

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

“Diseño e  
implementación de  
un frontend para R-  
Snort”

Curso 2024/2025

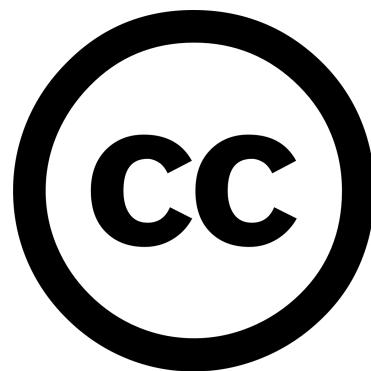
**Alumno/a:**

Deian Orlando Petrovics Tabacu

**Director/es:**

Julio Gómez López  
Nicolás Padilla Soriano





Este trabajo está bajo una licencia Creative Commons  
Atribución-NoComercial-CompartirIgual 4.0 Internacional.



## Agradecimientos

Quiero expresar mi más sincero agradecimiento a mi tutor, Julio Gómez López, y a mi co-tutor, Nicolás Padilla Soriano, por su orientación y apoyo durante todo el desarrollo de este trabajo. Su experiencia ha sido el pilar para mantener un enfoque de este proyecto.

También quiero dar las gracias a mi familia, en especial a mis padres, por haberme transmitido desde pequeño el valor del esfuerzo, la constancia y la curiosidad. Su confianza incondicional me ha permitido perseguir mis metas con libertad y determinación.

A mis amigos y compañeros de carrera, con quienes he compartido frustraciones, risas, noches de trabajo y logros, gracias por formar parte de este camino. Vuestra compañía ha sido tan importante como el contenido aprendido.

Y por último, a mí mismo, por no rendirme. Por seguir adelante incluso cuando el cansancio o la duda amenazaban con paralizarme. Este trabajo representa no solo un proyecto académico, sino también un ejercicio de crecimiento, disciplina y pasión por la tecnología como herramienta para construir un mundo más seguro y accesible.



# Índice general

<b>Introducción</b>	1
Motivación	2
Objetivos	3
Fases de la realización y cronograma	4
Estructura y metodología	6
<b>1 Sistemas de detección de intrusos: Snort y frontends</b>	9
1.1 IDS / NIDS	9
1.2 R-Snort	10
1.3 Frontends más utilizados de Snort	12
1.4 Comparativa de interfaces web	17
1.4.1 Interfaz web de Snort en pfSense	17
1.4.2 Interfaz BASE para Snort	19
1.4.3 Interfaz web de Suricata en T-Pot	20
<b>2 Diseño e implantación de un frontend para R-Snort</b>	23
2.1 Introducción	23
2.2 Especificaciones del sistema	25
2.3 Entorno de trabajo	26
2.3.1 Hardware: Raspberry Pi como agente de seguridad	26
2.3.2 Software: Snort 3 y complementos	30
2.3.3 Entorno de desarrollo	33
2.4 Diseño del frontend	33
2.4.1 Diseño de la arquitectura	33
2.4.2 Diseño de la interfaz de usuario	36
2.4.3 Esquema de la base de datos	38
2.5 Implementación y configuración del sistema	41
2.5.1 Snort 3: configuración de salida de alertas en JSON	41
2.5.2 Implementación del backend	48
2.5.3 Implementación del frontend	54
2.6 Script automático R-Snort	60
2.6.1 Instalador del agente: snort-agent	60
2.6.2 Instalador del módulo central: r-snort-central-module	63
<b>3 Caso práctico: utilización del frontend de R-Snort</b>	67
3.1 Entorno de trabajo	67
3.2 Instalación del sistema	69

3.2.1	Preparación de la instalación . . . . .	71
3.2.2	Instalación del agente R-Snort . . . . .	77
3.2.3	Instalación del módulo central de R-Snort . . . . .	78
3.3	Utilización y pruebas . . . . .	80
3.3.1	Configuración . . . . .	80
3.3.2	Pruebas de rendimiento . . . . .	84
3.3.3	Pruebas funcionales . . . . .	88
3.4	Resumen . . . . .	94
<b>4</b>	<b>Resultados y discusión . . . . .</b>	<b>96</b>
4.1	Resultados obtenidos . . . . .	96
4.1.1	Eficacia en la detección de amenazas . . . . .	96
4.1.2	Rendimiento y escalabilidad . . . . .	96
4.1.3	Usabilidad y experiencia de usuario . . . . .	97
4.1.4	Limitaciones y consideraciones . . . . .	97
4.2	Discusión . . . . .	97
<b>Conclusiones</b>		<b>100</b>
<b>Trabajo futuro</b>		<b>102</b>
<b>Bibliografía</b>		<b>104</b>
<b>I</b>	<b>Descripción de la plataforma R-Snort . . . . .</b>	<b>108</b>
I.1	¿Qué es R-Snort? . . . . .	108
I.2	Componentes principales de la plataforma . . . . .	108
I.3	A continuación se muestra el flujo: . . . . .	108
I.4	Requisitos mínimos . . . . .	109
I.5	Instalación y despliegue básico . . . . .	109
I.6	Funcionalidades principales . . . . .	109
I.7	Consejos de administración y mantenimiento . . . . .	110
I.8	Recursos y soporte . . . . .	110
<b>II</b>	<b>Repositorios y guía rápida . . . . .</b>	<b>112</b>
II.1	Repositorio del Agente R-Snort . . . . .	112
II.2	Repositorio del Módulo Central (WebApp) . . . . .	113
II.3	Consejos rápidos y soporte . . . . .	113
<b>III</b>	<b>Guía para escribir reglas Snort 3 . . . . .</b>	<b>116</b>
III.1	Estructura general de una regla . . . . .	116
III.2	Cabecera . . . . .	116
III.3	Opciones más comunes . . . . .	116
III.4	Ejemplo comentado paso a paso . . . . .	117
III.5	Buenas prácticas . . . . .	117
III.6	Validación y despliegue . . . . .	118
III.7	Recursos recomendados . . . . .	118

# Índice de figuras

1	Cronograma de ejecución del TFG R-SNORT. . . . .	5
2	Comparativa de tiempo estimado vs. tiempo real por categoría. . . . .	5
1.2	Logo proyecto R-Snort. . . . .	12
1.3	Interfaz de BASE. . . . .	13
1.4	Interfaz de Snort Report. . . . .	13
1.5	Interfaz de Aanval. . . . .	14
1.6	Interfaz de Sguil. . . . .	15
1.7	Interfaz de Snorby. . . . .	16
1.8	Panel de alertas de Snort en la interfaz web de pfSense. . . . .	18
1.9	BASE. . . . .	19
1.10	Panel principal de Suricata en T-Pot. . . . .	21
2.1	Arquitectura general del sistema R-Snort. . . . .	24
2.2	Esquema de despliegue de R-Snort en la red local. . . . .	28
2.3	Uso de CPU de la Raspberry Pi 5 bajo carga de trabajo habitual. . . . .	29
2.4	Uso de la CPU de la Raspberry Pi 5 bajo carga de trabajo extremo. . . . .	29
2.5	Carga del sistema de la Raspberry Pi 5 bajo en condiciones extremas. . . . .	29
2.6	Esquema de la arquitectura distribuida de R-Snort. . . . .	35
2.7	Tabla alertas. . . . .	38
2.8	Tabla métricas. . . . .	39
2.9	Tabla usuario. . . . .	39
2.10	Tablas base de datos. . . . .	40
2.11	BPMN del servicio de almacenamiento de alertas. . . . .	43
2.12	BPMN del servicio de almacenamiento de métricas. . . . .	44
2.13	BPMN del servicio de logrotate y archivado. . . . .	45
2.14	Dashboard de Grafana adaptado para R-Snort. . . . .	47
2.15	Diagrama de funcionamiento del agente. . . . .	48
2.16	Documentación Swagger de la API de un agente R-Snort. . . . .	49
2.17	Flujo de funcionamiento del backend. . . . .	53
2.18	Vista general. . . . .	54
2.19	Vista de alertas del agente seleccionado en la interfaz R-Snort. . . . .	55
2.20	Reglas del sistema de R-Snort. . . . .	55
2.21	Estado del agente seleccionado. . . . .	56
2.22	Estado de los servicios de un agente individual. . . . .	56
2.23	Diagrama de secuencia de interacción. . . . .	60
2.24	Proceso detallado de instalación del agente R-Snort. . . . .	63

2.25 Proceso escalonado de instalación del módulo central de R-Snort. . . . .	65
3.1 Funcionamiento del sistema R-Snort en modo mononodo. . . . .	67
3.2 Esquema lógico de red con múltiples agentes Snort y módulo central R-Snort. . . . .	68
3.3 Proceso de escritura de Ubuntu Server 25.04 (64-bit) en una Raspberry Pi 5 mediante Raspberry Pi Imager. . . . .	72
3.4 Conexión física de la Raspberry Pi 5 como módulo central. . . . .	73
3.5 Conexiones del switch. . . . .	73
3.6 Configuración del mirroring. . . . .	74
3.7 Raspberry Pi 3 actuando como agente. . . . .	74
3.8 Conexiones del switch secundario. . . . .	75
3.9 Inicio de la instalación del agente. . . . .	76
3.10 Resumen tras la instalación. . . . .	76
3.11 Inicio de la instalación del agente. . . . .	77
3.12 Finalización y resumen del agente instalado. . . . .	78
3.13 Inicio de la instalación del módulo central. . . . .	78
3.14 Finalización y resumen del módulo central. . . . .	79
3.15 Pantalla de inicio de sesión del sistema R-Snort. . . . .	80
3.16 Panel general con métricas y alertas. . . . .	80
3.17 Panel de alertas clasificadas por severidad. . . . .	81
3.18 Gestión de reglas Snort desde la interfaz. . . . .	81
3.19 Monitorización de temperatura, CPU y estado del servicio. . . . .	82
3.20 Estado detallado del agente central. . . . .	82
3.21 Menú de selección de agentes activos. . . . .	83
3.22 Formulario para añadir un nuevo agente Snort. . . . .	83
3.23 Visualización de métricas del agente secundario. . . . .	83
3.24 Gestión avanzada de agentes Snort desde la interfaz. . . . .	84
3.25 Temperatura de la CPU durante funcionamiento normal. . . . .	85
3.26 Consumo de CPU y uso de disco durante funcionamiento normal. . . . .	85
3.27 Alertas registradas durante tráfico normal. . . . .	87
3.28 Porcentaje de uso de CPU y disco durante el ataque. . . . .	87
3.29 Temperatura de la CPU durante la fase de estrés. . . . .	87
3.30 Dashboard de Grafana mostrando más de 250.000 alertas tras una prueba de estrés. Se observan tendencias por severidad, protocolo e IPs de origen. . . . .	90
3.31 Panel de alertas en la aplicación web con visualización correcta de alertas severas, medias y bajas tras las pruebas de carga. . . . .	90
3.32 Descarga de archivos de logs archivados generados por el sistema de rotación. . . . .	91
3.33 Opciones de descarga de alertas activas por agente o globales en formato CSV. . . . .	91
3.34 Inserción de una nueva regla de detección de conexiones SSH no autorizadas. . . . .	92
3.35 Regla personalizada añadida correctamente al sistema del agente central. . . . .	92
3.36 Confirmación de eliminación de una regla personalizada. . . . .	92
3.37 Verificación antes de eliminar un agente de la interfaz. . . . .	93
3.38 Estado de los servicios de los agentes y opción de reinicio remoto. . . . .	93

## ÍNDICE DE FIGURAS

---

II.1 Ejemplo de documentación y guía de inicio rápido disponible en el repositorio del agente. . . . .	112
II.2 Vista del repositorio y estructura de archivos del módulo central de R-Snort. . .	113
II.3 Ejemplo de funcionalidades destacadas y guía visual en el README de la WebApp.	114

# Índice de tablas

1.1	Comparativa de funcionalidades de las interfaces web de IDS analizadas. . . . .	21
2.1	Resumen de los endpoints expuestos por el backend del agente R-Snort. . . . .	49
3.1	Requisitos funcionales y no funcionales previos al despliegue del sistema R-Snort	71



## ÍNDICE DE TABLAS

# Índice de Códigos

2.1 Ejemplo de alerta en formato JSON generada por Snort 3 . . . . .	30
2.2 Configuración de Snort 3 para salida de alertas en JSON . . . . .	41
2.3 Esquema simplificado del servicio de almacenamiento de alertas . . . . .	43
2.4 Regla de logrotate para los logs de Snort . . . . .	45
2.5 Consulta SQL (simplificada) para panel de protocolos en Grafana . . . . .	46
2.6 Fragmento del código que maneja cada endpoint . . . . .	50
2.7 Ejemplo simplificado de llamada desde el backend central a un agente . . . . .	51
2.8 Ejemplo simplificado de authGuard . . . . .	58
2.9 Login y navegación tras autenticación . . . . .	59
2.10 Carga de alertas desde el servicio . . . . .	59
2.11 Fragmento de creación del archivo db.cnf . . . . .	61
2.12 Definición del servicio systemd para Snort . . . . .	61
2.13 Fragmento de inserción de alerta en ingest_service.py . . . . .	62
2.14 Fragmento del servicio systemd para el backend . . . . .	64
3.1 Instalación de R-Snort personalizada . . . . .	76
3.2 Comandos de instalación del agente . . . . .	77
3.3 Comandos de instalación del agente . . . . .	78
3.4 Script para generar ataques masivos contra R-Snort . . . . .	88
III.1 Plantilla mínima de regla Snort 3 . . . . .	116
III.2 Regla que detecta intentos de login FTP anónimos . . . . .	117



# Abreviaturas

- **API:** Application Programming Interface (Interfaz de Programación de Aplicaciones)
- **CSV:** Comma-Separated Values (Valores Separados por Comas)
- **DB:** DataBase (Base de Datos)
- **DNS:** Domain Name System (Sistema de Nombres de Dominio)
- **DoS:** Denial of Service (Denegación de Servicio)
- **GUI:** Graphical User Interface (Interfaz Gráfica de Usuario)
- **HIDS:** Host Intrusion Detection System (Sistema de Detección de Intrusos en Host)
- **HTTP:** Hypertext Transfer Protocol (Protocolo de Transferencia de Hipertexto)
- **IDS:** Intrusion Detection System (Sistema de Detección de Intrusos)
- **IPS:** Intrusion Prevention System (Sistema de Prevención de Intrusos)
- **JSON:** JavaScript Object Notation (Notación de Objetos de JavaScript)
- **LAN:** Local Area Network (Red de Área Local)
- **NIDS:** Network Intrusion Detection System (Sistema de Detección de Intrusos en Red)
- **PME:** Pequeña y Mediana Empresa (Small and Medium Enterprise, SME)
- **REST:** Representational State Transfer (Transferencia de Estado Representacional)
- **R-Pi:** Raspberry Pi (Plataforma de computación de bajo coste)
- **SIEM:** Security Information and Event Management (Gestión de Información y Eventos de Seguridad)
- **SMTP:** Simple Mail Transfer Protocol (Protocolo Simple de Transferencia de Correo)
- **SNMP:** Simple Network Management Protocol (Protocolo Simple de Administración de Red)
- **SOHO:** Small Office/Home Office (Pequeña Oficina / Oficina en Casa)
- **SQL:** Structured Query Language (Lenguaje de Consulta Estructurado)
- **SSH:** Secure Shell (Protocolo Seguro de Comunicación Remota)
- **SSL/TLS:** Secure Sockets Layer / Transport Layer Security (Capa de Conexión Segura / Seguridad en la Capa de Transporte)
- **UI:** User Interface (Interfaz de Usuario)
- **URL:** Uniform Resource Locator (Localizador Uniforme de Recursos)
- **VM:** Virtual Machine (Máquina Virtual)



# Introducción

La creciente dependencia digital de las empresas ha convertido a la ciberseguridad en un factor determinante para la supervivencia y éxito de cualquier negocio, especialmente en el caso de las pequeñas y medianas empresas (PYMEs). En España, donde las PYMEs constituyen la base del tejido empresarial (en torno al 99 % de las empresas) [1] , los ciberataques se han disparado en número y sofisticación. Se estima que casi la mitad de los ciberataques a nivel mundial van dirigidos a PYMEs, un problema particularmente serio en países como España [2]. De hecho, siete de cada diez ataques en España tienen como objetivo organizaciones de tamaño pequeño o mediano, aprovechando que suelen contar con menos medidas de seguridad y sistemas más vulnerables. Las consecuencias de estas brechas pueden ser devastadoras: según datos de Telefónica, un 60 % de las PYMEs que sufren un ciberataque acaban desapareciendo en menos de seis meses tras el incidente [3], ya sea por pérdidas financieras, daños reputacionales o interrupción prolongada de sus operaciones. Este panorama pone de manifiesto la importancia de una ciberseguridad eficaz para la supervivencia empresarial en la era digital.

Sin embargo, lograr una protección adecuada presenta desafíos particulares para las PYMEs. Estas organizaciones suelen enfrentar amenazas diversas desde campañas de phishing y malware hasta ransomware y ataques dirigidos a sus aplicaciones web pero carecen de los recursos financieros y técnicos que disponen las grandes corporaciones para contrarrestarlas. En general, muchas PYMEs son especialmente vulnerables, ya que “afrontan retos digitales con recursos limitados y, en ocasiones, con desconocimiento de las amenazas que conllevan”. La falta de personal especializado en seguridad, junto con presupuestos altos, deriva en lagunas importantes de protección: no siempre se monitoriza la red en busca de comportamientos anómalos, no se cuenta con mecanismos sólidos de detección de intrusiones, y la prevención de fugas de información suele quedar relegada o confiada únicamente a medidas básicas. Todo ello ocurre en un contexto donde los ataques no solo van en aumento (INCIBE gestionó 107.500 incidentes en 2022, un 15 % [3] más que el año anterior), sino que además los incidentes críticos están creciendo exponencialmente, poniendo en riesgo la continuidad del negocio.

En respuesta a esta problemática, organismos e iniciativas tanto nacionales como europeas han comenzado a centrar su atención en las PYMEs. La Unión Europea, consciente del papel fundamental de estas empresas en la economía, ha incluido a muchas de ellas dentro del alcance de la nueva Directiva NIS2 para reforzar su ciberresiliencia [4]. En España, instituciones como INCIBE impulsan programas de sensibilización y apoyo específico (Protege tu Empresa, Kit Digital, ayudas de Activa Ciberseguridad, etc.), reconociendo que la ciberseguridad de las PYMEs es un asunto de interés general. No obstante, sigue existiendo una brecha importante entre las necesidades de seguridad de las PYMEs y las soluciones disponibles en el mercado que puedan permitirse o gestionar sin un departamento técnico dedicado.

En este Trabajo Fin de Grado se plantea abordar dicha brecha. La idea central es explorar y justificar la viabilidad de una solución accesible, profesional y asequible basada en Snort 3 sobre una plataforma de bajo coste (Raspberry Pi), complementada con una interfaz web de

gestión y paneles visuales de Grafana –una solución que denominaremos R-Snort– para cubrir las necesidades actuales de ciberseguridad de las PYMEs españolas. En los siguientes apartados se detallarán la motivación y objetivos concretos del proyecto, así como un análisis del estado del arte y de los requerimientos de seguridad más apremiantes para este sector empresarial.

## Motivación

La motivación de este proyecto surge de la constatación de una necesidad real y urgente: las pequeñas y medianas empresas se encuentran en la mira de los cibercriminales, pero no disponen de las herramientas ni conocimientos adecuados para protegerse. A pesar de ser blanco de la mayoría de ataques, muchas PYMEs en España aún “no cuentan con el presupuesto suficiente para hacer frente a los ciberataques”, y más preocupante incluso, carecen de concienciación y conocimiento sobre las soluciones a su alcance. Expertos en seguridad señalan que esta falta de concienciación es un factor importante –en ocasiones mayor obstáculo que el económico–, pues existen medidas de bajo coste que podrían mitigar gran parte de las amenazas si las empresas supieran cómo aplicarlas. En otras palabras, el problema no es solo qué falta de recursos, sino también falta de accesibilidad a soluciones de ciberseguridad adaptadas a las limitaciones de las PYMEs.

Actualmente, las opciones profesionales en el mercado para monitorizar redes, prevenir fugas de datos o detectar intrusiones suelen implicar costosas inversiones en equipos especializados, licencias de software o servicios gestionados (firewalls de nueva generación, sistemas de prevención de intrusiones, plataformas SIEM, etc.). Estas soluciones están pensadas para empresas con departamentos de TI consolidados y con presupuesto holgado, lo que deja a las PYMEs en una situación de desventaja: o bien operan sin las debidas medidas de seguridad (asumiendo riesgos elevados), o intentan implementar herramientas gratuitas/open-source por su cuenta, encontrándose con dificultades técnicas de configuración y mantenimiento. Por ejemplo, Snort –un sistema de detección de intrusiones de código abierto ampliamente reconocido– requiere experiencia para su despliegue y gestión, lo que suele exceder las capacidades del personal de TI generalista con el que cuentan las PYMEs típicas. De igual modo, soluciones de prevención de fuga de información o monitorización de red continua se perciben como complejas y fuera del alcance práctico de estas organizaciones.

Este TFG nace con la motivación de democratizar el acceso a la ciberseguridad para las PYMEs, aprovechando herramientas open-source de probada eficacia pero integrándolas en una plataforma que minimice las barreras de entrada. La elección de una Raspberry Pi como base responde al objetivo de abaratizar costes de hardware al máximo, a la vez que proporciona la flexibilidad de un entorno GNU/Linux completo. Snort 3 se adopta por ser la evolución moderna de uno de los IDS más fiables a nivel industrial, ahora con mejoras de rendimiento y usabilidad que lo hacen más adaptable a entornos modestos. Al desarrollar una interfaz web amigable y complementarla con la visualización de datos mediante Grafana, buscamos que la solución resultante R-Snort sea utilizable por administradores no especializados, permitiéndoles vigilar el tráfico de su red, recibir alertas de intrusiones o comportamientos sospechosos, y supervisar potenciales fugas de información de forma sencilla e intuitiva.

En resumen, la motivación del proyecto se sustenta en tres pilares: (1) la necesidad palpable de mejorar la seguridad en PYMEs ante el aumento de amenazas, (2) la oportunidad de combinar tecnología open-source y hardware económico para crear una herramienta adaptada a dichas necesidades, y (3) la convicción de que una solución accesible y de bajo coste puede marcar la diferencia evitando incidentes que podrían suponer el cierre de muchas pequeñas empresas. Atendiendo a esta motivación, a continuación se definen los objetivos concretos que se pretenden alcanzar.

