro.

- Pliego de condiciones generales:

    - La plataforma de monitorización funciona con una arquitectura Cliente-Servidor. La parte que comunica con los dispositivos remotos, y que se encontraría en un servidor con Hyper-V, a través de una web sirve los datos que utilizará el usuario para explotar la información con un navegador.

    - El usuario podrá conectarse con sus credenciales a Icinga desde cualquier dispositivo con un navegador, siempre que el enrutamiento lo permita.

    - El usuario es el responsable final de mantener la confidencialidad de los datos obtenidos a través de la plataforma de monitorización independientemente de los permisos que apliquen al perfil. Adicionalmente, al contarse con una identificación a través de usuario y contraseña, éste debe mantener dicha combinación a buen recaudo y variar con cierta frecuencia el campo de contraseña.

    - Mantener el entorno actualizado (tanto la parte servidor como el equipo cliente) con las últimas versiones del software a fin de obtener las últimas mejoras en funcionalidad y materia de seguridad, algo que debería ser prioritario a fin de garantizar la privacidad de los datos que se manejan.

    - En caso de que cualquier fallo impida una correcta consulta de información a través de Icinga, se recomienda contactar con el personal destinado al soporte de

la infraestructura con celeridad, con el propósito de reducir al mínimo la parada de servicio y recuperar así la visibilidad del entorno lo antes posible.

  - Se recomienda encarecidamente la disposición de un mecanismo de copias de seguridad que permita la recuperación rápida de información relevante, así como de la configuración de Icinga, protegiendo así también al entorno de posibles ataques *ransomware*. Asimismo, se recomienda integrar la máquina virtual en un ámbito que permita utilizar herramientas de alta disponibilidad y/o mecanismos de replicación.

- Pliego de condiciones específicas:

  - Especificaciones de funcionamiento:

    - Conexión de área local, o bien enrutamiento del tráfico para hacer posible la comunicación del servidor con los dispositivos, así como del usuario con el servidor de monitorización.

    - Contar con un servidor DNS que asegure el buen funcionamiento del SSL configurado en el apartado de las mejoras.

  - Especificaciones técnicas de la parte servidor: Si bien el entorno puede montarse directamente en un servidor, se recomienda mantener la arquitectura de virtualización para dotar a la herramienta de una mayor flexibilidad y redundancia:

    - Contar con el rol de Hyper-V.

    - Posibilidad de desplegar máquinas virtuales en Versión 1.

    - El servidor debe admitir una versión de configuración 9.0 o superior.

    - Contar con un *VirtualSwitch* de tipo “Externo”

    - Asignación de 4 GB de RAM (con posibilidad de que sean dinámicos).

    - Asignación de al menos 2vCPU al recurso virtual.

        - Contar con almacenamiento disponible que permita al entorno albergar históricos a medida que se incorporan más recursos monitorizables (actualmente la máquina virtual puede crecer hasta 127GB de manera dinámica).

    - Especificaciones técnicas de la parte cliente:

        - Independencia del sistema operativo.

        - Navegador Web actualizado (recomendado Google Chrome v87.0.4280.88) o superior.

        - 1GB de RAM disponible.

        - Procesador dual core o superior.

        - Almacenamiento de 1GB disponible.

        - Conector .NET/MySQL instalado en el sistema.

        - Powershell 5.0 o superior.

        - Se recomienda disponer de las últimas actualizaciones instaladas en el sistema.

- Pliego de cláusulas administrativas:

    - La correcta adaptación de la web al navegador depende del dispositivo utilizado para su conexión.

    - La rapidez con la que se presentan los datos depende de los recursos asignados a la máquina virtual, los cuellos de botella ocasionados por la compartición de recursos así como del ancho de banda disponible.

    - La velocidad con la que pueden recuperarse datos de una consulta de históricos depende de la velocidad de lectura del disco en el que se encuentre, de la cantidad de datos almacenados en el sistema así como del número de dispositivos configurados en el servidor.

Apéndice D.

Presupuesto.

El propósito de este apéndice es realizar una estimación acerca del coste que tendría un despliegue similar al descrito a lo largo de esta memoria en un escenario real, a través de labores desempeñadas por trabajadores cualificados. Para ello, lo primero sería definir los roles del equipo de trabajo:

- Jefe de proyecto: Persona al cargo del proyecto y de su orientación.
    - Al cargo de las contrataciones del resto del equipo y la asignación de sus roles.
    - Encargado de encontrar clientes y mantener contacto fluido con ellos.
    - Supervisar el trabajo dispuesto y pendiente en reuniones periódicas con los trabajadores.
    - Responsable de la planificación y los plazos acordados con el cliente.
    - Salario estimado de 30 €/h.