## Objetivos

El objetivo general de este Trabajo Fin de Grado es diseñar, implementar y validar un sistema integral de monitorización y detección de intrusiones orientado a PYMEs, basado en Snort 3 sobre Raspberry Pi (R-Snort), con interfaz web de gestión y visualización mediante Grafana, que responda eficazmente a las necesidades de seguridad de este tipo de organizaciones.

De este objetivo principal se desprenden los siguientes objetivos específicos:

1. Diseñar la arquitectura de R-Snort, seleccionando los componentes adecuados (hardware Raspberry Pi y software Snort 3, junto con herramientas de apoyo como motores de almacenamiento de logs, dashboards de Grafana, etc.) y configurando una instancia optimizada de Snort que pueda funcionar de forma estable y eficiente en un entorno de recursos limitados (Complemento del TFG). Esto implica ajustar la carga de reglas, parámetros de rendimiento y posibles complementos para asegurar que la Raspberry Pi pueda analizar el tráfico en tiempo real sin degradación notable.
2. Desarrollar una interfaz web intuitiva para la gestión de Snort y la visualización de eventos de seguridad. La interfaz deberá permitir realizar las tareas más comunes (por ejemplo, actualizar reglas, iniciar/detener la captura de tráfico, revisar alertas) sin necesidad de recurrir a la línea de comandos, haciendo la solución más accesible a personal no experto. Asimismo, se integrará Grafana como sistema de visualización para presentar métricas e indicadores de la red (p. ej., volumen de tráfico, alertas por tipo, tendencias temporales) de forma gráfica y comprensible.
3. Validar el sistema R-Snort en un escenario representativo de PYME, verificando que cumple con los requisitos identificados. Esto incluirá pruebas de funcionalidad, de rendimiento (medir el tráfico máximo que puede manejar la Raspberry Pi con el ecosistema instalado y el impacto en tiempos de respuesta), y de usabilidad (evaluar si un usuario con conocimientos técnicos básicos puede manejar la interfaz y entender la información proporcionada). Los resultados de esta validación permitirán determinar en qué medida la solución efectivamente mejora la postura de seguridad de una PYME típica y qué limitaciones o consideraciones deben tenerse en cuenta en su despliegue real.

Con estos objetivos, se pretende que el TFG no solo culmine en un prototipo funcional, sino también en una justificación bien fundamentada de por qué dicha solución es adecuada para las PYMEs y cómo contribuye a cerrar la brecha existente entre los riesgos que enfrentan y los medios de protección actualmente a su disposición.

## Fases de la realización y cronograma

El desarrollo del TFG R-Snort ha estado marcado por una evolución intensiva, en la que la planificación inicial y la realidad del día a día rara vez coincidieron. Si algo queda claro tras repasar el cronograma real es que la flexibilidad y la capacidad de adaptación resultan tan valiosas como cualquier conocimiento técnico. El proceso, lejos de ser lineal, avanzó a base de iteraciones, ajustes y, en ocasiones, pasos atrás para luego avanzar con más claridad.

### ■ Planificación y arquitectura (abril 2025)

El punto de partida real del proyecto fue la definición del diseño de la arquitectura frontend-backend. Durante esta etapa, gran parte del esfuerzo se invirtió en entender cómo debía articularse la comunicación entre los distintos módulos: la API, el backend en Spring Boot y el frontend en Angular. Aunque sobre el papel todo parecía encajar, la experiencia pronto enseñó que cada elección técnica —desde la gestión de usuarios hasta la integración con Grafana— implicaba renuncias, refactorizaciones y más de una revisión en profundidad de las ideas iniciales.

### ■ Desarrollo intensivo y pivotes técnicos (primera mitad de mayo 2025)

Con la arquitectura ya clara, comienza la etapa de implementación. En paralelo se trabajó en la creación del frontend Angular y la implementación del backend Spring Boot, obligando a priorizar unas tareas sobre otras según surgían problemas o bloqueos. El desarrollo del sistema de login y verificación, así como la integración de dashboards dinámicos para la visualización de los datos, requirieron investigar soluciones y adaptar el sistema a nuevas necesidades no previstas y soluciones que cumplieran con normas de seguridad. No hubo grandes pausas: la sucesión de tareas como la gestión de reglas personalizadas, la visualización de alertas en tiempo real y la descarga/archivado de logs se fue solapando, encadenando sprints de trabajo.

### ■ Refinamiento, pruebas y validación (segunda mitad de mayo 2025)

Una vez implementadas las funcionalidades principales, el foco cambió hacia la validación práctica y la mejora continua. Las pruebas funcionales y las pruebas de rendimiento del sistema no solo sirvieron para detectar errores, sino también para ajustar configuraciones de las automatizaciones, pulir detalles de usabilidad y comprobar hasta qué punto el sistema soportaba cargas reales. La integración de métricas y gráficos exigió replantear parte del backend y afinar queries para no penalizar el rendimiento, mientras que la gestión de logs evidenció la necesidad de mantener una documentación clara y procedimientos automáticos para facilitar el mantenimiento futuro.

### ■ Redacción del documento y revisión final

La redacción y revisión del documento supusieron el último gran sprint, combinando jornadas intensas de escritura técnica con revisiones profundas para garantizar la coherencia y la claridad del texto final.

### ■ Análisis del cronograma y esfuerzo real

Las gráficas de planificación y carga horaria son ilustrativas. A pesar de una estimación inicial razonable, el tiempo real dedicado a muchas tareas (en especial, el desarrollo del módulo central y la integración de métricas) superó ampliamente lo previsto. Resulta

tentador culpar a los “imprevistos”, pero la realidad es que los proyectos tecnológicos —y más aún en solitario— tienden a requerir mucha más iteración y refino de lo esperado. Cada decisión arrastra efectos secundarios: una mejora en la interfaz puede complicar el backend, una funcionalidad extra supone más pruebas y documentación, y así sucesivamente.

Como se aprecia en la Figura 1, el cronograma de ejecución muestra una progresión marcada por fases diferenciadas, aunque no siempre simétricas en cuanto a tiempo invertido. Por otro lado, la Figura 2 revela una diferencia entre el tiempo inicialmente estimado y el tiempo real empleado por categoría, lo cual subraya la importancia de dejar márgenes de planificación más amplios en futuros desarrollos.

El resultado final, reflejado en las imágenes adjuntas, habla de un esfuerzo sostenido, a menudo desigual, pero siempre orientado a superar obstáculos y cerrar cada fase con una solución mejor que la anterior.

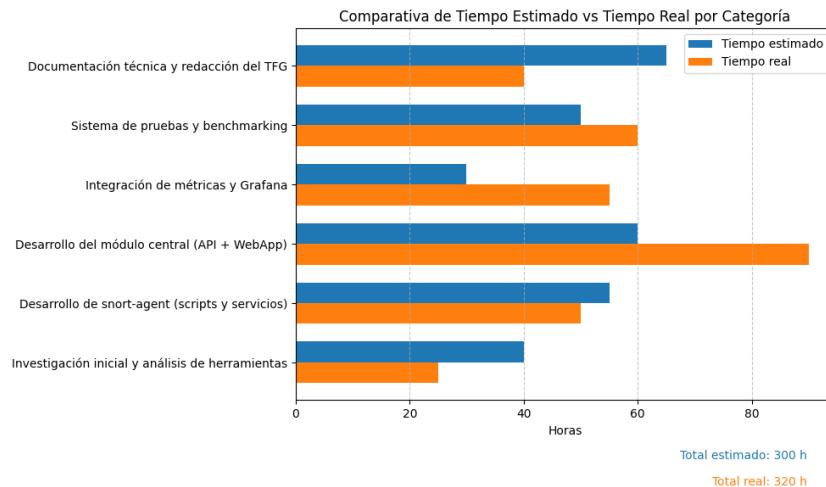


Figura 1: Cronograma de ejecución del TFG R-SNORT.

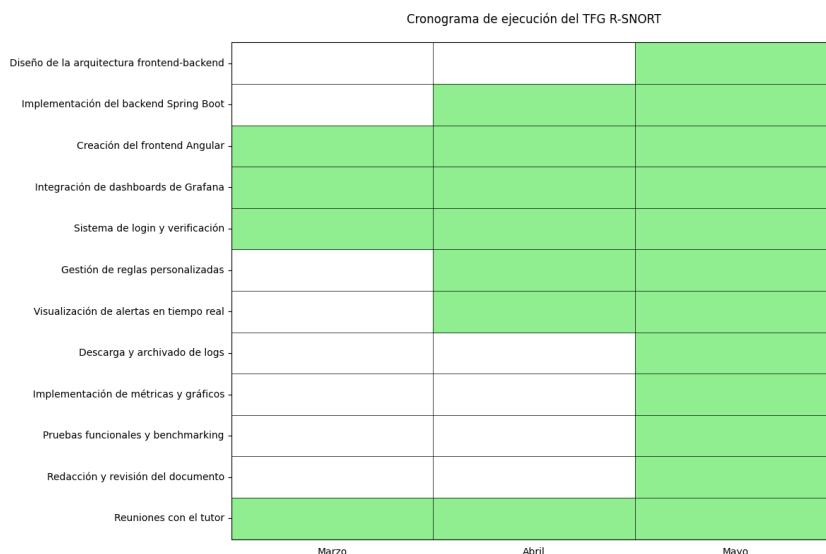


Figura 2: Comparativa de tiempo estimado vs. tiempo real por categoría.

En resumen, el desarrollo del TFG R-Snort ha sido un proceso de aprendizaje constante, donde la gestión real del tiempo y la adaptación a problemas inesperados han resultado tan importantes como el propio resultado técnico. Si algo queda para futuros proyectos es que toda planificación es solo una hipótesis inicial: lo realmente valioso es la capacidad de adaptarse, perseverar y terminar el viaje con una solución de calidad.

## Estructura y metodología

El presente documento está estructurado en varios capítulos que permiten seguir el desarrollo del proyecto de manera lógica y progresiva.

### ■ Introducción, Motivación y Objetivos

El documento comienza contextualizando la problemática de la ciberseguridad en PYMEs, explicando la motivación personal y profesional detrás del proyecto, así como los objetivos generales y específicos que se han perseguido durante su desarrollo.

### ■ Fases de la realización y cronograma

En este apartado se describe cómo se ha organizado y ejecutado el trabajo a lo largo del tiempo, recogiendo las distintas etapas del desarrollo: desde la planificación inicial y el diseño de la arquitectura, pasando por la implementación de los distintos módulos, hasta las fases finales de pruebas, validación y documentación.

### ■ Capítulo 1: Sistemas de detección de intrusos: Snort y frontends

Este primer capítulo técnico introduce al lector en el mundo de los sistemas IDS y NIDS, explicando sus conceptos y diferencias principales. Se describe el sistema R-Snort desarrollado en este trabajo, se comparan diferentes frontends existentes para Snort y se analiza el estado del arte de las interfaces web en este ámbito.

### ■ Capítulo 2: Diseño e implantación de un frontend para R-Snort

Este capítulo es el núcleo del documento y detalla el proceso de diseño y construcción de la solución propuesta. Se presentan las especificaciones y requisitos del sistema, se describen el entorno hardware y software utilizado, y se profundiza en la arquitectura del sistema, el diseño de la interfaz de usuario y el esquema de la base de datos. Además, se detallan las fases de implementación tanto del backend como del frontend, así como la automatización de la instalación mediante scripts dedicados.

### ■ Capítulo 3: Caso práctico — utilización del frontend de R-Snort

En este capítulo se describe cómo se ha desplegado y utilizado el sistema en un entorno realista. Se documentan los pasos de instalación, se muestran capturas de la interfaz y se explican las pruebas funcionales y de rendimiento realizadas, incluyendo la metodología seguida y los resultados obtenidos.

### ■ Resultados y discusión

A continuación, se presentan y analizan los resultados obtenidos tras la validación del sistema.

- **Conclusiones y trabajo futuro**

Para finalizar se señala la relevancia práctica de la solución, así como recomendaciones para mejorar o adaptar el sistema en adelante.

- **Bibliografía y anexos**

El documento se cierra con un compendio de referencias bibliográficas y anexos que sirven de manual rápido de la plataforma y de repositorio de recursos prácticos relacionados con el desarrollo, la instalación y la utilización de R-Snort.

En conjunto, la estructura del documento está pensada para que el lector pueda seguir el recorrido completo del proyecto.



# 1. Sistemas de detección de intrusos: Snort y frontends

## 1.1. IDS / NIDS

Un *Intrusion Detection System* (IDS), o sistema de detección de intrusos, es una herramienta de seguridad cuya función principal es identificar accesos no autorizados o comportamientos anómalos en un sistema o red informática [6]. Estos sistemas analizan los eventos de la red o del host en tiempo real buscando patrones que puedan indicar amenazas, como ataques de denegación de servicio, escaneos de puertos o intentos de intrusión [7]. En general, un IDS no actúa directamente sobre el tráfico; se limita a monitorizar y generar alertas para notificar a los administradores cuando detecta actividades sospechosas o violaciones a las políticas de seguridad.

Existen dos grandes categorías de IDS según el ámbito que vigilan: los basados en host (HIDS) y los basados en red (NIDS). Un HIDS se despliega en un equipo específico y analiza los registros (*logs*) y actividades de ese sistema para descubrir intrusiones locales (por ejemplo, modificaciones no autorizadas, accesos indebidos, etc.). Por otro lado, un *Network IDS* o NIDS inspecciona el tráfico de una red completa o segmento de red para detectar amenazas que transitan por ella. El NIDS examina todos los paquetes que atraviesan la red en tiempo real, buscando en ellos patrones o firmas conocidas de ataques (malware, *port scanning*, explotación de vulnerabilidades, etc.) y puede detectar tanto tráfico malicioso entrante como saliente. Debido a su naturaleza pasiva (escucha en modo promiscuo una copia del tráfico), un NIDS no introduce prácticamente latencia ni altera el flujo de datos en la red que vigila.

Es importante distinguir un IDS de un sistema de prevención de intrusos o IPS (*Intrusion Prevention System*). La diferencia es que el IDS opera de forma pasiva (detectando y alertando sobre posibles ataques), mientras que un IPS actúa de forma activa/intervencionista: un IPS posee todas las capacidades de detección de un IDS pero, adicionalmente, puede bloquear o impedir automáticamente el tráfico malicioso una vez identificado [5]. En otras palabras, el IDS avisa de una intrusión, pero es el administrador quien debe tomar acciones (por ejemplo, actualizar reglas de cortafuegos o aislar equipos comprometidos); en cambio, un IPS está situado en línea en la red y, al detectar un ataque, puede descartarlo o cortarlo en el momento. Por este motivo, a menudo se habla de sistemas IDPS (detección y prevención) cuando una misma solución combina ambas facetas.

Diversos ejemplos ilustran cada tipo de sistema. En el ámbito de host (HIDS) se pueden citar soluciones como OSSEC, Wazuh o Samhain, que monitorizan archivos de log y actividades de un servidor específico. En el ámbito de red (NIDS), destacan herramientas de código abierto ampliamente utilizadas como Snort, Suricata o Bro/Zeek. En particular, Snort se ha convertido en el estándar de facto con el que se comparan todos los IDS de red desde hace más de dos décadas [11]. Snort es un NIDS (e IPS) open source que emplea un conjunto actualizado de *reglas* de detección para identificar patrones de tráfico malicioso; cuando un paquete o flujo coincide con

alguna firma o criterio definido en sus reglas, Snort genera una alerta que notifica del posible incidente [8]. Además, Snort puede desplegarse en modo *inline* (en línea) actuando como IPS, de forma que no solo detecte sino que también bloquee aquellos paquetes que violen las reglas de seguridad. Esta versatilidad ha hecho que Snort sea una referencia obligada en IDS de red tanto para uso personal como empresarial, contando con una amplia comunidad que contribuye con reglas, mejoras y soporte.

## 1.2. R-Snort

R-Snort es la denominación del sistema desarrollado en este proyecto, el cual consiste en una solución integral de monitorización y detección de intrusos basada en Snort 3 sobre hardware de bajo coste (una plataforma Raspberry Pi). En esencia, R-Snort implementa un sensor NIDS autónomo que aprovecha las mejoras de la nueva generación de Snort junto con una interfaz web para la gestión y visualización remota de eventos. A diferencia de un IDS tradicional aislado, R-Snort se concibe de forma modular con componentes que automatizan la recolección de datos, el análisis y la presentación de la información de seguridad, todo ello utilizando únicamente tecnologías abiertas.

El **núcleo de detección** de R-Snort lo constituye Snort 3 ejecutándose en una Raspberry Pi. Snort 3 (también conocido como Snort++) es una reimplementación modernizada del motor Snort que aporta importantes ventajas técnicas respecto a la rama 2.X. Por ejemplo, Snort 3 fue reescrito en C++ para lograr una base de código más modular y fácil de mantener. Incorpora soporte nativo de multiproceso (hilos) y uso de memoria compartida, permitiendo explotar mejor los procesadores multi-núcleo y ofreciendo mayor rendimiento y escalabilidad en la inspección de tráfico. Asimismo, Snort 3 introduce un sistema flexible de complementos (*plugins*) e integración con el lenguaje Lua (LuaJIT) para extender sus funcionalidades, por ejemplo añadiendo nuevas opciones de reglas o analizadores de protocolos de forma más sencilla que antes. La sintaxis de las reglas de detección también se refinó para hacerlas más concisas y fáciles de escribir, reduciendo partes innecesarias y optimizando la velocidad de evaluación [9]. Todas estas mejoras hacen de Snort 3 un motor más adaptable, eficiente y potente para nuestro sistema de detección de intrusos.

En R-Snort, Snort 3 actúa como sensor de red, inspeccionando el tráfico (por ejemplo, mediante una interfaz en modo promiscuo o conectado a un puerto espejo de switch) y generando alertas de intrusión en formato JSON. Dichas alertas son procesadas y almacenadas para su consulta mediante la capa de **back-end**, que está implementada con un servidor web desarrollado en Spring Boot (framework Java) siguiendo una arquitectura de API REST. Este servidor intermedia entre el sensor y la interfaz de usuario, proporcionando servicios como el registro de alertas en una base de datos, la gestión de las reglas activas, y la exposición de endpoints seguros para que los administradores consulten el estado del sistema o apliquen configuraciones de forma remota.

La **interface web** o frontend de R-Snort se ha construido como una *single-page application* moderna utilizando Angular (framework JavaScript/TypeScript). A través de esta webapp,

accesible desde cualquier navegador, el usuario puede visualizar en tiempo real las alertas generadas por Snort, revisar estadísticas históricas, filtrar y buscar eventos por múltiples criterios, así como realizar tareas de administración del sensor (activar/desactivar reglas, reiniciar el servicio IDS, cargar actualizaciones, etc.). El diseño frontend se ha orientado a la simplicidad y claridad en la presentación de datos, inspirándose en las mejores prácticas de UX de aplicaciones de monitorización.

Para enriquecer la experiencia de monitorización, R-Snort integra además la plataforma **Grafana** como herramienta de visualización de datos de seguridad. Grafana es un software open source ampliamente utilizado para construir paneles de control interactivos, y en este proyecto se emplea para generar gráficas y paneles personalizados a partir de los datos de alertas y métricas del IDS. Por ejemplo, se han creado dashboards que muestran el número de alertas por unidad de tiempo, la clasificación de las alertas por severidad o tipo de ataque, e incluso mapas de calor de IP de origen/destino más detectadas. Este enfoque sigue la línea de otras implementaciones de comunidad que utilizan Grafana para visualizar eventos de Snort (a menudo en conjunción con bases de datos de tiempo real como Elasticsearch/Graylog) [10]. En R-Snort, Grafana se conecta al almacenamiento de alertas del back-end y permite tener, dentro de la misma solución, una vista gráfica y analítica de la seguridad de la red en tiempo real.



(a) Logo de Grafana-Labs.



(b) Ejemplo de posible dashboard de Grafana.

En conjunto, R-Snort supone una propuesta de IDS/NIDS completo de bajo coste y código abierto. Combina el potente motor de detección de Snort 3 (sensor) con un ecosistema web moderno (Spring Boot/Angular en el servidor y cliente) y herramientas de visualización profesionales (Grafana) para proporcionar a pequeñas y medianas empresas una plataforma accesible para proteger sus redes. Toda la solución se despliega sobre hardware económico (una Raspberry Pi como nodo sensor), lo que reduce la barrera de entrada en términos de inversión. Pese a su sencillez de despliegue, el sistema mantiene un enfoque modular y escalable: cada componente (detección, almacenamiento, visualización, gestión) está desacoplado, permitiendo en el futuro reemplazar o ampliar funcionalidades (por ejemplo, agregar más sensores en varias ubicaciones reportando al mismo back-end, o incorporar nuevas fuentes de datos de seguridad). En resumen, R-Snort demuestra cómo es posible aprovechar tecnologías abiertas actuales para construir un IDS integrado, manejable vía web y orientado a entornos con recursos limitados, sin incurrir en los altos costes ni complejidad de soluciones comerciales tradicionales.



Figura 1.2: Logo proyecto R-Snort.

### 1.3. Frontends más utilizados de Snort

A lo largo de la evolución de Snort han surgido múltiples aplicaciones frontend para facilitar la gestión y análisis de las alertas generadas por este IDS. Estos frontends web proveen consolas gráficas donde los eventos de Snort pueden visualizarse, filtrarse y reportarse de forma más amigable que mediante los logs en texto plano. A continuación se describen algunos de los frontends históricos más populares asociados a Snort –como Snorby, BASE, Aanval, Sguil o Snort Report– incluyendo sus características principales, enfoques y limitaciones, para luego comparar sus conceptos con la propuesta de nuestro R-Snort.

**BASE (Basic Analysis and Security Engine)** es uno de los frontales web clásicos para Snort. Nació como una continuación del proyecto anterior ACID (*Analysis Console for Intrusion Databases*), el cual fue un pionero en proveer una interfaz web para consultar las alertas registradas por Snort en una base de datos. ACID, desarrollado a inicios de los 2000, quedó discontinuado alrededor de 2003 [12], pero BASE retomó su código y lo mejoró, añadiendo nuevas funciones y compatibilidad con múltiples idiomas. Al igual que ACID, BASE está escrito en PHP y se apoya típicamente en un stack LAMP (Linux-Apache-MySQL-PHP). Snort almacena las alertas en una base de datos relacional (por ejemplo MySQL) –usando para ello complementos como Barnyard2– y BASE ofrece consultas, gráficos básicos y gestión de alertas desde una página web dinámica. Durante muchos años, BASE fue la interfaz preferida por la comunidad, llegando a superar las 200.000 descargas [13]. Entre sus fortalezas estaban la simplicidad de despliegue y su funcionalidad probada para navegación y búsqueda en los eventos (permitiendo ordenar por fecha, tipo de ataque, IP, etc., y ver detalles de cada alerta). Como limitaciones, al ser una herramienta concebida hace más de 15 años, su interfaz resulta poco moderna y no ofrece visualizaciones avanzadas; además depende de tecnologías y librerías ya obsoletas, lo que puede dificultar su instalación en entornos actuales. A pesar de planes para rediseñar BASE e incluso cambiar su formato de base de datos, el proyecto perdió ímpetu tras la salida de sus mantenedores originales y su desarrollo activo se ha ralentizado considerablemente.

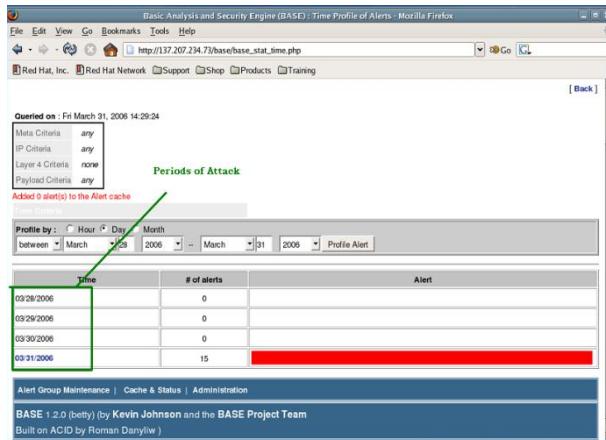


Figura 1.3: Interfaz de BASE.

**Snort Report** es otra solución veterana, centrada en proporcionar informes rápidos del estado del IDS. Surgido alrededor de 2001, Snort Report se distribuía como un módulo adicional ligero para obtener “instantáneas” de las alertas más recientes y resumir la actividad de Snort en la red [14]. A diferencia de BASE, que brinda múltiples vistas y filtros de análisis forense, Snort Report se enfocaba en la monitorización en tiempo real: presentaba en una pantalla principal un tablero con las alertas actuales (número de eventos, tipo más frecuente, origen de las últimas alarmas, etc.), permitiendo al administrador hacerse una idea inmediata de lo que estaba ocurriendo en su sensor IDS. Era una aplicación sencilla en PHP que leía directamente de la base de datos de Snort o de los archivos de log. Entre sus ventajas estaba la facilidad de uso y la mínima configuración necesaria. Sin embargo, también ofrecía menos profundidad analítica que otras herramientas (no tenía tantas opciones de búsqueda histórica o correlación) y con el tiempo fue quedando relegada en favor de frontends más completos. Aun así, para pequeñas implementaciones Snort Report fue útil por su simplicidad. Con el lanzamiento de Snort 2.x y la aparición de otros dashboards más sofisticados, Snort Report dejó de actualizarse regularmente y hoy en día se considera descontinuado.

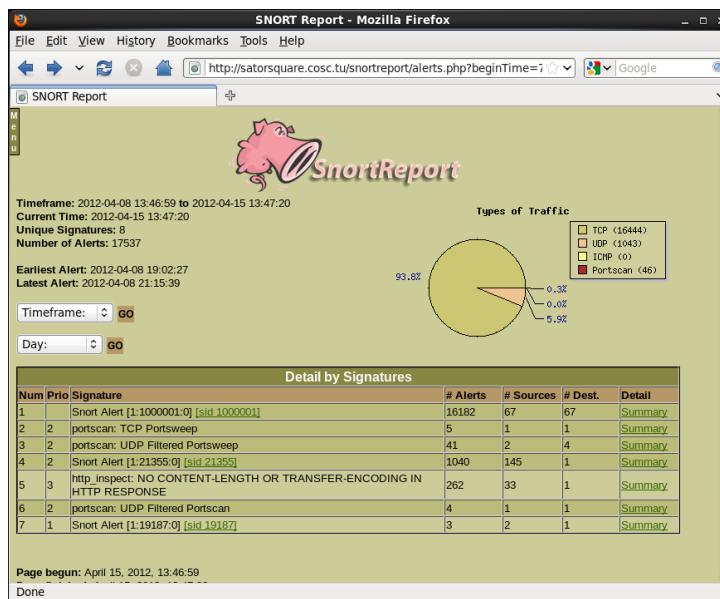


Figura 1.4: Interfaz de Snort Report.

**Aanval** representa una aproximación distinta, orientada a entornos empresariales que buscaban una solución más completa de tipo SIEM integrando a Snort. Es un producto comercial (desarrollado por Tactical FLEX, Inc.) que desde 2003 ha ofrecido soporte para recoger y correlacionar eventos de Snort, Suricata y fuentes de logs generales (syslog) en una plataforma unificada [15]. Aanval se caracteriza por una interfaz web propietaria bastante pulida, con dashboards personalizables, mapas de topologías, alertas en tiempo real y gestión centralizada de múltiples sensores Snort. A lo largo de sus numerosas versiones, ha incorporado funcionalidades avanzadas como: clasificación de eventos por categorías de ataque, generación de informes ejecutivos, alertamiento vía correo/SMS, e integración con herramientas externas (por ejemplo, ticketing de incidentes). En esencia, Aanval va más allá de un simple visor de alertas y se posiciona como un centro de operaciones de seguridad (*Security Operations Center* simplificado) para quienes despliegan Snort. Entre sus fortalezas está la robustez y amplitud de características, así como su continuidad en el tiempo (es uno de los frontends para Snort con más larga trayectoria, con mantenimiento activo por más de 15 años). Su principal desventaja, desde la perspectiva de nuestro proyecto, es que no es software libre; si bien tuvo versiones gratuitas limitadas, la versión completa de Aanval es de pago, lo que supone una barrera para PYMEs con presupuesto reducido. Asimismo, su enfoque todo-en-uno puede resultar complejo de desplegar en escenarios muy pequeños. No obstante, conceptualmente Aanval demuestra cómo una interfaz web puede escalar para administrar múltiples sensores Snort y agregar inteligencia de seguridad a partir de sus alertas.

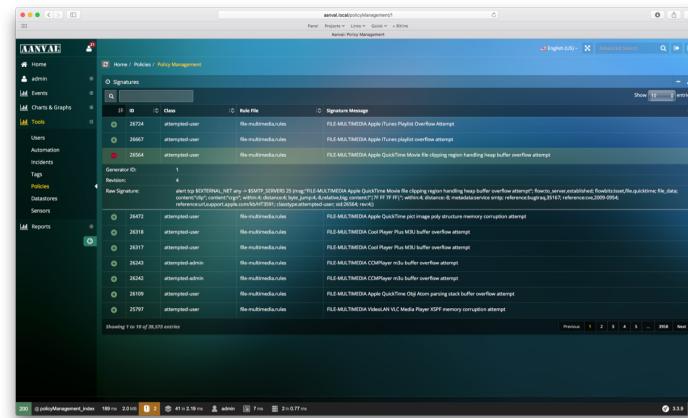


Figura 1.5: Interfaz de Aanval.

**Sguil** es un frontend ampliamente reconocido, aunque su filosofía y arquitectura difieren notablemente de las opciones antes mencionadas. Sguil nació alrededor de 2004 impulsado por Bamm Visscher como parte del paradigma de *Network Security Monitoring* (NSM). A diferencia de BASE o Snorby, que son aplicaciones web puras, Sguil es una solución cliente-servidor pesada escrita en Tcl/Tk que provee una consola para analistas de seguridad. Consta de tres componentes principales: sensores (por ejemplo Snort ejecutándose en modo sensor), un servidor central que recopila eventos, y uno o varios clientes GUI que los analistas utilizan para conectarse y revisar los datos [16]. Sguil se distingue por proporcionar acceso a información muy detallada: su interfaz gráfica muestra los eventos de Snort en tiempo real (pestaña de *RealTime Events*), y permite al analista profundizar en cada alerta consultando datos de sesión (conexiones relacionadas) e incluso ver las capturas completas de los paquetes asociados al evento, gracias a que

integra un sistema de captura continua de tráfico. En esencia, Sguil no solo alerta de un posible ataque, sino que ofrece las evidencias crudas para que un analista las verifique (por ejemplo, reconstruir la sesión TCP o examinar la carga útil del paquete sospechoso). Esto convierte a Sguil en una potente herramienta de análisis e investigación de intrusiones. Muchos administradores empezaban usando BASE para lo básico y luego migraban a Sguil cuando necesitaban capacidades forenses más avanzadas. Sin embargo, toda esta funcionalidad tenía contrapartidas: la aplicación, al estar escrita en Tcl/Tk, resulta menos accesible (no es web, requiere instalar el cliente gráfico) y su usabilidad es más tosca en comparación con las soluciones web modernas. Sguil está muy enfocada al especialista en seguridad, por lo que su curva de aprendizaje es mayor y su interfaz es densa en información técnica. Pese a ello, marcó un hito en cuanto a profundidad de datos disponibles para un IDS y sentó las bases de suites NSM actuales (como Security Onion, que integra Sguil/Squert). Cabe mencionar que existieron frontends web complementarios a Sguil, como Squert, que ofrecían en navegador una vista simplificada de los datos de Sguil, pero incluso estos han ido quedando obsoletos con el tiempo. En resumen, Sguil ofreció una aproximación de “análisis total” del tráfico en torno a Snort, sacrificando la estética y simplicidad a cambio de capacidades analíticas únicas en su momento.

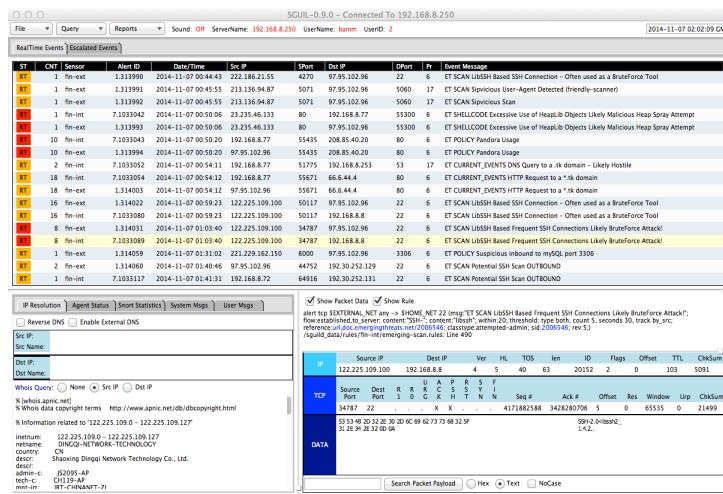


Figura 1.6: Interfaz de Sguil.

**Snorby** es uno de los frontends para Snort más destacables de la década de 2010, pues supuso un intento de modernizar la experiencia de usuario en la gestión de alertas IDS. Presentado originalmente en 2010, Snorby es una aplicación web desarrollada en Ruby on Rails que enfatizaba la interfaz gráfica elegante y la facilidad de uso. Sus principios fundamentales eran la simplicidad y la potencia: el objetivo declarado del proyecto Snorby fue crear una herramienta abierta, gratuita y altamente competitiva para monitorización de redes, dirigida tanto a entornos empresariales como a usuarios particulares [12]. A nivel funcional, Snorby retomó muchas características conocidas de BASE (consultas a la base de datos de Snort, filtros por protocolos, exportación de alertas a CSV/PDF, etc.) pero añadiendo numerosas mejoras. Entre las funcionalidades que incorporó estaban: un panel de inicio con métricas y gráficas de las alertas (*dashboard* dinámico), generación automática de informes diarios, semanales y mensuales enviados por correo, sistema de comentarios y anotaciones colaborativas en cada evento (útil para trabajo en equipo), categorización personalizada de la severidad de alertas, y actualizaciones en tiempo real

de nuevas alarmas vía AJAX. Incluso ofreció integración con captura de paquetes a través de OpenFPC y aplicaciones móviles (desarrollaron un cliente para iOS). Todo ello con una estética web 2.0 atractiva: gráficos interactivos, uso intensivo de HTML5/CSS3, y una experiencia similar a aplicaciones modernas. Snorby fue considerado durante un tiempo el sucesor natural de BASE en entornos donde no se requería la profundidad de Sguil. Otra ventaja fue la facilidad de despliegue relativamente alta para la época, proporcionándose máquinas virtuales preconfiguradas (Insta-Snorby) para probar el sistema rápidamente. No obstante, Snorby también tuvo limitaciones: su instalación manual podía ser compleja debido a dependencias (particularmente, ciertas gemas de Ruby y librerías como ImageMagick que en distribuciones Linux estables estaban desactualizadas). Además, con el tiempo el proyecto dejó de mantenerse activamente (sus últimas actualizaciones datan de mediados de la década de 2010). Esto, sumado a la aparición de otras soluciones integrales (por ejemplo SIEM completos o el propio Security Onion), hizo que Snorby cayera en desuso recientemente. Aun así, su influencia se nota en el énfasis que puso en la usabilidad del análisis de alertas IDS.

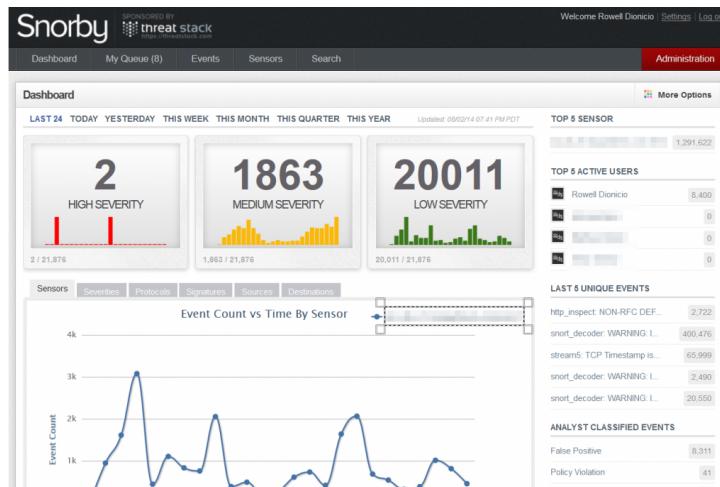


Figura 1.7: Interfaz de Snorby.

En perspectiva, cada frontend de Snort mencionado abordó la necesidad de manejar las alertas de intrusión desde un ángulo distinto: unos priorizaron la simplicidad y rapidez (Snort Report, BASE en sus inicios), otros la profundidad de datos (Sguil), otros la estética y facilidad de manejo (Snorby), u ofrecer un ecosistema integral (Aanval). Muchos de estos proyectos, sin embargo, ya no se actualizan o han quedado técnicamente anticuados para los estándares actuales (por ejemplo, ACID/BASE se basaba en PHP5 y Snorby en Rails 3, entornos que hoy presentan problemas de compatibilidad) [17]. La propuesta R-Snort toma inspiración conceptual de estas soluciones previas pero busca aprender de sus limitaciones para ofrecer algo más acorde a los tiempos actuales. En particular, R-Snort comparte con Snorby la filosofía de una interfaz web amigable y orientada al usuario general, donde la información importante esté fácilmente disponible sin mucha complejidad. Del mismo modo que Snorby perseguía “simplicidad y poder”, nuestra aplicación web Angular pretende ser intuitiva pero sin sacrificar funcionalidades clave (por ejemplo, R-Snort permite comentar o marcar eventos, de forma similar al enfoque colaborativo que introdujo Snorby). Por otro lado, recogemos la idea de Sguil de tener una arquitectura modular con sensores dedicados y un servidor central que consolida eventos [16]; no en vano, R-Snort implementa su sensor Snort en Raspberry Pi y envía las alertas a un servidor

web central para su almacenamiento y análisis, emulando en pequeña escala el esquema sensor-servidor-cliente de Sguil (aunque reemplazando el pesado cliente Tcl por una ligera aplicación web). Además, R-Snort brinda cierta capacidad de inspección de datos más allá de la alerta básica integrando Grafana para visualizaciones; esto se inspira en la filosofía NSM de proveer contexto adicional al analista, si bien nuestro proyecto no llega al nivel de captura completa de paquetes que ofrecía Sguil. En comparación con Aanval, R-Snort busca democratizar el acceso a este tipo de herramientas: optamos por tecnologías 100 % open source y asequibles, evitando licencias comerciales o dependencias propietarias, de forma que incluso pequeñas organizaciones puedan desplegarlo sin trabas. Conceptualmente aprendemos de las lecciones de Aanval en cuanto a consolidar múltiples componentes (detección, base de datos, visualización) en un solo sistema cohesionado, pero simplificando la implementación para que no requiera personal altamente especializado para mantenerlo. En resumen, R-Snort moderniza la idea del frontend de Snort integrando las mejores ideas de proyectos previos (la accesibilidad de Snorby, la arquitectura distribuida de Sguil, la visión general de Snort Report, etc.) y actualizándolas con una arquitectura vigente y orientada a la facilidad de despliegue. El resultado es una solución actualizada que mejora a aquellas históricas en varios aspectos: interfaz web responsiva y actual, instalación sencilla en hardware barato, soporte nativo a Snort 3 (frente a muchas consolas legadas que solo manejaban Snort 2.X), y capacidad de adaptación a las necesidades de monitorización de redes pequeñas con recursos limitados, manteniendo al mismo tiempo un enfoque profesional en la detección de intrusiones.

## 1.4. Comparativa de interfaces web

Antes de abordar el desarrollo de la interfaz propuesta para R-Snort, es importante analizar las herramientas actuales de monitorización de sistemas de detección de intrusos (IDS) a través de interfaces web. A continuación se presentan tres de las principales soluciones existentes – la interfaz web de Snort en **pfSense**, la consola **BASE** para Snort y la plataforma **T-Pot** orientada a Suricata – describiendo sus características técnicas, ventajas y limitaciones desde el punto de vista técnico y de usabilidad. Este análisis proporciona la base conceptual e inspiración para el diseño de R-Snort, identificando funcionalidades deseables y carencias a evitar en el nuevo sistema.

### 1.4.1. Interfaz web de Snort en pfSense

pfSense es un popular cortafuegos de código abierto que puede actuar además como sistema de detección y prevención de intrusiones mediante paquetes adicionales como Snort o Suricata [18]. La interfaz web de administración de pfSense incluye un módulo para Snort, a través del cual es posible **configurar el IDS** (por ejemplo, seleccionando reglas, activando la inspección en determinadas interfaces de red, etc.) y **visualizar las alertas** detectadas. Una vez instalado el paquete de Snort en pfSense, el administrador accede a una sección dedicada donde se listan todas las alertas registradas por el motor IDS. En la Figura 1.8 se muestra un ejemplo de la pantalla de alertas de Snort en pfSense.

Aunque esta integración facilita la gestión unificada (permitiendo, por ejemplo, **habilitar o deshabilitar reglas** desde la misma interfaz del cortafuegos), presenta limitaciones importan-

tes en cuanto a usabilidad para el análisis de eventos. En su pestaña de *Alerts*, pfSense muestra el registro completo de alertas en orden cronológico, pero **no ofrece opciones avanzadas de filtrado o visualización gráfica** de dichos eventos. El analista se encuentra con un listado plano de eventos, lo cual dificulta extraer rápidamente tendencias o detalles específicos cuando el volumen de alertas es elevado. Por ejemplo, no es posible filtrar interactivamente por dirección IP sospechosa, rango de fechas o tipo de alerta desde la interfaz; cualquier filtrado o búsqueda debe realizarse manualmente recorriendo las páginas de resultados. Esta carencia resalta con herramientas especializadas de monitorización, pero es entendible dado que el enfoque principal de pfSense está en la administración del firewall y la configuración del IDS, más que en el análisis exhaustivo de sus alertas.

Cabe destacar que la interfaz de pfSense sí permite realizar ajustes de **configuración sobre Snort** que no están disponibles en otras soluciones de monitorización. El administrador puede modificar parámetros del IDS (por ejemplo, modo de detección IDS/IPS, políticas de prevención o gestión de listas de exclusión) directamente desde el entorno web. Esta capacidad de control es una ventaja técnica significativa de pfSense como plataforma integrada. Sin embargo, desde el punto de vista de visualización de datos y experiencia de usuario, la presentación de las alertas es básica: esencialmente tablas de texto sin agregaciones dinámicas ni gráficos interactivos.

The screenshot shows the pfSense web interface with the following navigation path: Services / Snort / Rules / WAN. The main menu bar includes System, Interfaces, Firewall, Services, VPN, Status, Diagnostics, Gold, and Help. Below the main menu, there are several tabs: Snort Interfaces (selected), Global Settings, Updates, Alerts, Blocked, Pass Lists, Suppress, IP Lists, SID Mgmt, Log Mgmt, and Sync. Underneath these are sub-tabs: WAN Settings, WAN Categories, WAN Rules (selected), WAN Variables, WAN Preprocs, WAN Barnyard2, WAN IP Rep, and WAN Logs. A sidebar on the left lists "Available Rule Categories" with "IPS Policy - Connectivity" selected. The main content area displays "Rule Signature ID (SID) Enable/Disable Overrides" with buttons for SID Actions (Apply, Reset All, Reset Current, Disable All, Enable All). A note below says "When finished, click APPLY to save and send any SID enable/disable changes made on this tab to Snort." The "Selected Category's Rules" section shows a table of rules:

GID	SID	Proto	Source	Sport	Destination	DPort	Message
1	5808	tcp	\$HOME_NET	any	\$EXTERNAL_NET	\$HTTP_PORTS	BLACKLIST User-Agent known malicious user agent - SAH Agent
1	5900	tcp	\$HOME_NET	any	\$EXTERNAL_NET	\$HTTP_PORTS	BLACKLIST User-Agent known malicious user agent - Async HTTP Agent
1	19493	tcp	\$HOME_NET	any	\$EXTERNAL_NET	\$HTTP_PORTS	BLACKLIST URI request for known malicious uri config.ini on 3222.org domain
1	33907	tcp	\$HOME_NET	any	\$EXTERNAL_NET	\$HTTP_PORTS	BLACKLIST User-Agent known malicious user-agent - KAILOOO0871 - Win.Trojan.Dridex
1	26898	tcp	\$EXTERNAL_NET	\$FILE_DATA_POR...	\$HOME_NET	any	BROWSER-PLUGINS Java Applet sql.DriverManager fakedriver exploit attempt
1	27766	tcp	\$EXTERNAL_NET	\$FILE_DATA_POR...	\$HOME_NET	any	BROWSER-PLUGINS Oracle Java Security Slider feature bypass attempt
1	27870	tcp	\$EXTERNAL_NET	\$FILE_DATA_POR...	\$HOME_NET	any	BROWSER-PLUGINS HP LoadRunner WriteFileString ActiveX function call attempt
1	27869	tcp	\$EXTERNAL_NET	\$FILE_DATA_POR...	\$HOME_NET	any	BROWSER-PLUGINS HP LoadRunner WriteFileString ActiveX function call

Figura 1.8: Panel de alertas de Snort en la interfaz web de pfSense.