- Analista: empleado al cargo del discernimiento de las necesidades.
    - Capturar los requisitos del cliente y las limitaciones.
    - El análisis funcional de la plataforma.
    - El análisis técnico del entorno donde se desplegará la plataforma.
    - Encontrar alternativas durante la fase de desarrollo frente a posibles limitaciones del entorno.
    - Salario estimado de 25 €/h.

- Desarrollador: personal destinado a la creación de los recursos necesarios para la adaptación de las necesidades al entorno de monitorización:
    - Creación y configuración de los ficheros que contienen los parámetros de monitorización.
    - Creación de *plugins* y *scripts* que cubran las necesidades específicas del cliente.
    - Implantación de mejoras específicas a las necesidades del cliente.
    - Creación de nuevos módulos y apartados en la parte web de la plataforma.
    - Salario estimado de 18 €/h.

- Administrador del entorno: Responsable del mantenimiento del servicio y sistemas.
    - Instalación y configuración de los componentes del entorno.
    - Corrección de fallos y restablecimiento de la monitorización.
    - Mantener actualizada la documentación con la configuración más reciente.
    - Actualizaciones del entorno y subidas a producción en colaboración con el desarrollador. Realización de las copias de seguridad.
    - Salario estimado de 18 €/h.

- Personal de pruebas: Personal con experiencia en el manejo de este tipo de plataformas que prueba la calidad del producto final.
    - Detectar fallos y carencias en base a la funcionalidad establecida.
    - Validar las nuevas actualizaciones que se vayan incorporando al producto.
    - Probar la cobertura de necesidades establecidas en los requisitos.
    - Salario estimado de 18 €/h.

En el supuesto de que se decidiera cambiar parte de la infraestructura, deberían tenerse en cuenta para este presupuesto los costes asociados al licenciamiento de los productos que formen parte del entorno (el uso de diferentes sistemas operativos, software de pago para copias de seguridad, antivirus, etc.). Ya que para la implantación propuesta en este proyecto únicamente se ha utilizado software libre, el coste asociado a esta parte sería inexistente. De igual manera, se han obviado los costes asociados a la compra de hardware, así como de aquellos asociados al funcionamiento activo del proyecto, como sería el caso de gastos de alimentación eléctrica o contratación de un ISP.

Para el desarrollo completo del proyecto se ha decidido dividir la totalidad del proceso en las siguientes fases:

- **Fase 1:** Negociación con el cliente y planteamiento de la solución.
- **Fase 2:** Captura de requisitos.
- **Fase 3:** Planificación y establecimiento de objetivos
- **Fase 4:** Diseño y planteamiento de procedimientos
- **Fase 5:** Configuración nativa del entorno.
- **Fase 6:** Incorporación y desarrollo de mejoras específicas (fase sujeta a las preferencias y necesidades del cliente, para esta estimación se establecen las incorporadas al proyecto actual).
- **Fase 7:** Pruebas y corrección de errores.

Por lo tanto, teniendo en cuenta todo lo anterior, y considerando que cada uno de los trabajadores realiza su trabajo en jornadas de 8 horas (40 horas semanales; para la puesta en marcha de este proyecto no se contempla el uso de horas extra), se estima que la duración completa del despliegue sería de 427 horas, distribuidas según se indica en la Tabla 13.

	Jefe Proyecto	Analista	Desarrollador	Sistemas	P. Pruebas	Coste
Fase 1	15 hrs.	5 hrs.	-	-	-	575 €
Fase 2	8 hrs.	20 hrs.	-	-	-	740 €
Fase 3	15 hrs.	10 hrs.	5 hrs.	-	-	790 €
Fase 4	4 hrs.	10 hrs.	5 hrs.	10 hrs.	-	640 €
Fase 5	4 hrs.	5 hrs.	30 hrs.	40 hrs.	20 hrs.	1865 €
Fase 6	6 hrs.	10 hrs.	50 hrs.	40 hrs.	25 hrs.	2500 €
Fase 7	5 hrs.	5 hrs.	20 hrs.	20 hrs.	40 hrs.	1715 €
					Coste total:	**8825 €**

Tabla 13. Tabla de costes con un desglose por fases del proyecto.