### 1.4.2. Interfaz BASE para Snort

El *Basic Analysis and Security Engine* (BASE) es una de las interfaces web clásicas para la monitorización de Snort. Derivado originalmente del proyecto ACID (Analysis Console for Intrusion Databases), BASE provee un **frontal web para consultar y analizar las alertas** generadas por un sistema Snort [19]. Este sistema está construido sobre una arquitectura LAMP, empleando un servidor web Apache con páginas PHP que consultan los datos almacenados en una base de datos MySQL. En otras palabras, Snort vuela sus eventos de alerta en una base de datos y BASE permite explotarlos mediante consultas dinámicas.

A diferencia de pfSense, BASE se centra exclusivamente en el **análisis de las alertas** y no ofrece funciones para cambiar la configuración del sensor IDS. Su fortaleza radica en las capacidades de **filtrado y agrupación de información** que brinda al analista. La interfaz permite generar consultas predefinidas y personalizadas: por ejemplo, listar las alertas más recientes únicas, obtener las cinco alertas que más se repiten, o visualizar las direcciones IP origen y destino más frecuentes en los registros de intrusiones. La Figura 1.9 ilustra una de estas visualizaciones, mostrando un resumen de las direcciones IP más detectadas en las alertas. Con estas funciones, BASE facilita identificar patrones de ataque (como hosts atacantes recurrentes o tipos de alertas predominantes) de forma más eficiente que revisando listas planas de eventos.

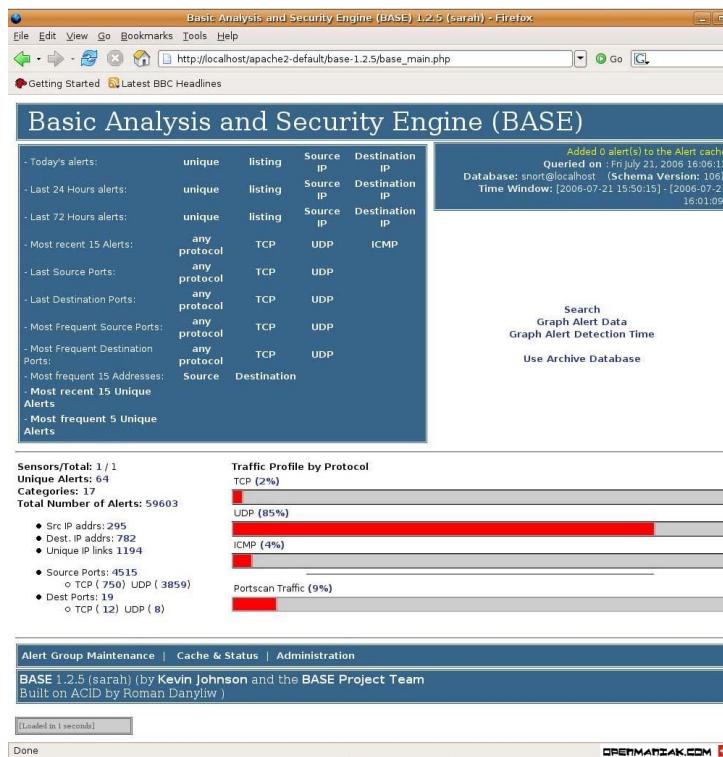


Figura 1.9: BASE.

En términos de **usabilidad**, la interfaz de BASE, si bien es de corte tradicional, resulta útil para un análisis forense o de seguimiento de incidentes, ya que presenta opciones de búsqueda por múltiples criterios (IP, puerto, fecha, tipo de firma, etc.) y organiza los resultados en formato tabular paginado. No obstante, su diseño es menos interactivo y visual comparado con

herramientas modernas: carece de gráficos integrados avanzados o mapas de ataque, y su estética refleja la era en que fue desarrollada (aproximadamente mediados de la década de 2000). Aun así, BASE continúa siendo una referencia en cuanto a funcionalidad de consulta de alertas Snort, proporcionando una base sólida de características sobre la cual se pueden concebir mejoras. Por ejemplo, la idea de incluir filtros por rango temporal o listas de “Top N” alertas en R-Snort toma inspiración directa de este tipo de funcionalidades ya presentes en BASE.

#### 1.4.3. Interfaz web de Suricata en T-Pot

T-Pot es una plataforma integral de *honeypots* mantenida por Telekom Security, que incluye múltiples sensores de seguridad (como Cowrie, Dionaea, conpot, entre otros) y utiliza a Suricata como motor IDS de red. Uno de los atractivos principales de T-Pot es su moderna interfaz de visualización basada en el stack ELK (Elastic Stack). Esta interfaz brinda **paneles de control gráficos e interactivos** para los datos de alerta. En el caso de Suricata, T-Pot ofrece un panel web integrado (basado en Kibana) que permite analizar las alertas detectadas de forma muy visual. Esta interfaz muestra, por ejemplo, el número total de alertas en un intervalo de tiempo, distribuciones por tipos de amenaza o reglas disparadas, y representaciones como histogramas de eventos en el tiempo y mapas geográficos de las IP de origen de ataques. De hecho, T-Pot soporta innumerables opciones de visualización mediante Elastic Stack e incluye mapas de ataque en tiempo real [20]. Esto proporciona una experiencia de monitorización más rica y profesional comparada con las interfaces tradicionales.

Al igual que BASE, la consola de Suricata en T-Pot está enfocada en la **inspección y análisis** de las alertas, sin exponer opciones para reconfigurar el IDS desde la web. Sin embargo, T-Pot supera a BASE en cuanto a presentación: la disponibilidad de gráficos interactivos y estadísticas automáticas facilita la interpretación rápida del estado de seguridad. Además, la interfaz permite aplicar **filtros dinámicos** sobre los datos: por ejemplo, es posible filtrar las alertas por una dirección IP específica para ver sólo eventos relacionados con esa fuente, o limitar la vista a un rango temporal definido (por ejemplo, las últimas 24 horas o los últimos 5 días) para focalizar el análisis en determinados períodos de actividad. Estas capacidades de filtrado por campo y tiempo aportan una gran flexibilidad al analista, similar a la que se obtiene con herramientas SIEM, y son implementadas de forma transparente gracias a la potencia de Elasticsearch en el backend.

Desde el punto de vista técnico, la solución de T-Pot es más pesada en recursos que las otras dos interfaces: requiere correr contenedores Docker con Elasticsearch, Logstash y Kibana, entre otros componentes, lo cual demanda memoria y almacenamiento significativos. No obstante, este costo se ve compensado por la **riqueza funcional** que ofrece en monitorización. En entornos donde se despliega T-Pot, normalmente dedicados a investigación de amenazas o vigilancia de *honeypots*, esta interfaz ha demostrado ser muy efectiva para identificar patrones de ataque complejos y correlaciones entre eventos. La Figura 1.10 muestra un ejemplo del panel principal de Suricata en T-Pot, donde se aprecian varias visualizaciones simultáneas de los datos de alerta.

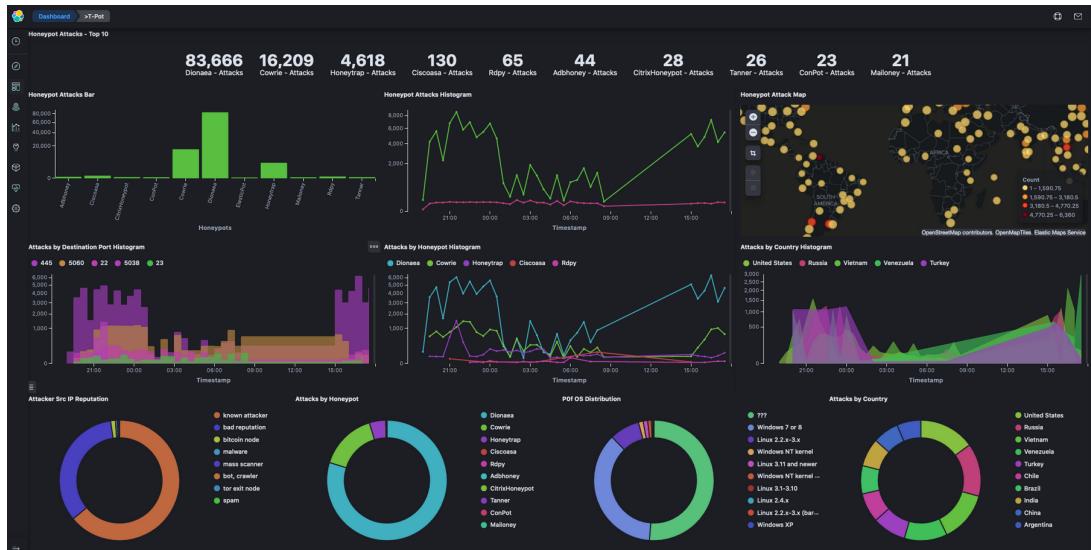


Figura 1.10: Panel principal de Suricata en T-Pot.

Una comparativa resumida de las funcionalidades de estas interfaces se presenta en la Tabla 1.1. Puede observarse que cada solución prioriza un aspecto diferente: pfSense destaca en configurabilidad del IDS, BASE en capacidades de consulta detallada, y T-Pot en visualización avanzada. Estas perspectivas complementarias han sido consideradas en el diseño de R-Snort, buscando integrar lo mejor de cada enfoque. En particular, se procura que R-Snort combine la **gestión configurable** del sensor (inspirada por pfSense) con **mecanismos de análisis exhaustivo y visualización intuitiva** (tomando como referentes las fortalezas de BASE y T-Pot, respectivamente).

Característica	BASE (Snort)	pfSense (Suricata)	T-Pot (Suricata)
Configuración del IDS	No	Sí (completa)	No
Filtrado de alertas	Sí (varios criterios)	No	Sí (por IP, tiempo, etc.)
Visualización gráfica	Limitada (tablas/listas)	No	Sí (gráficos, mapas, dashboards)
Tecnología backend	LAMP (Apache/PHP/MySQL)	Integrado en GUI (PHP)	Elastic Stack (Kibana/Elasticsearch)
Enfoque principal	Análisis forense de alertas	Gestión de IDS/Firewall	<i>Honeypot</i> con monitorización

Tabla 1.1: Comparativa de funcionalidades de las interfaces web de IDS analizadas.



## 2. Diseño e implantación de un frontend para R-Snort

### 2.1. Introducción

En este capítulo se describe el diseño técnico y la implementación de la interfaz web (*frontend*) y la arquitectura general del sistema **R-Snort**. Este sistema integra el motor de detección de intrusiones Snort en su versión 3 ejecutándose sobre una **Raspberry Pi 5**, junto con una aplicación web desarrollada en Angular (versión 19) y un servicio backend basado en Spring Boot (Java 17). El objetivo es convertir la instancia de Snort en un *agente* de seguridad monitorizable de forma remota a través de una interfaz web intuitiva y un conjunto de *dashboards* gráficos para alertas y métricas.

R-Snort adopta una arquitectura modular y desacoplada. Por un lado, la Raspberry Pi actúa como **sensor IDS** ejecutando Snort 3 y exponiendo una **API REST** mediante el backend Spring Boot; por otro lado, un cliente web Angular permite al usuario interactuar con el sistema (visualizar alertas, configurar parámetros, etc.). Además, se integra la plataforma **Grafana** para la visualización en tiempo real de las alertas y métricas de rendimiento a través de paneles personalizados. Todos estos componentes se comunican de forma coordinada para proporcionar una solución unificada de monitorización de intrusiones en la red.

Una de las razones para elegir **Snort 3** como núcleo del sistema es que esta versión ofrece mejoras significativas en eficiencia y escalabilidad respecto a Snort 2.X, incluyendo una arquitectura *multihilo* capaz de aprovechar sistemas multi-núcleo [21]. Esto resulta en mayor rendimiento y extensibilidad, aspectos clave al desplegar un IDS en hardware limitado como Raspberry Pi. En efecto, Snort 3.0 presenta un diseño renovado y un superconjunto de funcionalidades de Snort 2, logrando mejor eficacia, rendimiento, escalabilidad, usabilidad y extensibilidad. Gracias a esta capacidad multihilo de Snort 3, es posible utilizar todos los núcleos de la Raspberry Pi 5 para el procesamiento de tráfico, aumentando la tasa de análisis de paquetes.

Para la implementación del backend web, se decidió utilizar el framework **Spring Boot** por su madurez y las facilidades que ofrece (inicio rápido, servidor web embebido, integración con bibliotecas de seguridad, acceso a bases de datos, etc.). Se consideró incluso una alternativa basada en Python como FastAPI, conocida por su alto rendimiento en servicios REST [22], pero finalmente Spring Boot fue elegido por su estabilidad y por alinear mejor con los conocimientos y requisitos del proyecto (p.ej. facilidad de integración con Snort a bajo nivel) aunque la herramienta de FastAPI ha servido para la implementación de los agentes individuales. La comunicación entre el frontend Angular y el backend Spring Boot se realiza mediante peticiones HTTP a la API REST, utilizando datos en formato JSON. Para proteger estas comunicaciones y restringir el acceso solo a usuarios autorizados, el sistema implementa autenticación basada en **JSON Web Tokens (JWT)**. Un JWT es un estándar abierto que define un método compacto y seguro de transmitir información entre partes en formato JSON [25], lo que permite verificar la

identidad del usuario sin mantener sesiones en el servidor. De este modo, el frontend obtiene un token JWT tras el inicio de sesión y lo incluye en las cabeceras de las peticiones subsecuentes, garantizando un acceso seguro a las funcionalidades de R-Snort.

En la Figura 2.1 se muestra un diagrama general de la arquitectura de R-Snort. En dicha ilustración se aprecian los principales componentes: la Raspberry Pi ejecuta Snort 3 como agente sensor junto al servicio backend (API REST) que interactúa con Snort y con la base de datos de alertas, el servicio de FAST API se encarga de exponer endpoints para controlar cada sensor; el usuario accede desde un navegador web al frontend Angular, el cual consume la API para presentar información y opciones de configuración; finalmente, Grafana se conecta a la fuente de datos de alertas (base de datos) para ofrecer paneles gráficos actualizables en tiempo real.

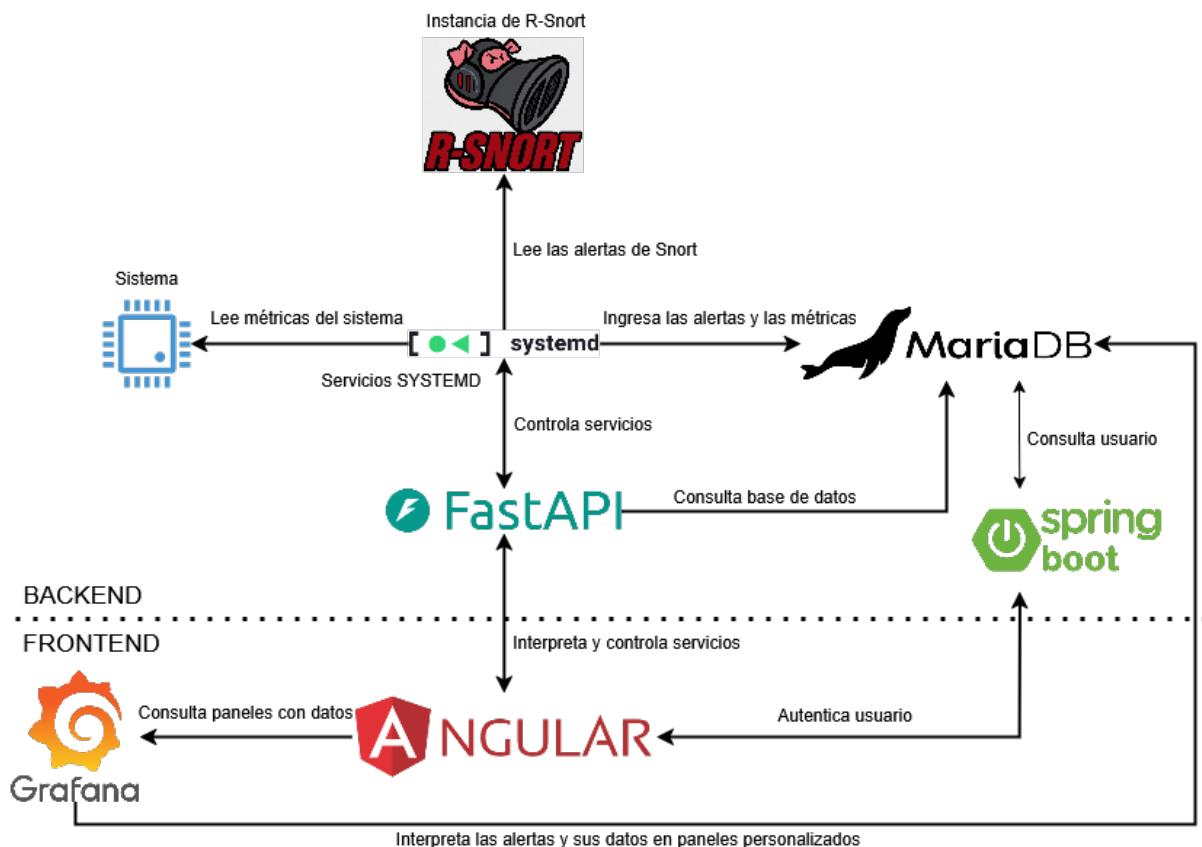


Figura 2.1: Arquitectura general del sistema R-Snort.

En las secciones siguientes se detallan, primero, las especificaciones y requisitos del sistema (funcionales y técnicos) y, a continuación, el entorno de trabajo utilizado, tanto a nivel de hardware (la plataforma Raspberry Pi empleada como agente de seguridad) como de software (componentes instalados, configuraciones relevantes y herramientas auxiliares). Asimismo, se describe brevemente el entorno de desarrollo y pruebas, donde se aclaran las tecnologías usadas para implementar el frontend y backend, así como la configuración de la red de laboratorio empleada para validar el funcionamiento de R-Snort.

## 2.2. Especificaciones del sistema

A continuación, se enumeran las principales **características y requisitos** que se definieron para el sistema R-Snort, abarcando tanto funcionalidades esperadas como consideraciones de diseño e implantación:

- **Motor IDS basado en Snort 3.** El sistema debe emplear Snort en su versión 3 como núcleo de detección de intrusiones. Snort 3 proporciona capacidades mejoradas (p. ej., soporte multihilo, nuevos plugins y sintaxis de reglas mejorada) que incrementan la eficacia y rendimiento de la detección.
- **Plataforma hardware accesible.** El IDS ha de poder ejecutarse sobre una Raspberry Pi, aprovechando la portabilidad y bajo coste de este dispositivo. Esto permite desplegar el sensor en entornos domésticos o de pequeña empresa sin requerir hardware dedicado costoso.
- **Interfaz web para gestión y monitorización.** Se requiere un frontend amigable, accesible vía navegador, que permita al usuario administrar el sistema y visualizar las alertas detectadas. Este frontend se desarrollará como una *Single Page Application* en Angular, garantizando una experiencia interactiva y fluida.
- **Backend REST.** La arquitectura debe seguir un modelo cliente-servidor desacoplado. Para ello, se implementará una API REST en el backend (usando Spring Boot y FAST API) que exponga las funcionalidades necesarias: consulta de alertas, estado de Snort, gestión de configuración, etc. Este backend actúa de intermediario entre Snort, la base de datos y el frontend.
- **Visualización de alertas y métricas.** El sistema debe ser capaz de presentar no solo listados de alertas, sino también gráficos y métricas agregadas. Se integrará **Grafana** para este propósito, configurando *dashboards* que muestren por ejemplo el número de alertas por tipo, tendencias temporales, y métricas de rendimiento (CPU, uso del disco, etc.). Grafana es una plataforma de monitorización y visualización de datos de código abierto ampliamente utilizada en la industria para analítica interactiva [24].
- **Almacenamiento persistente de datos.** Las alertas de intrusión y otros datos relevantes deben almacenarse en una base de datos para permitir su consulta histórica y correlación. Se optó por utilizar una base de datos relacional **MariaDB** (derivada de MySQL) como repositorio principal de alertas.
- **Gestión de reglas y configuración de Snort.** La solución debe facilitar la administración de Snort sin requerir acceder directamente por consola. Esto incluye la posibilidad de agregar reglas IDS (por ejemplo, descargar nuevas reglas de la comunidad o de Snort), eliminar conjuntos de reglas según las necesidades, y ajustar parámetros de configuración de interfaces a través de la web o la API.
- **Seguridad y control de acceso.** Dado que la interfaz de gestión expone funcionalidades críticas (ej. ver alertas de seguridad, modificar reglas), es imprescindible restringir el acceso

solo a usuarios autorizados. Para ello, se integra un mecanismo de autenticación basado en JWT tal como se mencionó, de forma que cada administrador de R-Snort deba iniciar sesión para obtener un token válido. La comunicación entre frontend y backend debe estar protegida (idealmente usando HTTPS para entornos de producción, evitando la transmisión de credenciales o tokens en texto claro).

- **Registro y rotación de logs.** Snort genera archivos de registro de alertas que pueden crecer rápidamente. El sistema debe implementar esquemas de *log rotation* para controlar el tamaño de los logs y evitar agotar el espacio en la Raspberry Pi. Por ejemplo, mediante herramientas del sistema (como `logrotate` en Linux) se rotarán y comprimirán periódicamente los archivos de alertas, manteniendo un histórico reciente y eliminando registros antiguos de forma automatizada.
- **Scripts de instalación automatizada.** Para facilitar la puesta en marcha del sistema, se prepararon *scripts* modulares que automatizan la instalación y configuración de los distintos componentes (base de datos MariaDB, entorno Grafana, despliegue del backend Spring Boot, etc.). Cada módulo de instalación puede ejecutarse independientemente, permitiendo por ejemplo instalar solo el agente Snort en una máquina o solo el servidor web en otra, según la arquitectura deseada. Esto refleja la filosofía modular de R-Snort, haciendo posible su adaptación o escalado a futuros despliegues (por ejemplo, múltiples sensores Snort en distintas Raspberry Pi reportando a un servidor central).
- **Rendimiento aceptable en hardware limitado.** Se exige que el sistema funcione de manera estable en la Raspberry Pi, atendiendo a sus limitaciones de CPU, memoria y E/S. Esto implica optimizar la configuración de Snort (por ejemplo, cargar únicamente las reglas necesarias, ajustar hilos de procesamiento) y minimizar el consumo de recursos del backend y la base de datos. Las pruebas deben verificar que el IDS puede manejar el tráfico de la red objetivo (típicamente redes domésticas o de oficina pequeña) sin pérdidas de paquetes significativas y con latencia aceptable en la interfaz web.

En síntesis, los requisitos anteriores trazan el camino a un sistema completo de detección de intrusos gestionable vía web, construido con tecnologías modernas y enfocado en facilidad de uso, todo ello sobre una plataforma de bajo coste. A continuación, se detalla el entorno de trabajo y la arquitectura de los componentes que dan cumplimiento a estas especificaciones.

## 2.3. Entorno de trabajo

En esta sección se describe el entorno de hardware y software utilizado para implementar R-Snort, así como el entorno de desarrollo y pruebas empleado durante la construcción del sistema. Se hace énfasis en la configuración de la Raspberry Pi como agente de seguridad, en los componentes de software integrados (Snort 3, complementos y herramientas auxiliares), y en cómo se llevó a cabo el desarrollo del frontend y backend y su validación en un entorno controlado.

### 2.3.1. Hardware: Raspberry Pi como agente de seguridad

El dispositivo hardware central de la solución es una **Raspberry Pi**, empleada aquí como un agente de seguridad en la red. En concreto, se utilizó una Raspberry Pi 5, que cuenta con un

procesador Quad-core Arm Cortex-A76 a 2,4 GHz y 8 GB LPDDR4X de memoria RAM, capacidades suficientes para ejecutar un IDS ligero junto con un servidor web básico. La Raspberry Pi se eligió por varias razones: es un equipo de bajo coste y tamaño reducido, con un consumo eléctrico mínimo, lo que permite tener un sensor de red siempre activo sin apenas impacto económico; además, su tamaño compacto facilita ubicarla directamente en el entorno de red a monitorizar (por ejemplo, junto al router o switch principal de la organización).

A pesar de sus recursos limitados, la Raspberry Pi ha demostrado ser capaz de albergar sistemas IDS/IPS funcionales en escenarios de pequeña escala. Por ejemplo, existen guías prácticas que muestran cómo convertir una Raspberry Pi en un IDS/IPS completamente funcional capaz de detectar intrusiones en la red. De hecho, la comunidad de seguridad ha validado la viabilidad de esta aproximación, aprovechando la portabilidad y suficiencia de procesamiento que ofrece este microordenador [23]. En nuestro caso, la Pi actúa como un *appliance* de seguridad dedicado: captura el tráfico de la red, ejecuta Snort para analizar dicho tráfico en busca de patrones maliciosos, y envía las alertas generadas a los subsistemas correspondientes (base de datos, interfaz web, dashboards, etc.).

La forma de **integración en la red** de la Raspberry Pi depende del modo de funcionamiento del IDS. En este proyecto, R-Snort se despliega en modo *pasivo* (IDS puro, no IPS), es decir, la Pi monitoriza el tráfico sin interrumpirlo. Para lograr esto, una práctica común es conectar la interfaz de red de la Raspberry Pi a un puerto espejo (*mirror/span port*) de un switch. De esta manera, la Pi recibe copias del tráfico de red pero no introduce latencia ni punto de falla en la comunicación. Alternativamente, si se buscara un modo *inline* (IPS), sería necesario contar con dos interfaces de red en la Raspberry Pi y configurarla como puente; sin embargo, para simplificar la implantación, R-Snort se centra en la monitorización pasiva.

En la Figura 2.2 se ilustra un posible **esquema de despliegue** de R-Snort en una red local. La Raspberry Pi está conectada al switch de la red mediante cable Ethernet; el switch está configurado para duplicar el tráfico de los dispositivos hacia el puerto donde se encuentra la Pi, permitiendo que Snort inspeccione los paquetes. El administrador puede acceder al frontend web y a los paneles de Grafana desde un equipo en la misma red (o remotamente vía VPN), conectándose a la dirección IP de la Raspberry Pi. Así, la Pi actúa como nodo de monitorización, analizando el tráfico y reportando alertas sin interferir en la comunicación normal de la red.

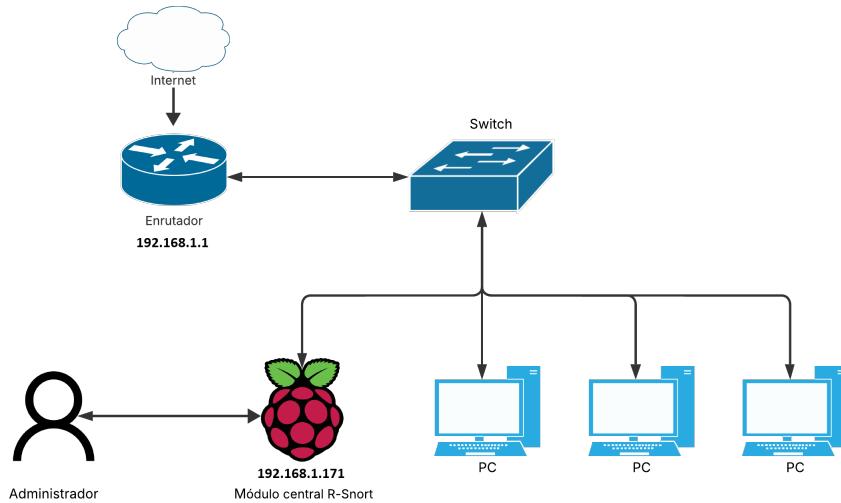


Figura 2.2: Esquema de despliegue de R-Snort en la red local.

En cuanto al **sistema operativo** utilizado en la Raspberry Pi, se optó por una distribución Linux de 64 bits para garantizar compatibilidad con Snort 3 y un buen aprovechamiento de la memoria. En particular, se empleó Ubuntu Server de 64 bits. Esta elección minimiza la carga de procesos innecesarios y permite dedicar la mayor parte de los recursos al IDS y al servidor web. Todas las interacciones con el sistema (instalación de paquetes, edición de configuraciones, revisión de logs) se realizaron vía acceso remoto SSH, ya que operaba en modo head-less (sin monitor ni periféricos).

Es importante resaltar que, aunque la Raspberry Pi 5 ofrece un rendimiento notable para su tamaño, sigue estando lejos del hardware de un servidor. Por ello, el volumen de tráfico que puede analizar en tiempo real es limitado. En entornos de alta carga (por ejemplo, redes superiores a 1 gigabit con tráfico sostenido muy elevado) es posible que la Pi se vea sobrepasada. Sin embargo, para los entornos objetivo de este proyecto —como redes domésticas o pequeñas oficinas—, con anchos de banda típicos de decenas de Mbps hasta 1 gigabit y tráfico intermitente, su capacidad es adecuada.

Las pruebas de rendimiento realizadas en el propio complemento de TFG han mostrado que, bajo condiciones de tráfico pasivo y también con cargas simuladas extremas, la Raspberry Pi 5 mantiene un uso controlado de CPU, RAM y disco, sin alcanzar límites de saturación que impidan el correcto funcionamiento. Por ejemplo, en la Figura 2.3 se observa que, durante el funcionamiento habitual del sistema, la CPU se mantiene en estado inactivo durante la mayor parte del tiempo. Bajo una situación de carga intensa, como se muestra en la Figura 2.4, se produce un aumento significativo del uso del procesador, especialmente en espacio de kernel, sin llegar a comprometer la estabilidad del sistema. A su vez, la Figura 2.5 demuestra que la carga media del sistema crece de forma sostenida pero controlada, lo que indica un reparto adecuado de los recursos sin cuellos de botella.

Todo esto respalda que el sistema puede operar de forma continua y estable dentro de sus márgenes. Estudios previos también han señalado que Snort (y herramientas similares como

Suricata) pueden alcanzar rendimientos de varios cientos de Mb/s en este tipo de dispositivos con configuraciones optimizadas [23].

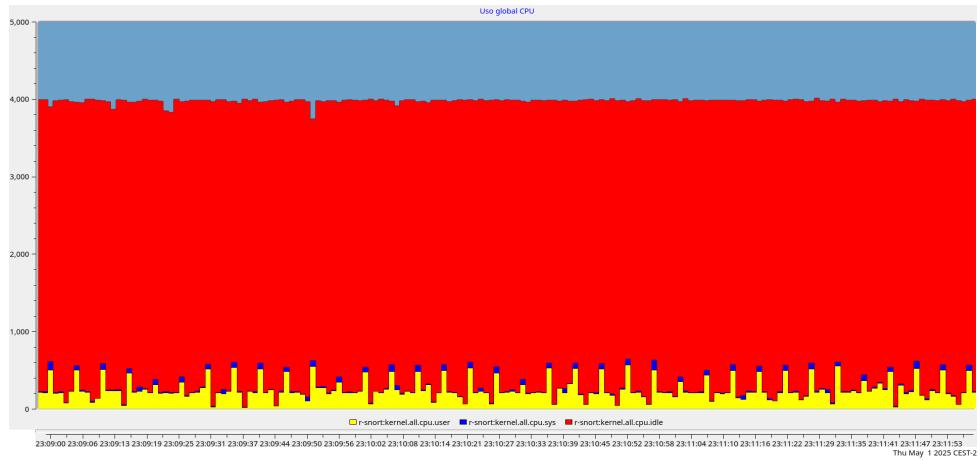


Figura 2.3: Uso de CPU de la Raspaberry Pi 5 bajo carga de trabajo habitual.

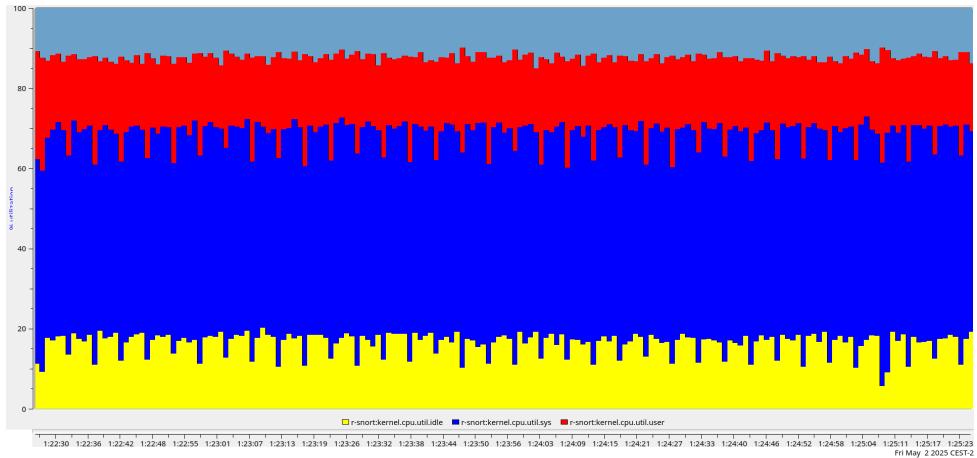


Figura 2.4: Uso de la CPU de la Raspaberry Pi 5 bajo carga de trabajo extremo.

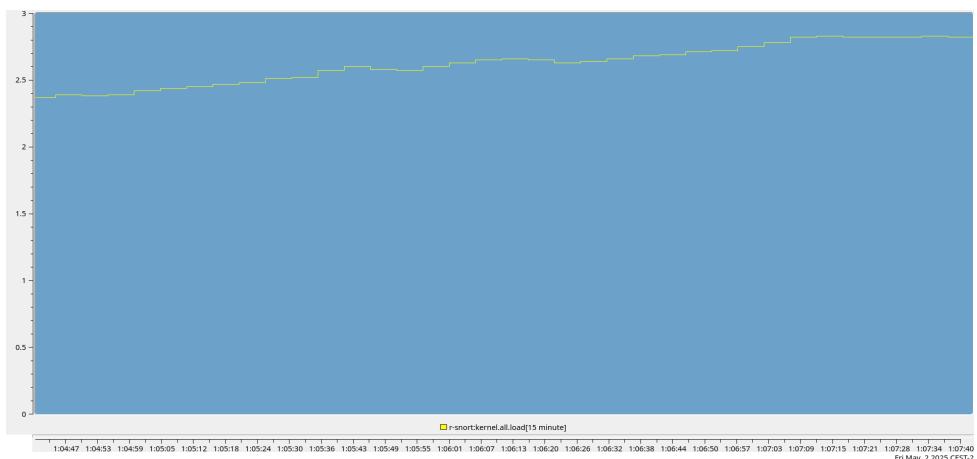


Figura 2.5: Carga del sistema de la Raspberry Pi 5 bajo en condiciones extremas.

### 2.3.2. Software: Snort 3 y complementos

A nivel de software, el componente principal es el **motor IDS Snort 3**. Snort es un sistema de detección de intrusiones de código abierto ampliamente utilizado, basado en análisis de firmas/patrones definidos en reglas. La versión 3 de Snort representa una reimplementación moderna del IDS clásico, aportando una serie de mejoras que resultan beneficiosas para este proyecto. Entre ellas cabe mencionar el soporte multi-hilo (permitiendo procesar paquetes en paralelo), un nuevo lenguaje de reglas más flexible, y la disponibilidad de más de 200 complementos o *plugins* para diversas funcionalidades. Todas estas mejoras conllevan una mayor eficiencia y facilidad de extensión del IDS, factores cruciales al desplegarlo en un entorno limitado pero que puede requerir personalizaciones.

En R-Snort, Snort 3 se configuró en **modo IDS** (no bloquea tráfico, solo alerta) y se ejecuta como servicio en la Raspberry Pi. Se personalizó el archivo de configuración de Snort (en formato Lua, propio de Snort3) para adaptarlo a la red local monitorizada: se definió la variable `HOME_NET` con el rango de IP interno a proteger, se ajustaron los puertos considerados para ciertos protocolos (por ejemplo incluir puertos de servicios comunes en la red local) y se activaron preprocesadores relevantes (como *Stream5* para seguimiento de sesiones TCP, *HTTP Inspect* para analizar tráfico web, etc.). Asimismo, se habilitó la salida de alertas en **formato JSON**, mediante el uso del *plugin* `alert_json`, de modo que cada evento de alerta que genera Snort se registra como una entrada JSON estructurada (conteniendo campos como `timestamp`, dirección IP de origen y destino, puerto, identificación de la regla que se disparó, mensaje de alerta, etc.). Este formato resulta conveniente para su posterior procesamiento automatizado por otras partes del sistema. A continuación un ejemplo de alerta:

```
1  {
2      "timestamp": "2025-05-25T18:30:12.123456+0200",
3      "flow_id": 123456789012345,
4      "in_iface": "eth0",
5      "event_type": "alert",
6      "src_ip": "192.168.1.10",
7      "src_port": 54321,
8      "dest_ip": "192.168.1.100",
9      "dest_port": 80,
10     "proto": "TCP",
11     "alert": {
12         "action": "allowed",
13         "gid": 1,
14         "signature_id": 1000001,
15         "rev": 1,
16         "signature": "Deteción de tráfico HTTP sospechoso",
17         "category": "Actividad sospechosa",
18         "severity": 2
19     }
20 }
```

Código 2.1: Ejemplo de alerta en formato JSON generada por Snort 3

Adicionalmente, se utilizaron ciertos **complementos de Snort** necesarios para su correcto funcionamiento en Raspberry Pi. En particular, se instaló la librería **DAQ** (*Data Acquisition*) apropiada para capturar el tráfico de red; en este caso, se usó `af_packet` (incluida con Snort

3) que permite leer directamente de la interfaz de red en modo promiscuo con buen rendimiento. También se configuró Snort para que se ejecute con privilegios limitados por seguridad (utilizando el usuario `snort` y grupo `snort` creados en el sistema) y se creó un servicio `systemd` para su lanzamiento automático en el arranque de la Pi.

Otro aspecto importante es la gestión de las **reglas IDS**. Para este proyecto, se decidió comenzar utilizando las *reglas comunitarias* gratuitas de Snort (Snort Community Rules) y algunas reglas emergentes de Emerging Threats, para disponer de un conjunto base que cubra amenazas comunes. Se implementó un script de actualización que, bajo demanda o de forma programada, descarga las últimas reglas (mediante *oinkcode* si aplica, en el caso de Snort) y actualiza el directorio de reglas de Snort. Así, desde la interfaz de R-Snort, el usuario podría desencadenar la actualización de reglas sin tener que hacerlo manualmente. Aunque herramientas clásicas como PulledPork aún no tienen versión oficial para Snort 3, se logró una funcionalidad similar mediante scripts en Bash que automatizan la descarga y colocación de nuevos archivos de reglas, seguidos de un reinicio suave de Snort para aplicar los cambios. Para más información sobre la sintaxis y estructura de las reglas en Snort 3, véase el Anexo III.

Además del propio Snort, el sistema incorpora varios **componentes software complementarios** que, en conjunto, constituyen la solución completa:

- **Backend central en Spring Boot.** desarrollado en Java 17 mediante el framework Spring Boot, este módulo no se instala en las Raspberry Pi (agentes), sino que reside en el servidor principal, actuando como centro de control y despliegue del sistema. Su función se limita a gestionar la autenticación, la gestión de usuarios y la coordinación de los distintos agentes de la red. Para ello, expone varios endpoints REST como `/api/login` (para iniciar sesión mediante credenciales y obtener un token JWT), `/api/agents` (para listar y registrar agentes disponibles). Un **agente** en este contexto es una Raspberry Pi con Snort y el software auxiliar instalado, dedicada a monitorizar el tráfico de red local de una ubicación concreta. Cada agente se comunica con el backend central para facilitar la administración remota y permitir que un único punto de control visualice, gestione y configure todos los nodos desplegados.
- **Backend local en cada agente (FastAPI).** cada Raspberry Pi funciona de manera autónoma como agente de monitorización, gracias a un backend ligero desarrollado en Python mediante FastAPI, denominado internamente `fasatpi`. Este componente se instala localmente en cada dispositivo o módulo de R-Snort y es el encargado de proporcionar la información propia del agente a la interfaz web. Expone endpoints REST como `/alerts`, `/status`, `/rules` o `/logs`, los cuales permiten consultar alertas generadas, comprobar el estado de Snort, gestionar reglas o descargar logs archivados. Internamente, `fasatpi` accede a los archivos JSON generados por Snort, así como a la base de datos local MariaDB donde se almacenan las alertas y métricas. Además, implementa acciones sobre el sistema, como reiniciar Snort mediante scripts o comprobar el estado de servicios mediante comandos shell. Gracias a este diseño, cada agente puede funcionar de forma autónoma incluso sin conexión constante al backend central, y ofrece una interfaz clara para que R-Snort WebApp pueda consultar y controlar su estado desde la red.

- **Base de datos MariaDB.** para persistir las alertas y otros datos, se configuró un servidor de base de datos MariaDB 10.x corriendo en la propia Raspberry Pi. Cada alerta generada por Snort es insertada en una tabla (p. ej. `alerts`) con campos como timestamp, IP origen, IP destino, puerto, protocolo, SID de la regla, mensaje, etc. Esta inserción se realiza de manera automática: el backend Spring Boot incluye un componente que supervisa la aparición de nuevas entradas en el log JSON de Snort (mediante un hilo que observa el archivo o utilizando mecanismos de notificación del sistema de ficheros) y en cuanto detecta una nueva alerta, la parsea del JSON e inserta los datos en la base MariaDB mediante JDBC. De esta forma, construimos una **pipeline** de datos desde Snort hacia la base de datos casi en tiempo real. Cabe mencionar que Grafana y el propio frontend consultarán esta base para mostrar la información al usuario.
- **Grafana.** es la herramienta seleccionada para la visualización gráfica de los datos. Grafana es una aplicación web de monitorización que permite crear paneles interactivos a partir de diversas fuentes de datos (tiempo real, series temporales, etc.) [24]. En R-Snort, se instaló Grafana OSS (código abierto) versión 12 en la Raspberry Pi, aprovechando que existen builds para arquitectura ARM. La instancia de Grafana se configuró para conectarse a la base de datos MariaDB como *data source* utilizando el conector MySQL. De esta forma, se pudieron escribir consultas SQL sobre la tabla de alertas y representar resultados en gráficas. Grafana se ejecuta como un servicio aparte, corriendo en el puerto 3000 de la Raspberry Pi, y la interfaz Angular tiene la opción de embeber ciertos paneles de Grafana mediante iframes o bien el usuario puede acceder directamente al dashboard completo de Grafana para análisis más avanzados. Ambos (frontend propio y Grafana) beben de la misma base de datos de alertas, garantizando consistencia en la información.
- **Rotación de logs y mantenimiento.** a nivel de sistema operativo, se añadieron configuraciones para mantener el sistema funcionando sin intervención manual. En particular, se creó una tarea cron diaria que ejecuta `logrotate` sobre los archivos de log de Snort (y opcionalmente logs del backend), de modo que cada día se archiva el log del día anterior y se comienza uno nuevo, conservando, por ejemplo, solo 7 días de logs comprimidos. Asimismo, se desarrollaron pequeños scripts de *health-check* que comprueban si el proceso de Snort sigue activo y, si no, intentan reiniciarlo usando `systemctl`; esto agrega robustez frente a eventuales caídas del IDS. El backend Spring Boot se configuró también como servicio de sistema para iniciarse automáticamente en el arranque (`systemd service`) y reiniciarse en caso de falla.

En resumen, la Raspberry Pi ejecuta de forma concurrente varios servicios: Snort 3 (detección de intrusos), Spring Boot (API REST y lógica de negocio), FAST API (datos del sistema), MariaDB (almacenamiento de datos) y Grafana (visualización). Gracias a la optimización y a la moderada carga esperada, la Raspberry Pi es capaz de manejar estos componentes simultáneamente. Cada uno cumple un rol específico pero se complementan para brindar la funcionalidad global de R-Snort. El diseño modular permite incluso que, de ser necesario, alguno de estos componentes pudiera desplazarse a otro equipo (por ejemplo, usar un servidor externo para Grafana o la base de datos) sin grandes cambios, evidenciando la flexibilidad de la arquitectura planteada.

### 2.3.3. Entorno de desarrollo

El desarrollo del frontend y backend de R-Snort se llevó a cabo utilizando herramientas modernas y siguiendo buenas prácticas para garantizar la calidad del software. Para la parte de **frontend Angular**, se empleó **TypeScript** como lenguaje de programación y el *framework* Angular en su versión 19. La estructura del proyecto Angular se inicializó con `angular-cli`, organizando los componentes, servicios y rutas de la aplicación de forma modular. Durante la implementación, se hizo uso de un conjunto de librerías de interfaz comunes (por ejemplo, Angular Material) para agilizar la construcción de elementos UI como tablas, diálogos modales y gráficos simples. El entorno de desarrollo incluyó herramientas como **Visual Studio Code** como editor principal y el servidor de desarrollo de Angular (`ng serve`) que permitía ver los cambios en vivo en el navegador durante la codificación. Para el manejo de versiones del código se utilizó Git, lo cual facilitó llevar un control de cambios y colaborar en caso de integración de diferentes partes.