Adicionalmente a la puesta en marcha de todo el entorno, podría sugerirse al cliente la contratación de un servicio de soporte y/o mantenimiento de la plataforma, así como un servicio de guardias en caso de que se considere oportuno, ya que es habitual que estos servicios requieran de una respuesta rápida ante los imprevistos que pudieran surgir. Los costes asociados a este concepto se facturarían adicionalmente de forma periódica en función de las necesidades contratadas, o bien con un formato de bolsa de horas.

Parte IV

Bibliografía

Bibliografía

Adams, M. (24 de Mayo de 2019). *Windows Report*. Recuperado el Diciembre de 2020, de https://windowsreport.com/restore-missing-snmp-windows-10/

Apache Documentation. (2020). *Apache HTTP Server Project*. Recuperado el Noviembre de 2020, de https://httpd.apache.org/

Castaños, I. (2017). *Semantic Systems*. Recuperado el Octubre de 2020, de https://www.semantic-systems.com/semantic-noticias/articulos-tecnologicos/en-que-consiste-la-monitorizacion-de-sistemas/

Colomer, J., Joaquim, M., & Jordi, A. (2000). *Comité Español de Automática*. Obtenido de https://intranet.ceautomatica.es/sites/default/files/upload/10/files/sistemas%20de%20supervision.pdf

Davin, J., Proteon, I., & Case, J. (Noviembre de 1987). *Internet Archive*. (University of Tennessee at Knoxville) Recuperado el Noviembre de 2020, de https://web.archive.org/web/20050925012544/http://www.cse.ohio-state.edu/cgi-bin/rfc/rfc1028.html

Dell Inc. (2019). *Centro Aprendizaje Dell*. Obtenido de iDrac con Lifecycle Controller: https://www.dell.com/learn/mx/es/mxslg1/solutions/integrated-dell-remote-access-controller-idrac

Digital Ocean. (5 de Diciembre de 2017). *Digital Ocean*. Recuperado el Diciembre de 2020, de https://www.digitalocean.com/community/tutorials/an-introduction-to-metrics-monitoring-and-alerting

Enlyft. (2021). *enlyft*. Recuperado el Noviembre de 2020, de https://enlyft.com/tech/products/nagios#:~:text=We%20have%20data%20on%2030%2C729,in%20the%20Computer%20Software%20industry.

Enlyft. (2021). *enlyft - Icinga*. Recuperado el Noviembre de 2020, de https://enlyft.com/tech/products/icinga

Erk, B. (6 de Mayo de 2009). *Icinga*. (Icinga Development Team) Recuperado el Noviembre de 2020, de https://icinga.com/blog/2009/05/06/announcing-icinga/

F.Kurose, J., & W. Ross, K. (2010). Redes de computadoras (Un enfoque descendente). Pearson Education.

Ferri, A. (14 de Febrero de 2019). *A3sec*. Recuperado el Octubre de 2020, de https://blog.a3sec.com/monitorizaci%C3%B3n-activa-vs.-monitorizaci%C3%B3n-pasiva

Gold, K. (8 de Febrero de 2019). *EXFO*. Obtenido de https://www.exfo.com/es/recursos/blog/active-passive-network-monitoring/

Hein, D. (26 de Febrero de 2019). *Solutions Review*. Recuperado el Octubre de 2020, de https://solutionsreview.com/network-monitoring/active-monitoring-and-passive-monitoring-whats-the-difference/

Hewlett Packard Enterprise. (2020). *HPE Integrated Lights Out*. Recuperado el Noviembre de 2020, de https://www.hpe.com/es/es/servers/integrated-lights-out-ilo.html

IBM. (2020). *IBM Knowledge Center*. Recuperado el Noviembre de 2020, de https://www.ibm.com/support/knowledgecenter/SSB23S_1.1.0.2020/gtpc1/pdus.html

Icinga Development Team. (Mayo de 2009). *Internet Archive*. Recuperado el Noviembre de 2020, de https://web.archive.org/web/20100514004246/http://www.icinga.org/faq/why-a-fork/

Icinga Development Team. (2020). *Icinga - Customers*. Recuperado el Diciembre de 2020, de https://icinga.com/about/customers/