En cuanto al **backend Spring Boot**, se utilizó el *stack* típico de Spring: Spring MVC para la creación de controladores REST, Spring Data JPA para la capa de persistencia con MariaDB (definiendo entidades JPA que representan las alertas, etc.), y Spring Security para implementar la capa de autenticación JWT. El proyecto se configuró con Maven como gestor de dependencias. Durante el desarrollo, se probaban los endpoints localmente en un equipo de escritorio antes de desplegarlos en la Raspberry Pi, usando herramientas como **Postman** para verificar las respuestas JSON de la API. Una vez validados, el backend se compilaba en un *JAR ejecutable* (`spring-boot fat jar`) y se transfería a la Raspberry Pi para su ejecución en producción. El sistema cuenta con OpenJDK 17 instalado para correr la aplicación Java.

Para asegurar la **integración del frontend con el backend**, se habilitó **CORS** (Cross-Origin Resource Sharing), ya que durante la etapa de desarrollo el frontend Angular se ejecutaba en `localhost:4200` y necesitaba consumir la API alojada en la Raspberry Pi. Spring Boot se configuró para permitir el origen del frontend en desarrollo. En la versión final, al compilar la aplicación Angular para producción, los archivos estáticos (HTML/JS/CSS) del frontend se sirvieron directamente desde el backend (Spring Boot puede servir contenido estático), evitando así problemas de CORS en el uso real.

El entorno de desarrollo se aseguró que todos los componentes de R-Snort funcionaran de manera armónica. La elección de Angular y Spring Boot simplificó la construcción del frontend manteniendo un código mantenible.

## 2.4. Diseño del frontend

### 2.4.1. Diseño de la arquitectura

El sistema **R-Snort** presenta una arquitectura distribuida y modular, compuesta por múltiples **agentes** desplegados junto a instancias de Snort y un **módulo central** que coordina el funcionamiento general.

### Agentes Snort distribuidos (**snort-agent**)

Cada snort-agent es un componente ligero (desarrollado en Python) que se instala en los equipos a monitorizar junto al software Snort. La instalación del agente es sencilla y automatizada, mediante un script o paquete que despliega el propio agente y todas sus dependencias (incluyendo la base de datos local MariaDB) sin necesidad de configurar manualmente cada detalle. Una vez instalado, el agente arranca varios servicios internos que gestionan la detección de intrusiones de forma autónoma:

- `ingest_service.py`: servicio responsable de leer las alertas generadas por Snort e insertarlas en la base de datos MariaDB local del agente (tabla de alertas).
- `metrics_timer.py`: proceso en segundo plano que periódicamente recoge métricas del sistema anfitrión (uso de CPU, uso de disco, temperatura de la CPU, etc.) y las almacena en la base de datos local (tabla de métricas del sistema). Esto permite tener un registro histórico del rendimiento y estado de cada sensor.
- `agent_api.py`: servidor web ligero (implementado con *FastAPI*) que expone una API REST para interactuar con el agente. Este componente proporciona endpoints para consultar el estado (/status) del sensor y de Snort, obtener listas de alertas (/alerts) y métricas (/metrics), gestionar las reglas de detección (GET /rules para listar reglas personalizadas, POST /rules para agregarlas, DELETE /rules/sid para eliminarlas) y controlar servicios (/services/status y /services/restart para verificar o reiniciar Snort, entre otros). La API actúa de intermediario entre el nodo sensor y el resto del sistema, permitiendo que el módulo central recupere datos y emitan órdenes de forma estandarizada.

El agente también configura automáticamente el dashboard Grafana, utilizando una plantilla adaptada del dashboard *Snort IDS* disponible originalmente en Grafana Labs [26]. Aunque inicialmente diseñada para InfluxDB, esta plantilla se ha modificado específicamente para funcionar con MariaDB, utilizando la estructura actual de datos del sistema R-Snort recibiendo y procesando datos del agente en cuestión para mostrar diversos paneles permitiendo visualizar claramente el estado y alertas del sistema filtrando por diversos protocolos, IPs, severidad entre otros.

Los snort-agent automatizan gran parte de la configuración y mantenimiento del sistema en cada nodo. Por ejemplo, se encargan de aplicar la configuración de Snort para recoger las alertas en un determinado fichero y ajustan componentes complementarios del entorno. Incluyen mecanismos para respaldo de datos, copias de seguridad periódicas de logs y gestionan la rotación de estos (*logrotate*) para evitar el crecimiento indefinido de los archivos de alertas. Todo ello reduce la intervención manual y facilita un enfoque *plug-and-play*, donde un usuario no técnico puede desplegar un sensor Snort de forma sencilla.

### Módulo central (coordinador + interfaz web)

El módulo central actúa tanto como un agente estándar como un nodo coordinador que aloja el backend en Spring Boot y el frontend Angular. El backend proporciona autenticación mediante

JWT y expone servicios REST que coordinan la comunicación con los diferentes agentes, permitiendo consultas unificadas desde la interfaz web. Así, todas las peticiones del usuario (obtención de alertas, modificación de reglas, reinicio de Snort) son redirigidas desde el backend hacia las APIs REST de cada agente remoto, garantizando una comunicación distribuida, estandarizada y eficiente.

El **módulo central** del sistema actúa a la vez como un agente y como el nodo coordinador que concentra la interfaz de usuario y la lógica de negocio. En el servidor central se ejecuta también un *snort-agent* (monitorizando el propio equipo central como un sensor más), pero adicionalmente aloja un backend desarrollado en Spring Boot y el frontend web en Angular. El backend Spring Boot expone una API REST (con autenticación JWT) que pone en marcha la comunicación entre el frontend y los distintos agentes distribuidos. A través de esta capa central, los usuarios pueden autenticarse y realizar consultas unificadas: por ejemplo, solicitar las alertas de un cierto agente, agregar una nueva regla de detección o verificar el estado de todos los sensores. El backend traduce esas solicitudes en llamadas a la API REST de los agentes correspondientes. De este modo, si el usuario solicita datos de un agente remoto, el servidor central invoca el endpoint apropiado del `agent_api.py` en ese nodo (por ejemplo `/alerts` para obtener las alertas recientes desde la base de datos local del agente). Del mismo modo, cuando el usuario emite una acción (p. ej. reiniciar Snort en un sensor), el backend central envía la orden mediante `/services/restart` al agente seleccionado. Toda la transferencia de datos se realiza en formato JSON sobre HTTP(S), manteniendo un bajo acoplamiento entre módulos y permitiendo escalar o reemplazar componentes fácilmente.

En la Figura 2.6 se muestra un esquema general de la arquitectura de R-Snort, donde se aprecian sus elementos principales y las interacciones entre ellos: el usuario accede mediante un navegador al frontend Angular (alojado en el servidor central), el cual se comunica con el backend Spring Boot. El backend a su vez envía peticiones REST hacia los *snort-agent* distribuidos (incluido el agente del propio servidor central) para recopilar información o ejecutar comandos. Esta arquitectura distribuida y modular permite monitorizar múltiples puntos de la red de forma centralizada y coherente.

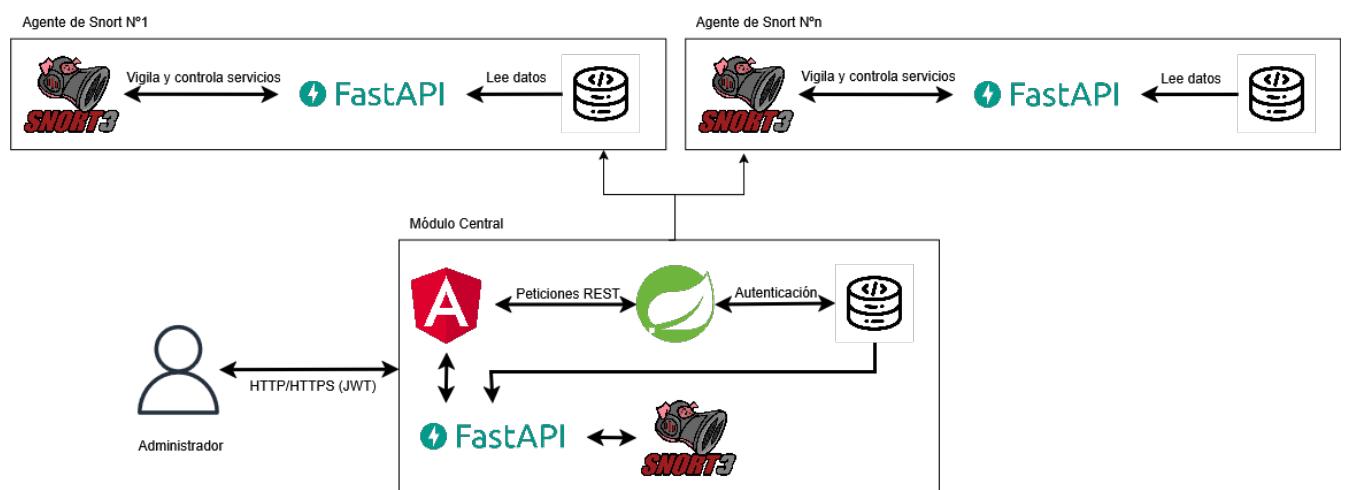


Figura 2.6: Esquema de la arquitectura distribuida de R-Snort.

El diseño plug-and-play de R-Snort, aunque orientado para administradores de sistemas y seguridad, se ha creado para que cualquiera con conocimientos mínimos de Linux sea capaz de instalarlo: agregar un nuevo sensor es tan simple como instalar un agente en el nuevo equipo e informarlo al sistema central, sin configuraciones manuales complejas. Esta aproximación aporta sencillez en entornos de ciberseguridad distribuida, ya que ofrece visibilidad unificada de múltiples sensores de intrusión. Cada agente opera de manera autónoma en su segmento de red, pero todos son gestionados centralmente, lo que aumenta la cobertura de detección y la resiliencia (si un sensor falla o es comprometido, los demás continúan operativos y reportando al panel central). Para una descripción detallada de este proceso de instalación y configuración de agentes, puede consultarse el Anexo II.

#### 2.4.2. Diseño de la interfaz de usuario

##### Frontend web (Angular SPA)

El sistema cuenta con una interfaz web de usuario desarrollada en Angular, construida como una *Single Page Application (SPA)* independiente que se comunica con el backend mediante peticiones HTTP a la API REST central. Esta aplicación frontend, desplegada en el servidor central, ofrece una experiencia interactiva y dinámica, con un diseño visual enfocado en el dominio de la ciberseguridad. La usabilidad se ha cuidado para que la información compleja de las alertas y métricas se presente de forma accesible y atractiva incluso para operadores no especializados.

La interfaz de usuario se estructura en varios módulos o pantallas principales, accesibles desde una barra de navegación superior:

- **Inicio de sesión.** Es la pantalla inicial de la aplicación, donde el usuario introduce sus credenciales (*correo electrónico* y *contraseña*) para acceder. Tras hacer clic en *Acceder*, el backend valida las credenciales y, de ser correctas, emite un token de autenticación JWT. La apariencia de esta vista es sencilla y centrada, siguiendo las pautas de diseño minimalista.
- **Selector de agente.** Una vez autenticado, el usuario puede seleccionar cuál de los agentes disponibles desea monitorizar. En la barra superior de la SPA se muestra un menú desplegable que lista los agentes registrados identificados por nombre o dirección IP, e.g. Central (192.168.1.171). Al elegir un agente, el frontend carga los datos correspondientes a ese sensor en las vistas de alertas, reglas y estado. Este mecanismo permite controlar múltiples sensores desde la misma interfaz de forma conveniente.
- **Vista de alertas.** Es el panel principal de monitorización, donde se presentan las detecciones de intrusiones reportadas por Snort en el agente seleccionado. Incluye indicadores numéricos de alertas por nivel de severidad (ej. cantidad de alertas altas, medias, bajas) y una tabla que enumera las últimas alertas detalladas con sus campos clave (hora, descripción de la alerta, IP de origen, IP de destino, puerto implicado, clasificación de la alerta y nivel de severidad). Esta vista permite al analista ver rápidamente qué incidentes se han detectado y su naturaleza. Además, proporciona botones para exportar los datos de alertas en formato CSV: por ejemplo, descargar alertas de este agente genera un archivo con todas

las alertas actuales del sensor activo, mientras que "Descargar todas las alertas" consolida las alertas de todos los agentes en un solo fichero (facilitando un análisis global).

- **Vista de reglas.** Sección de la aplicación dedicada a la gestión de reglas personalizadas de Snort en el agente seleccionado. Permite al usuario con rol adecuado agregar nuevas reglas de detección escribiendo la definición en formato Snort en un campo de texto y pulsando ".Añadir regla". Una vez enviada, la regla se almacena a través del backend en el agente correspondiente (el snort-agent la añade a la configuración de Snort y recarga el motor de detección para que la nueva regla entre en vigor). Asimismo, en esta vista se listan todas las reglas personalizadas previamente añadidas al sensor, mostrando por cada una su SID (identificador único), el mensaje descriptivo (MSG) y la sintaxis completa de la regla. Junto a cada regla se presenta una opción (ícono de papelera) para eliminarla si es necesario, lo cual provoca que el agente la remueva de Snort (actualizando la configuración).
- **Vista de estado del sistema.** Es la interfaz orientada a visualizar las métricas operativas del agente y el estado del servicio Snort. Contiene gráficas en tiempo real de los datos del sistema recabados (por ejemplo, un gráfico de línea de la temperatura de CPU del sensor, y otro del porcentaje de uso de CPU y de disco a lo largo del tiempo). Estos gráficos ofrecen al usuario una idea del rendimiento y condiciones del hardware donde corre Snort, ayudando a detectar posibles sobrecargas o problemas (p.ej., temperatura anómala). Además, esta vista muestra el estado actual de Snort en ese agente: indica si el proceso IDS está *Activo* y la marca de tiempo de la última alerta registrada. Si el usuario cuenta con permisos de administrador, desde aquí puede reiniciar remotamente el servicio Snort en el agente seleccionado mediante un botón dedicado ('Reiniciar Snort'). También se listan en esta página los *logs* archivados (ficheros históricos de alertas comprimidos tras la rotación de logs); el usuario puede descargarlos para un análisis forense si es necesario.
- **Overview e integración con Grafana.** La página inicial del panel (denominada *Overview*) ofrece un resumen gráfico consolidado de la información de seguridad. Esta vista integra paneles de visualización tipo *dashboard* para representar las alertas y eventos de forma gráfica y amigable, aprovechando herramientas como Grafana para las métricas de serie temporal. Por ejemplo, se muestran diagramas de dona con la distribución de alertas por severidad o por tipo, gráficas de líneas con el volumen de eventos en el tiempo y otros gráficos interactivos que ayudan a identificar patrones de ataque de un vistazo. La integración con Grafana permite reutilizar su potente motor de gráficos para presentar datos de los agentes dentro de la propia interfaz de R-Snort, manteniendo la consistencia del estilo visual oscuro y profesional.

### Mecanismo de autenticación y control de acceso

El frontend utiliza autenticación basada en **JWT**: tras iniciar sesión, el token JWT proporcionado por el backend se adjunta a cada petición subsiguiente, garantizando que solo usuarios autenticados (y con rol adecuado) accedan a los recursos. Asimismo, la aplicación cliente refleja en todo momento el estado de los agentes: se realiza una comprobación periódica de disponibilidad de cada sensor, ya sea mediante *ping* de red o invocando el endpoint /status de su API. Si un agente no responde (caído o desconectado), el panel lo indicará claramente (por ejemplo,

marcándolo como inactivo o mostrando una alerta visual), permitiendo al usuario identificar problemas de conectividad o fallos en los sensores de forma proactiva.

#### 2.4.3. Esquema de la base de datos

Para almacenar la información de alertas de intrusión, métricas del sistema y credenciales de usuarios, R-Snort emplea una base de datos relacional MariaDB. El esquema se ha mantenido sencillo, definiendo tres tablas principales: `alerts`, `system_metrics` y `users`. A continuación se presenta la estructura SQL de cada tabla junto con una breve descripción de su contenido y función.

La tabla `alerts` almacena cada evento de alerta generado por Snort en los agentes. Cada fila corresponde a una alerta detectada e incluye los detalles más relevantes: fecha y hora del evento, dirección IP de origen, dirección IP de destino, puerto implicado, descripción o mensaje de la alerta, clasificación asignada por Snort (p.ej. tipo de ataque o categoría) y nivel de severidad (prioridad). Esta tabla es alimentada en tiempo real por el componente `ingest_service.py` de cada agente, que inserta un nuevo registro cada vez que Snort produce una alerta. De este modo, `alerts` actúa como el registro histórico de incidentes detectados por cada sensor.

alerts	
<code>id</code> ↗	int
<code>timestamp</code>	datetime
<code>alert_msg</code>	varchar(255)
<code>source_ip</code>	varchar(45)
<code>dest_ip</code>	varchar(45)
<code>dest_port</code>	int
<code>classification</code>	varchar(100)
<code>severity</code>	varchar(10)

Figura 2.7: Tabla `alerts`.

Por su parte, la tabla `system_metrics` guarda las mediciones periódicas del estado de cada equipo monitorizado. Cada registro representa una muestra tomada por `metrics_timer.py` e incluye la marca temporal del muestreo junto con los valores de parámetros como porcentaje de CPU en uso, porcentaje de espacio de disco utilizado y temperatura de la CPU en grados Celsius, entre otros posibles indicadores del rendimiento del sistema. La frecuencia de inserción en esta tabla depende de la configuración del agente (por ejemplo, cada minuto o cada pocos minutos se añade un nuevo registro). Gracias a estos datos, el administrador puede consultar históricos de rendimiento y detectar condiciones anómalas (como un sobrecalentamiento sostenido o falta de recursos) a través de las gráficas en la interfaz.

system_metrics	
<b>id</b> ↗	int
timestamp	datetime
cpu_usage	decimal(5,2)
disk_usage	decimal(5,2)
cpu_temp	decimal(5,2)

Figura 2.8: Tabla métricas.

Finalmente, la tabla `users` contiene las cuentas de usuario autorizadas para ingresar al panel de control de R-Snort. Por motivos de seguridad, en esta tabla se almacenan las credenciales de forma cifrada (la contraseña se guarda con hash). Los campos principales incluyen el correo electrónico del usuario (que funciona como nombre de usuario para el inicio de sesión), la contraseña hasheada. Esta información es gestionada por el backend Spring Boot: durante el proceso de autenticación se verifica la combinación de email y contraseña contra los registros de `users` y, si coincide, se genera el JWT correspondiente.

users	
<b>id</b> ↗	int
email	varchar(100)
password	varchar(255)

Figura 2.9: Tabla usuario.

Con estas tres tablas, el sistema almacena de forma organizada tanto los eventos de seguridad recopilados por los sensores (`alerts`), como los datos de rendimiento de los mismos (`system_metrics`) y la información de quienes administran o supervisan la plataforma (`users`). El diseño relacional simplificado facilita las consultas necesarias: el backend puede obtener rápidamente las alertas filtrando por fecha, severidad u otros campos; extraer series de métricas para graficar en el dashboard; y verificar credenciales de inicio de sesión con consultas eficientes. Además, al mantener la base de datos en MariaDB (un sistema ampliamente soportado), se garantiza confiabilidad, rendimiento suficiente para los volúmenes de datos manejados, y la posibilidad de escalar o realizar copias de seguridad de la información de manera estándar.

alerts		system_metrics		users	
<b>id</b>	int	<b>id</b>	int	<b>id</b>	int
timestamp	datetime	timestamp	datetime	email	varchar(100)
alert_msg	varchar(255)	cpu_usage	decimal(5,2)	password	varchar(255)
source_ip	varchar(45)	disk_usage	decimal(5,2)		
dest_ip	varchar(45)	cpu_temp	decimal(5,2)		
dest_port	int				
classification	varchar(100)				
severity	varchar(10)				

Figura 2.10: Tablas base de datos.

## 2.5. Implementación y configuración del sistema

Partiendo de una instalación estándar de Snort 3, se llevó a cabo una serie de configuraciones y desarrollos adicionales para convertirlo en un agente IDS autónomo, monitorizable y funcional (denominado snort-agent). En esta sección se detallan estos pasos, desde la habilitación de un registro estructurado de alertas en formato JSON hasta la integración de un panel de control en Grafana respaldado por MariaDB. El resultado es un agente capaz de generar alertas procesables, registrar métricas operativas y exponer su estado a sistemas externos, sentando las bases para su posterior automatización.

### 2.5.1. Snort 3: configuración de salida de alertas en JSON

El primer paso fue modificar la configuración por defecto de Snort 3 para que las alertas se registraran en formato JSON, lo que facilita su posterior tratamiento automático. Snort 3 tiene un *plugin* de *logging* llamado `alert_json` que permite volcar cada alerta generada como una entrada JSON estructurada. Para habilitarlo, se editó el fichero de configuración principal (por defecto, `snort.lua`) en la sección de outputs, añadiendo un bloque de configuración específico para `alert_json`. En el Código 2.2 se muestra un extracto resumido de esta configuración:

```
1  alert_json =
2  {
3      file = true, -- habilita salida a archivo (no solo consola)
4      limit = 50, -- tamaño (MB) tras el cual se rota el archivo
5      fields = 'timestamp src_addr src_port dst_addr dst_port proto msg sid priority
6          ↪ classtype ...'
```

Código 2.2: Configuración de Snort 3 para salida de alertas en JSON

En esta configuración, se indica a Snort que vuelve las alertas en un archivo (en lugar de la salida estándar de consola) y se establece un límite de tamaño (`limit`) para que, al alcanzar 50MB, Snort rote automáticamente al siguiente archivo de alerta (evitando que un único fichero crezca indefinidamente). Además, mediante la opción `fields`, se especifica el conjunto de campos de cada alerta que serán incluidos en el JSON. En nuestro caso se optó por incluir todos los campos relevantes: marca de tiempo, direcciones IP origen y destino, puertos, protocolo, mensaje/descripción de la alerta, identificadores de regla (`sid/rev`), clasificación (`classtype`) y severidad (`priority`), entre otros. Cabe mencionar que Snort 3 añade por defecto un campo numérico de tiempo (`seconds`) al inicio del JSON para asegurar un ordenado procesado temporal. Tras habilitar `alert_json`, Snort deja de imprimir alertas por pantalla y las escribe en el archivo designado (`/var/log/snort/alert_json.txt`), listo para ser procesado por otros componentes del sistema.

Finalmente, para ejecutar Snort de forma persistente como agente en segundo plano, se configuró un servicio del sistema (`systemd`). Se creó un fichero de unidad `/etc/systemd/system/snort.service` que inicia Snort con unos determinados parámetros: la interfaz de red a monitorizar, el fichero de configuración `snort.lua` y el directorio de logs. De este modo, el IDS se inicia automáticamente al encender el sistema y permanece activo como *daemon*, escribiendo sus alertas en formato JSON en el archivo configurado.

### Servicio de almacenamiento de alertas en base de datos

Con las alertas de Snort ya estructuradas en JSON, se implementó un servicio auxiliar de **almacenamiento** cuyo cometido es leer continuamente el archivo de alertas JSON y volcar la información a una base de datos MariaDB. Este servicio actúa como puente entre Snort y el sistema de monitorización, extrayendo los datos de cada alerta y almacenándolos en las tablas correspondientes donde pueden ser consultados fácilmente por otras herramientas (p. ej., Grafana).

Para su implementación se desarrolló un servicio (en este caso, un script en Python) que se lanza junto con Snort. Dicho proceso abre el archivo de alertas (`alert\_json.txt`) en modo lectura continua y espera nuevos eventos. Cada vez que Snort escribe una nueva línea JSON, el servicio de almacenamiento la detecta, la parsea (utilizando una biblioteca JSON estándar) y extrae los campos de interés. A continuación, construye una sentencia `INSERT` para añadir la alerta a la base de datos. En la tabla `alertas` de MariaDB, cada registro almacena los campos principales de una alerta: marca de tiempo, IP origen, IP destino, puertos, protocolo, severidad, clasificación y mensaje de la alerta, entre otros. De este modo, cada alerta queda persistida. El proceso se asegura de manejar correctamente el flujo continuo de alertas (por ejemplo, manteniendo el fichero abierto y leyendo línea a línea conforme crece, similar al comando `tail -f`). Asimismo, ante una rotación del archivo de log (discutida más adelante), el servicio de almacenamiento está preparado para cerrar y reabrir el nuevo fichero sin interrumpir la captura de eventos.

Este enfoque aporta varias ventajas. Primero, las consultas SQL permiten filtrar y agregar alertas de forma flexible (por ejemplo, obtener conteos por tipo de alerta o por IP origen). Segundo, MariaDB facilita la gestión del histórico de alertas, a la vez que permite acceso concurrente desde múltiples aplicaciones (por ejemplo, la API del agente o el dashboard). Por último, se añadió a cada inserción un identificador del agente (un ID único para cada despliegue de R-Snort) para posibilitar entornos con múltiples sensores: de esta forma, una instancia centralizada de la base de datos puede distinguir el origen de las alertas y consolidar la información de varios agentes.

El Código 2.3 muestra la lógica descrita: el servicio se conecta a MariaDB, abre el fichero de alertas y entra en un bucle infinito leyendo nuevas líneas JSON según aparecen. Cada alerta se convierte de JSON a estructuras de datos (`alert`) y se inserta en la tabla `alertas` mediante una consulta SQL parametrizada. Nótese que en una implementación real se incluyen consideraciones adicionales (manejo de errores de conexión, reconexión en caso de reinicio de la base de datos, etc.), pero a grandes rasgos este mecanismo garantiza que todas las alertas generadas por Snort queden almacenadas en la base de datos en tiempo real.

Este flujo puede visualizarse también a nivel de proceso en la Figura 2.11, donde se esquematiza el comportamiento del servicio de almacenamiento desde su activación hasta el tratamiento de nuevas entradas, incluyendo la gestión de rotación de logs.

```

1 conn = mysql.connector.connect(user="snort", password="...", database="snortDB") #
2     ↳ Implementación real con variables de entorno obtenidas de un fichero de
3     ↳ configuración
4 cursor = conn.cursor()
5 f = open("/var/log/snort/alert_json.txt", "r")
6 f.seek(0, 2) # Ir al final del archivo
7 while True:
8     line = f.readline()
9     if not line:
10        time.sleep(1); continue # Esperar datos nuevos
11        alert = json.loads(line)
12        cursor.execute(
13            "INSERT INTO alertas(timestamp, src_ip, dst_ip, proto, puerto, severidad, clase,
14             ↳ mensaje, agente_id) "
15            "VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)",
16            (alert["timestamp"], alert["src_addr"], alert["dst_addr"], alert["proto"],
17             alert["dst_port"], alert["priority"], alert["class"], alert["msg"], AGENTE_ID))
18        conn.commit()

```

Código 2.3: Esquema simplificado del servicio de almacenamiento de alertas

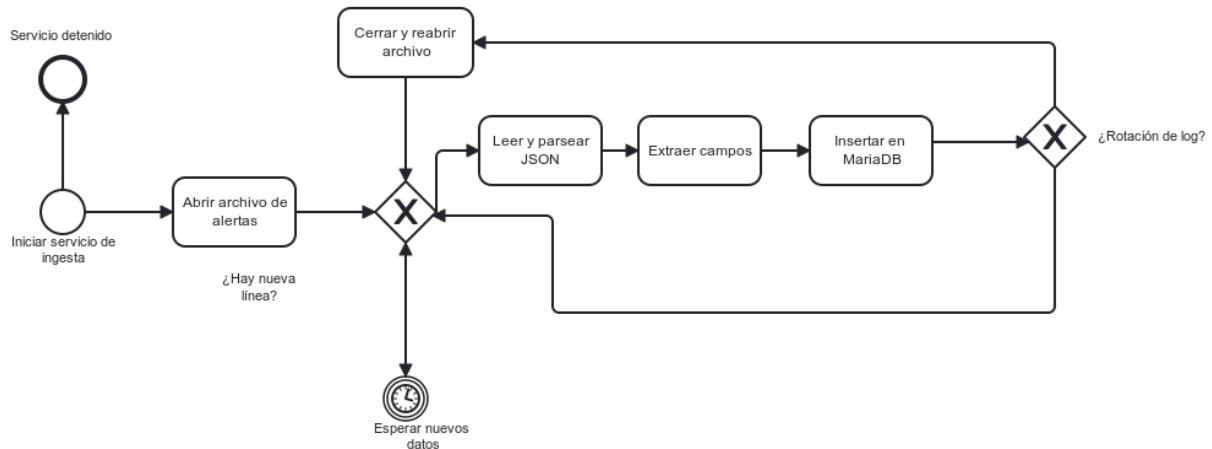


Figura 2.11: BPMN del servicio de almacenamiento de alertas.

### Servicio de métricas y monitorización del agente

Además de las alertas, el agente R-Snort recopila y expone métricas operativas que permiten monitorizar su estado y rendimiento. Para ello se desplegó un segundo servicio auxiliar encargado de recolectar periódicamente datos como el uso de CPU y memoria del proceso Snort, el estado de sus interfaces de captura, etc. Algunas de estas métricas se obtienen directamente de la base de datos de alertas (por ejemplo, un conteo de alertas por severidad en la última hora), mientras que otras provienen del propio sistema operativo (por ejemplo, consultando /proc para recursos utilizados por Snort).

Este servicio de métricas corre en segundo plano y vuelca los datos recopilados también en la base de datos MariaDB, en tablas diseñadas para métricas. Por ejemplo, se define una tabla `metricas` donde cada registro puede contener el timestamp, el nombre de la métrica y el valor observado en ese momento (p.ej., `cpu_usage = 30 %`). De esta forma, se va construyendo

un historial temporal no solo de eventos de seguridad (alertas) sino también de indicadores de rendimiento del sensor.

Al igual que el servicio de almacenamiento, el servicio de métricas está pensado para ejecutarse automáticamente al iniciar el agente. Se ha optado por integrarlo en el propio agente para simplificar la coordinación. Este componente rellena las tablas de métricas en MariaDB y también expone la información importante a través de la API REST del agente (por ejemplo, un endpoint GET /metrics devuelve las últimas métricas recopiladas, facilitando integraciones externas).

El comportamiento descrito se refleja en la Figura 2.12, que muestra cómo se recopilan, procesan y almacenan periódicamente las métricas del sistema, incluyendo la gestión de errores y la actualización del endpoint REST correspondiente.

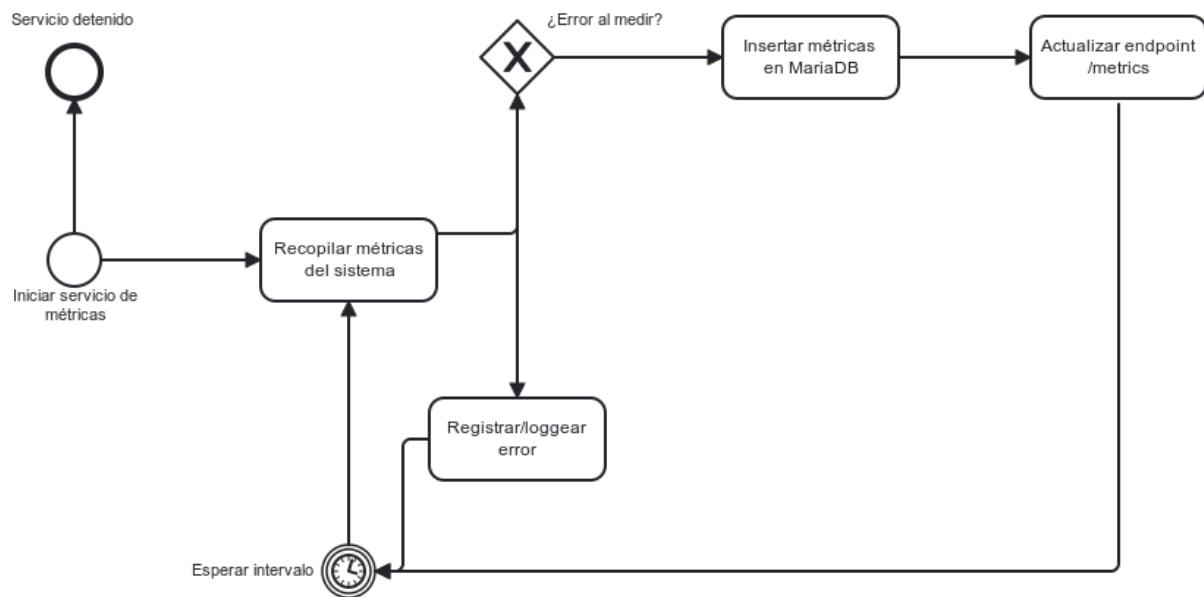


Figura 2.12: BPMN del servicio de almacenamiento de métricas.

### Rotación y archivado de logs de Snort

Dado el carácter continuo de la monitorización, es de vital importancia gestionar los ficheros de registro que Snort genera, evitando tanto la pérdida de información como el consumo ilimitado de espacio en disco. Para ello se implementó una estrategia combinada de rotación y archivado de logs.

Por un lado, como ya se indicó, Snort 3 rota internamente el archivo de alertas JSON cuando este alcanza un tamaño predefinido (limit en la configuración). En el despliegue, se eligió un límite (p.ej., 50MB) de modo que, al superarse, Snort cierre `alert_json.txt` y comience un nuevo archivo (generalmente nombrado con un sufijo temporal, por ejemplo `alert_json.txt.1684500000` para garantizar un nombre único). Esto asegura que el archivo activo de alertas se mantenga de un tamaño manejable para el servicio de almacenamiento y que las alertas antiguas queden preservadas en ficheros aparte.

Por otro lado, para gestionar esos ficheros históricos y evitar la acumulación indefinida, se configuró la herramienta **logrotate** en el sistema. Se añadió una regla específica para los logs de Snort en `/etc/logrotate.d/`, de forma que diariamente (o bajo ciertas condiciones de tamaño) se compriman y archiven los archivos de alerta rotados. La configuración de *logrotate* establece, por ejemplo, mantener solo los N archivos más recientes (p.ej., los últimos 7 días de alertas), comprimir automáticamente los logs rotados (formato `.gz`) y eliminar aquellos que excedan el período de retención.

En el ejemplo del Código 2.4, *logrotate* rotará cualquier fichero que comience con `alert_json.txt` (incluyendo los generados por Snort al alcanzar el límite de tamaño) de forma diaria, conservando un máximo de 7 días de logs y comprimiendo los antiguos (Véase Figura 2.13). La sección `postrotate` envía una señal de recarga a Snort tras la rotación, para garantizar que Snort vuelve a abrir correctamente el nuevo fichero de log (esto es relevante sobre todo si por alguna razón se rotase el fichero activo antes de que Snort lo haga por su cuenta). Gracias a este mecanismo combinado, el sistema mantiene un archivo *activo* de alertas para ingestión en tiempo real, a la vez que almacena de forma segura los archivos históricos de alertas (ya comprimidos) en caso de que se requiera algún análisis forense o consulta retroactiva de eventos.

```

1  /var/log/snort/alert_json.txt* {
2      daily
3      rotate 7
4      compress
5      missingok
6      notifempty
7      postrotate
8          systemctl reload snort3.service > /dev/null 2>&1 || true
9      endscript
}

```

Código 2.4: Regla de logrotate para los logs de Snort

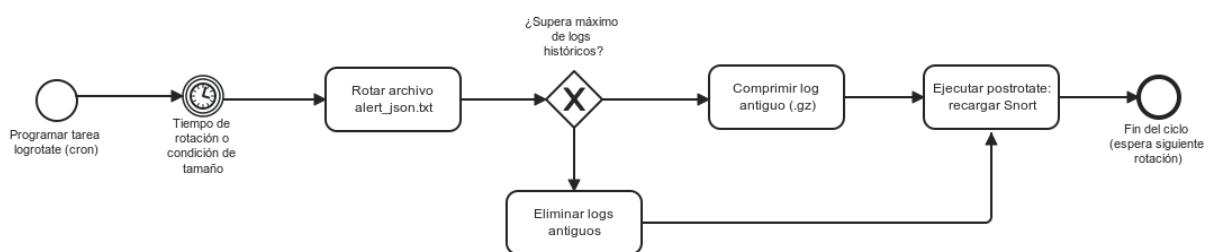


Figura 2.13: BPMN del servicio de logrotate y archivado.

### Integración de Grafana con base de datos MariaDB

Una vez que las alertas y métricas de Snort se encuentran almacenadas en MariaDB, se procedió a integrar una instancia de **Grafana** para visualizar dicha información de manera dinámica e interactiva. Grafana es una plataforma de visualización de series temporales y datos operativos que, mediante paneles (dashboards) personalizables, permite monitorear tanto las alertas de seguridad generadas por Snort como el estado del propio agente.

En primer lugar, se configuró Grafana para que utilizara a MariaDB como fuente de datos. Grafana soporta MySQL/MariaDB mediante un conector nativo, por lo que se creó un *Data Source* de tipo MySQL apuntando al servidor MariaDB donde residían las tablas de alertas y métricas. Se le proporcionaron las credenciales de solo-lectura adecuadas. A continuación, se diseñó el dashboard de Grafana. Para acelerar este desarrollo, se partió de una plantilla pública existente para Snort (originalmente creada para stacks ELK/InfluxDB) y se adaptó a la solución basada en MariaDB [27]. Concretamente, se tomó como referencia el dashboard *Snort IDS/IPS* (ID 11191 en Grafana), reutilizando su disposición y elementos visuales (gráficas de líneas, tablas, paneles de métricas clave, mapas, etc.), pero modificando las consultas de datos para ajustarlas al esquema SQL de nuestro sistema y a las métricas disponibles.

En la adaptación, cada panel del dashboard original fue traducido a una consulta SQL equivalente. Por ejemplo, donde el template original empleaba una consulta en InfluxQL (InfluxDB) para contar eventos por severidad o tipo de ataque, ahora se utiliza una consulta SQL sobre la tabla `alertas` agrupando por el campo de severidad o clasificación. Del mismo modo, los paneles que muestran series temporales (ej. número de alertas en el tiempo, tráfico por protocolo) se implementaron con consultas SQL que aprovechan las funciones de tiempo de MariaDB (p. ej., agrupando por minuto u hora usando `DATE_FORMAT`). A continuación se presenta una consulta simplificada (Código 2.5) empleada en el panel de “top protocolos” del dashboard adaptado:

```
1   SELECT proto AS protocolo, COUNT(*) AS total_alertas
2   FROM alertas
3   WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 24 HOUR)
4   GROUP BY proto
5   ORDER BY total_alertas DESC;
```

Código 2.5: Consulta SQL (simplificada) para panel de protocolos en Grafana

Esta consulta devuelve el número de alertas detectadas en un intervalo de tiempo de red observado, ordenando de mayor a menor frecuencia. Consultas similares se prepararon para otros paneles: conteo de alertas por nivel de severidad (mapeando el campo numérico `priority` de Snort a etiquetas descriptivas “Alta”, “Media”, “Baja”), listados de las “Top N” reglas disparadas, direcciones IP de origen más frecuentes en las alertas, y gráficos de series temporales mostrando la evolución de alertas en el tiempo. También se incorporó un panel tipo tabla para mostrar el detalle de las últimas alertas registradas (hora, descripción, IP de origen y destino, puerto, clasificación y severidad), obteniendo directamente los registros más recientes de la tabla `alertas`.

La Figura 2.14 muestra una captura del dashboard resultante. Este panel de control ofrece una vista unificada del estado de seguridad monitorizado por el agente: en la parte superior, métricas agregadas como el número de alertas recientes categorizadas por severidad; en el centro, gráficas de pastel destacando los principales tipos de alerta detectados y los protocolos de red involucrados; a la derecha, los IP de origen más comunes en las alertas (potenciales fuentes maliciosas); y en la parte inferior, una tabla actualizada con las últimas alertas generadas, permitiendo al analista ver detalles puntuales de cada evento de seguridad.

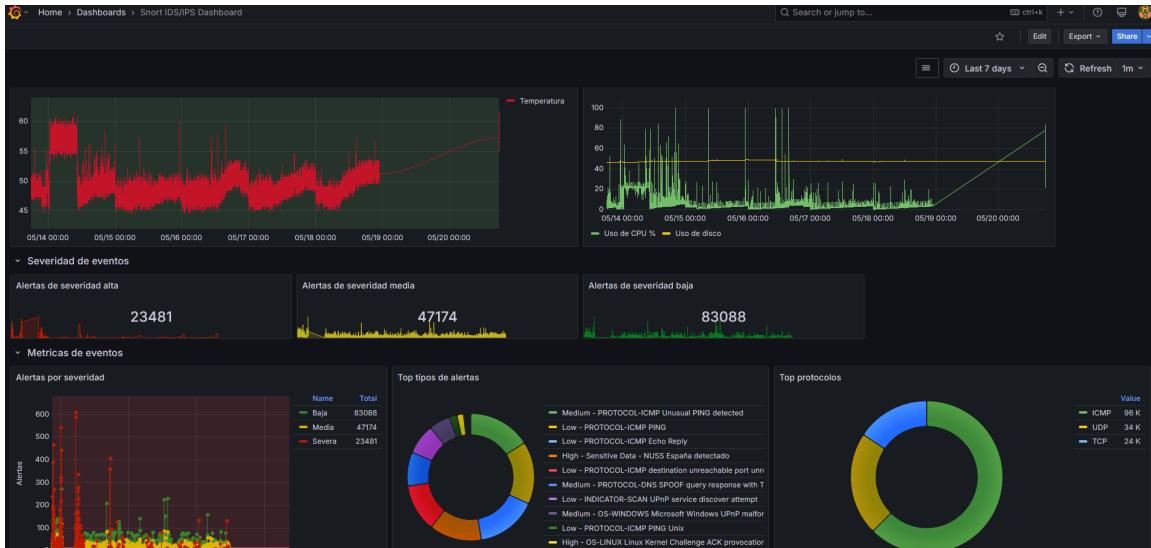


Figura 2.14: Dashboard de Grafana adaptado para R-Snort.

Para lograr que este dashboard esté disponible inmediatamente tras desplegar un agente R-Snort, se automatizó su provisión. Se preparó el JSON de la plantilla adaptada y, durante el proceso de instalación del agente, un script de configuración importa dicho dashboard en Grafana de forma programática (utilizando la API HTTP de Grafana o su mecanismo de *provisioning*). De esta manera, el usuario no necesita configurar manualmente el panel: una vez que Grafana arranca apuntando a la base de datos de Snort, el dashboard personalizado aparece pre-cargado y listo para su uso. Esta automatización reduce posibles errores de configuración y asegura la consistencia del entorno de monitorización en todos los despliegues del sistema (véase Figura 2.15).

En resumen, mediante la integración con Grafana se concreta la faceta “monitorizable” de R-Snort: el administrador de seguridad dispone de una interfaz gráfica intuitiva desde la cual puede supervisar en tiempo real la actividad del IDS y la salud del agente. La combinación de alertas almacenadas en MariaDB con visualizaciones de Grafana adaptadas a dichas alertas proporciona una solución ligera pero efectiva, que cubre las necesidades de un SIEM básico: detección (Snort), recolección/almacenamiento (servicio de almacenamiento + base de datos) y presentación de información (Grafana) en un solo sistema cohesionado.

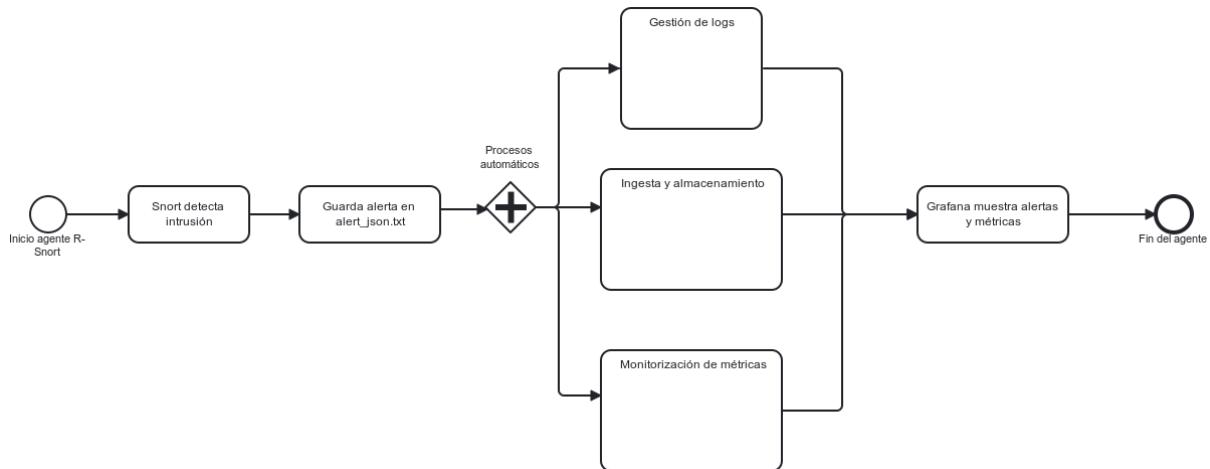


Figura 2.15: Diagrama de funcionamiento del agente.