Icinga Development Team. (2020). *Icinga Documentation (About)*. Recuperado el Enero de 2021, de https://icinga.com/docs/icinga-2/latest/doc/01-about/
Icinga Development Team. (2021). *Icinga Documentation - Monitoring Basics*. Recuperado el Febrero de 2021, de https://icinga.com/docs/icinga-2/latest/doc/03-monitoring-basics/
IEEE. (13 de Noviembre de 2017). *IEEE Xplore*. Recuperado el Octubre de 2020, de https://ieeexplore.ieee.org/document/8270432
IETF. (Mayo de 1990). *Tools IETF*. Obtenido de https://tools.ietf.org/html/rfc1157
Microsoft. (2 de 12 de 2016). *Documentación Microsoft - Hyper-V en Windows Server*. Recuperado el 2020 de Noviembre, de https://docs.microsoft.com/es-es/windows-server/virtualization/hyper-v/get-started/install-the-hyper-v-role-on-windows-server
Microsoft. (25 de 06 de 2018). *Documentación Microsoft*. Recuperado el Noviembre de 2020, de https://docs.microsoft.com/es-es/virtualization/hyper-v-on-windows/about/
MySQL. (2021). *MySQL Documentation*. Obtenido de https://www.mysql.com/
Nagios. (1999). *Nagios*. Obtenido de https://www.nagios.com/
Nagios. (7 de 12 de 2016). *Nagios Exchange Center - NSCA*. Obtenido de https://exchange.nagios.org/directory/Addons/Passive-Checks/NSCA--2D-Nagios-Service-Check-Acceptor/details
Nagios Enterprises, LLC. (2018). *Nagios Docs - Threshold and Ranges*. Recuperado el Diciembre de 2020, de https://nagios-plugins.org/doc/guidelines.html#THRESHOLDFORMAT
Nagios Enterprises, LLC. (2019). *Nagios Docs*. Recuperado el Diciembre de 2020, de https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/checkscheduling.html
Nagios Enterprises, LLC. (13 de Enero de 2021). *Nagios Support Knowledgebase*. Recuperado el Enero de 2021, de https://support.nagios.com/kb/article.php?id=88
National Institute of Standard and Technology. (Mayo de 2015). *NIST*. Recuperado el Noviembre de 2020, de https://csrc.nist.gov/glossary/term/Master_Terminal_Unit
NET-SNMP. (15 de Enero de 2004). *SourceForge*. Recuperado el Diciembre de 2020, de http://net-snmp.sourceforge.net/docs/man/snmptrapd.html#lbAE
NET-SNMP. (29 de Junio de 2005). *SourceForge*. Recuperado el Diciembre de 2020, de http://net-snmp.sourceforge.net/docs/man/snmpcmd.html
NET-SNMP. (24 de Agosto de 2019). *NET-SNMP*. Recuperado el Enero de 2021, de http://net-snmp.sourceforge.net/docs/mibs/host.html
NET-SNMP. (24 de Agosto de 2019). *NET-SNMP*. Recuperado el Enero de 2021, de http://www.net-snmp.org/tutorial/tutorial-5/commands/snmptrap.html
NET-SNMP. (22 de Mayo de 2020). *NET-SNMP Official Website*. Recuperado el Enero de 2021, de http://www.net-snmp.org/
Notes, E. (2020). *Electronics Notes*. Recuperado el Noviembre de 2020, de https://www.electronics-notes.com/articles/connectivity/serial-data-communications/rs232-eia-v24-standard.php
Open SSL. (2020). *OpenSSL (Cryptography and SSL/TLS Toolkit)*. Recuperado el Enero de 2021, de https://www.openssl.org/docs/
R. Mauro, D., & J. Schmidt, K. (2005). Essential SNMP, Second Edition. R' REILLY.
SNMPTT Documentation. (2020). *Source Forge - SNMPTT*. Obtenido de http://snmptt.sourceforge.net/docs/snmptt.shtml#SNMPTT.CONF-Configuration-file-format
SSL market. (2019). *SSL Market*. Recuperado el Enero de 2021, de https://www.sslmarket.es/ssl/help-formatos-de-los-certificados-ssl

Supervisor. (2021). *Supervisor*. Recuperado el Enero de 2021, de http://supervisord.org/configuration.html
Wikipedia. (27 de Mayo de 2020). *Wikipedia - Monitoreo de red*. Recuperado el Octubre de 2020, de https://es.wikipedia.org/wiki/Monitoreo_de_red
Wikipedia. (16 de Mayo de 2020). *Wikipedia - Monitorización Servidores de Internet*. Recuperado el Octubre de 2020, de https://es.wikipedia.org/wiki/Monitorizaci%C3%B3n_de_Servidores_de_Internet
Wikipedia. (4 de Marzo de 2021). *Wikipedia - Modelo TCP/IP*. Recuperado el Noviembre de 2020, de https://es.wikipedia.org/wiki/Modelo_TCP/IP

Universidad de Alcalá