### 2.5.2. Implementación del backend

El sistema R-Snort cuenta con una arquitectura de backend dividida en dos partes principales: un backend de agentes, implementado con FastAPI (Python) y ejecutado en cada nodo sensor; y un backend central, desarrollado en Spring Boot (Java), que coordina la información de todos los agentes y provee servicios a la interfaz web. A continuación se detallan ambos componentes en términos de endpoints, arquitectura y mecanismos de seguridad.

#### Backend de los agentes (FastAPI)

Cada agente R-Snort ejecuta un servicio web ligero construido con FastAPI, el cual expone múltiples endpoints REST para monitorear y controlar remotamente el sensor. Estos endpoints permiten consultar el estado de Snort, obtener las alertas detectadas, extraer métricas del sistema, gestionar las reglas personalizadas de detección y manejar archivos de registro, entre otras acciones (véase Figura 2.16 y Tabla 2.1).

## R-Snort Agent API 1.2.0 OAS 3.1

[/openapi.json](#)

### default

<code>GET</code>	<code>/status</code> Status
<code>GET</code>	<code>/services/status</code> Get Service Status
<code>POST</code>	<code>/services/restart</code> Restart Service
<code>GET</code>	<code>/alerts</code> Get Alerts
<code>GET</code>	<code>/metrics</code> Get Metrics
<code>GET</code>	<code>/rules</code> Get Rules
<code>POST</code>	<code>/rules</code> Add Rule
<code>DELETE</code>	<code>/rules/{sid}</code> Delete Rule
<code>GET</code>	<code>/archived-files</code> List Archived Files
<code>GET</code>	<code>/archived-files/{filename}</code> Download Archived File
<code>GET</code>	<code>/download-alerts</code> Descargar alertas activas como CSV
<code>GET</code>	<code>/alerts/last</code> Get Last Alert

Figura 2.16: Documentación Swagger de la API de un agente R-Snort.

Método	Endpoint	Funcionalidad
GET	<code>/status</code>	Obtener un estado básico del agente (sirve como <i>ping</i> de salud).
GET	<code>/services/status</code>	Consultar el estado de los servicios internos del agente (por ejemplo, si Snort y otros procesos están activos).
<b>POST</b>	<code>/services/restart</code>	Reiniclar el servicio de detección en el agente (p.ej., reinicia el proceso Snort u otro servicio especificado).
GET	<code>/alerts</code>	Listar las alertas recientes o activas detectadas por Snort en ese agente.
GET	<code>/metrics</code>	Obtener métricas del sistema local (uso de CPU, memoria, temperatura, etc.) del agente.
GET	<code>/rules</code>	Listar las reglas de Snort en el agente, incluyendo reglas personalizadas añadidas.
<b>POST</b>	<code>/rules</code>	Añadir una nueva regla de Snort personalizada en el agente. El cuerpo de la petición incluye la regla en formato Snort.
<b>DELETE</b>	<code>/rules/{sid}</code>	Eliminar una regla personalizada identificada por su SID del conjunto de reglas del agente.
GET	<code>/archived-files</code>	Listar archivos de log archivados en el agente (logs históricos de alertas, comprimidos o rotados).
GET	<code>/archived-files/{filename}</code>	Descargar un archivo de log archivado específico desde el agente.
GET	<code>/download-alerts</code>	Descargar todas las alertas activas del agente en formato CSV.
GET	<code>/alerts/last</code>	Obtener la información de la última alerta detectada por el agente.

Tabla 2.1: Resumen de los endpoints expuestos por el backend del agente R-Snort.

En términos de implementación, cada endpoint del agente está manejado por una función Python utilizando FastAPI. Por ejemplo, un fragmento simplificado para eliminar una regla podría ser:

```
1 @app.delete("/rules/{sid}")
2 def delete_rule(sid: int):
3     # Eliminar la regla con el SID dado del archivo de reglas de Snort
4     if remove_rule_from_file(sid):
5         return {"detail": "Regla eliminada"}
6     raise HTTPException(status_code=404, detail="Regla no encontrada")
```

Código 2.6: Fragmento del código que maneja cada endpoint

Mediante estos endpoints, el backend de un agente permite al servidor central y a la interfaz web consultar y modificar el estado de ese sensor. La comunicación se realiza a través de HTTP: el backend central invoca las rutas del agente según sea necesario. Por ejemplo, si el usuario solicita ver las alertas de un agente en la interfaz, el servidor central hará una petición GET al endpoint /alerts de ese agente para obtener los datos. Del mismo modo, al añadir o eliminar una regla desde la interfaz, el servidor central enviará una petición POST/DELETE al agente correspondiente. Esto mantiene la lógica de recolección de datos en el propio nodo sensor, mientras que la lógica de negocio ocurre en el nivel central.

### Backend central (Spring Boot)

El **backend central** es una aplicación desarrollada con Spring Boot, encargada de coordinar todos los agentes y de ofrecer una **API unificada** al frontend. Este servidor central expone endpoints REST (p. ej., bajo un prefijo /api) que la aplicación Angular consume para obtener información de alertas, reglas, estado de agentes, etc. Internamente, el backend central actúa como **intermediario**: recibe las solicitudes de la interfaz web, aplica lógica de negocio y, cuando es necesario, consulta a los agentes remotos a través de los endpoints antes descritos.

Se ha optado porque el backend central siga una **arquitectura por capas**, separando responsabilidades en distintos paquetes dentro del proyecto Spring Boot. Esta organización facilita la mantenibilidad y la escalabilidad del sistema. En particular, se definen las siguientes capas principales:

- **Controller (Controlador).** Maneja las peticiones HTTP entrantes desde el frontend. Por ejemplo, un AgentController podría gestionar rutas como /api/agents/{id}/alerts para devolver las alertas de un agente dado. Los controladores procesan los parámetros de la petición (p. ej., identificación del agente) y delegan la lógica al servicio correspondiente. Cada método del controlador típicamente devuelve datos en formato JSON (usando las clases DTO apropiadas) al cliente.
- **Service (Servicio).** Contiene la **lógica de negocio** de la aplicación. Los servicios implementan funcionalidades como consultar las alertas de un agente, agregar una nueva regla o autenticar a un usuario. En esta capa es donde el backend central interactúa con los agentes remotos: por ejemplo, un método del servicio de alertas realizará internamente una petición HTTP al endpoint /alerts de un agente para obtener sus alertas, o iterará sobre varios agentes para consolidar información. También es la capa donde se aplica lógica adicional (filtros, transformaciones, combinaciones de datos de múltiples fuentes, etc.).

- **Repository (Repositorio).** Gestiona la **persistencia de datos** en la base de datos. Aunque muchas de las alertas y métricas se obtienen *on-demand* desde los agentes, el sistema central puede almacenar información persistente como los usuarios (credenciales), la lista de agentes registrados y quizás un historial consolidado o metadatos de alertas. Los repositorios (basados en JPA/Hibernate u otro ORM) proporcionan métodos CRUD para estas entidades persistentes. Por ejemplo, podría haber un `UserRepository` para consultar usuarios en la base de datos, o un `AgentRepository` para registrar la información (IP, nombre, clave de acceso) de cada agente.
- **Model (Modelo/Entidad).** Define las clases de dominio que representan los datos manejados por el sistema. Por ejemplo, podrían existir clases entidad como `Alerta`, `Regla`, `Agente` o `Usuario`, que reflejan tablas en la base de datos y sirven como modelo de datos interno. Estas clases suelen anotarse con `@Entity` en JPA para mapear a la base de datos, en caso de haber persistencia, y se utilizan en la capa de servicio y repositorio.
- **DTO (Data Transfer Object).** Son clases simplificadas utilizadas para **transferir datos** hacia y desde el frontend. Los DTO se utilizan para no exponer directamente las entidades internas a la capa de presentación y para enviar solamente los campos necesarios. Por ejemplo, el backend podría definir un `AlertaDTO` con campos como `fecha`, `tipo`, `severidad`, `origen`, `destino`, etc., que el controlador rellena a partir de las entidades o de la respuesta del agente antes de enviarlo al frontend.
- **Security (Seguridad) y Configuración.** Agrupa las clases relacionadas con la **seguridad de la aplicación** (por ejemplo, configuración de CORS, filtros de autenticación JWT, gestión de roles/permisos) y otras configuraciones globales. Aquí es donde se configura Spring Security para integrar JWT (ver siguiente apartado), definiendo qué endpoints requieren autenticación, y se implementan filtros que interceptan las peticiones para validar tokens. También se incluyen clases de configuración general de Spring Boot (beans, propiedades, etc.) y constantes como la clave secreta para JWT (manteniéndola segura, idealmente en variables de entorno).

Esta organización modular permite que el backend central sea **extensible**. Por ejemplo, si se quisiera agregar una nueva funcionalidad (como notificaciones por correo de ciertas alertas), se podría añadir un nuevo servicio y controlador sin alterar las demás capas.

A nivel de interacción con los agentes, el servidor central utiliza **clientes HTTP** para consumir las APIs FastAPI. Por ejemplo, Spring Boot puede usar la clase `RestTemplate` o la librería `WebClient` para realizar solicitudes REST a los agentes. Un fragmento de código ilustrativo del servicio central de alertas podría ser:

```
String url = "http://" + agente.getIp() + ":8000/alerts";
2 ResponseEntity<Alerta[]> resp = restTemplate.getForEntity(url, Alerta[].class);
Alerta[] alertasAgente = resp.getBody();
4 // ... (procesar las alertas obtenidas según la lógica de negocio)
```

Código 2.7: Ejemplo simplificado de llamada desde el backend central a un agente

En el ejemplo anterior, el servicio construye la URL del agente (usando su IP y puerto conocido, aquí 8000 por defecto para FastAPI) y realiza una petición GET al endpoint `/alerts`.

La respuesta (un arreglo de alertas JSON) se mapea a objetos Alerta automáticamente. De manera similar, se invocan los demás endpoints del agente según se necesite (por ejemplo, `restTemplate.postForObject` para agregar una regla, o `delete` para eliminarla). Finalmente, el servicio devuelve la información obtenida (o el resultado de la acción) al controlador, que a su vez lo envía al frontend en formato JSON.

## Autenticación JWT

Para proteger el acceso a la API central, R-Snort implementa un sistema de autenticación **JWT (JSON Web Token)**. En el backend central, sólo el endpoint de autenticación (por ejemplo `/api/login` o `/auth`) se expone sin seguridad; el resto de endpoints requieren un token válido. El proceso funciona de la siguiente manera:

1. **Inicio de sesión.** El cliente (frontend) envía las credenciales de usuario (p. ej., nombre de usuario y contraseña) al endpoint de login del backend central. El backend verifica esas credenciales comparándolas con los usuarios almacenados (por ejemplo, en la base de datos, probablemente con las contraseñas en hash por seguridad).
2. **Generación de token.** Si las credenciales son correctas, el servidor genera un token JWT firmado con una clave secreta conocida sólo por el servidor. Este token incluye en su *payload* la identidad del usuario (y opcionalmente roles o privilegios) y una fecha de expiración. La firma criptográfica asegura que el token no pueda alterarse sin conocimiento de la clave secreta.
3. **Envío del JWT al cliente.** El token JWT se envía de vuelta al frontend (normalmente en el cuerpo de la respuesta del login). A partir de ese momento, el cliente guarda este token (típicamente en `localStorage` o `sessionStorage` del navegador) y lo incluye en cada petición subsiguiente al backend central, usando la cabecera HTTP: `Authorization: Bearer <token>`.
4. **Validación en cada petición.** Del lado del servidor, se configura un filtro de seguridad (por ejemplo, un `JWTFilter` en `Spring Security`) que intercepta todas las peticiones entrantes. Este filtro extrae el token de la cabecera `Authorization`, y verifica su validez: comprueba la firma con la clave secreta (asegurando que no haya sido modificada), y valida que no esté expirado. Si el token es válido, el filtro permite el paso de la petición y marca el contexto de seguridad como autenticado (asociando el token o la identidad del usuario a la petición actual). En caso contrario (token ausente, inválido o expirado), el filtro bloquea la petición y devuelve un error `401 Unauthorized`.
5. **Acceso a recursos protegidos.** Con este mecanismo, sólo usuarios autenticados (poseedores de un JWT válido) pueden acceder a los endpoints protegidos del backend central, por ejemplo para obtener alertas o añadir reglas. El backend central permanece *stateless* (sin sesión server-side), ya que la autenticación se confía al token que el cliente presenta en cada solicitud.

Este sistema JWT brinda una forma escalable y segura de manejar la autenticación: elimina la necesidad de mantener sesiones en el servidor y funciona bien con la arquitectura de SPA. Además, el token puede incluir información adicional (*claims*) como el rol del usuario,

permitiendo que el backend autorice ciertas operaciones solo a administradores, por ejemplo. En R-Snort, dado que es una aplicación de gestión centralizada, es probable que todos los usuarios autenticados tengan permisos similares (o exista al menos un rol administrador), pero el mecanismo admite extensiones. La clave secreta para firmar/verificar los tokens se define en la configuración del servidor (por seguridad, es importante que esté protegida y no se exponga). Asimismo, se estableció una política de expiración del token (por ejemplo, el token expira tras cierta cantidad de minutos u horas), de modo que se mitiga el riesgo en caso de robo del token y se puede requerir re-autenticación pasado un tiempo.

### Resumen del backend

En resumen, la implementación del backend de R-Snort se basa en una arquitectura **distribuida y modular**. Los agentes ejecutan procesos FastAPI que interactúan directamente con Snort (leyendo sus logs, gestionando su configuración y estado) y exponen endpoints REST para esas funciones. El servidor central Spring Boot coordina estos agentes: actúa como un hub que recibe las solicitudes de la interfaz web, valida la seguridad mediante JWT, y delega en los agentes las operaciones específicas, consolidando la información cuando es necesario (véase Figura 2.17). Gracias a la separación por capas (controladores, servicios, repositorios, etc.), el backend central es mantenible y escalable, y el uso de JWT garantiza que sólo usuarios autorizados puedan gestionar el sistema. Esta combinación permite administrar de forma centralizada múltiples sensores Snort de manera segura y eficiente.

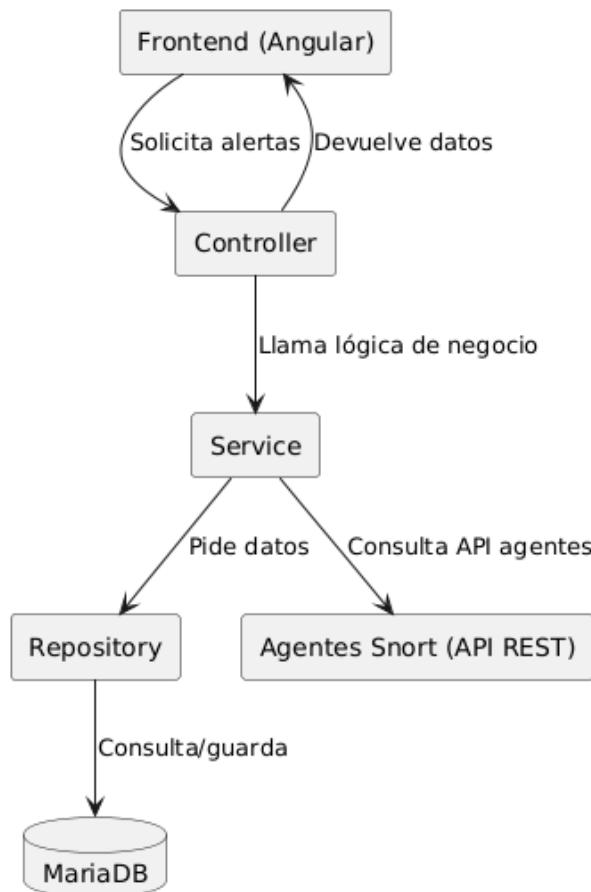


Figura 2.17: Flujo de funcionamiento del backend.

### 2.5.3. Implementación del frontend

El **frontend de R-Snort** se desarrolló como una **SPA (Single Page Application)** usando Angular, lo que proporciona una experiencia de usuario interactiva y fluida en el navegador, similar a una aplicación de escritorio. La interfaz está diseñada para que un administrador de seguridad pueda visualizar el estado de la red y configurar el sistema de detección de intrusos de forma unificada. A nivel de organización, la aplicación Angular se estructuró en distintos módulos o secciones, correspondientes a las funcionalidades principales: alertas, reglas, vista general (overview), estado del sistema y autenticación (login). Cada sección agrupa componentes, servicios y otros elementos específicos de esa funcionalidad, manteniendo el código modular y claro.

#### Módulos principales de la SPA

- **Login.** Maneja el proceso de autenticación de usuarios. Contiene el componente de inicio de sesión, donde se solicitan las credenciales. Al autenticarse correctamente, este módulo se encarga de almacenar el token JWT recibido y redirigir al usuario hacia la interfaz principal.
- **Overview (Vista general).** Provee un tablero de control con estadísticas globales del sistema IDS. En esta sección se concentran visualizaciones resumidas de la información de seguridad, como gráficos de la cantidad de alertas por severidad a lo largo del tiempo, distribuciones de los tipos de alerta más frecuentes, protocolos más detectados, y otras métricas agregadas. Sirve para que el administrador obtenga de un vistazo el panorama de la actividad reciente en la red monitoreada.

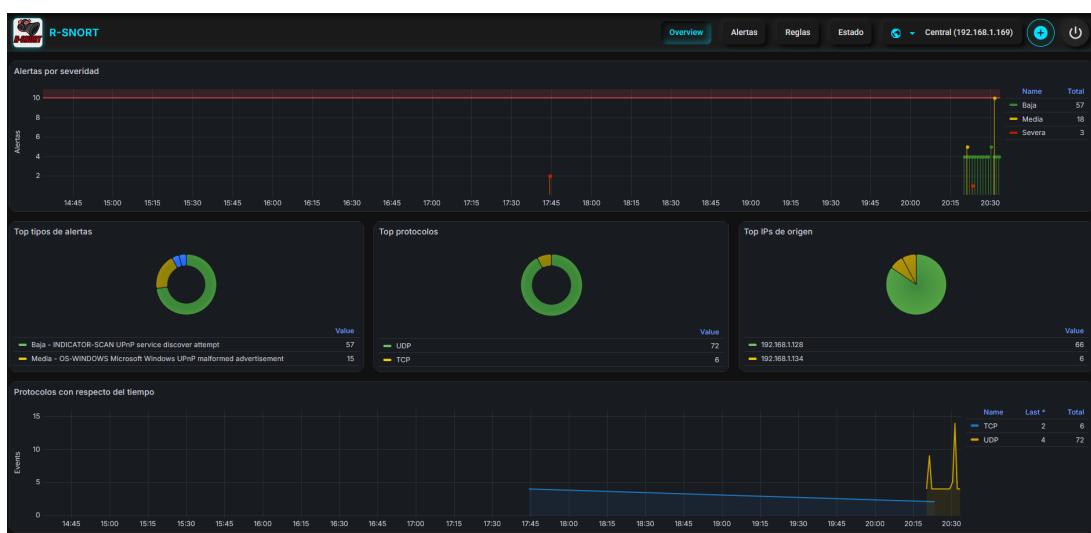


Figura 2.18: Vista general.

- **Alertas.** Incluye los componentes para visualizar las alertas de seguridad provenientes de un agente seleccionado. Presenta un Panel de Alertas con indicadores numéricos de alertas por nivel de severidad (alta, media, baja) y una tabla con el listado de las últimas alertas detectadas con sus detalles (hora, tipo, origen, destino, etc.). Además, provee acciones para exportar datos, como botones para “Descargar alertas de este agente” o “Descargar todas las alertas” en formato CSV, facilitando el análisis externo de los eventos.



Figura 2.19: Vista de alertas del agente seleccionado en la interfaz R-Snort.

- **Reglas.** Ofrece la interfaz para gestionar reglas de Snort de forma centralizada. Un componente de este módulo permite agregar una nueva regla mediante un formulario donde el usuario escribe la regla en formato Snort (texto), y enviarla al backend. Las reglas personalizadas existentes se muestran en una lista, incluyendo su SID (identificador), mensaje descriptivo y el contenido de la regla, con la opción de eliminarlas individualmente mediante un ícono de papelera. Si la regla no cuenta con la sintaxis correcta es rechazada.

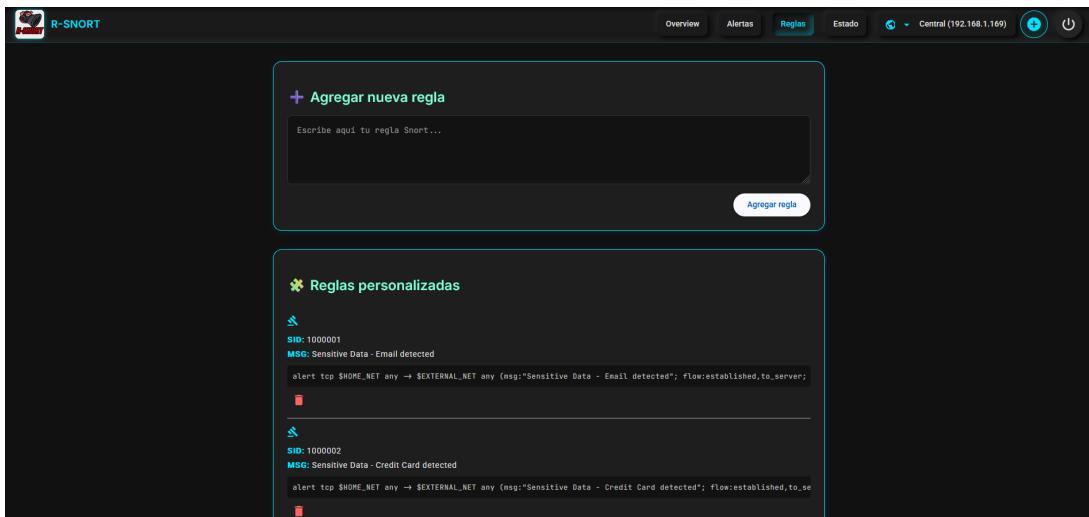


Figura 2.20: Reglas del sistema de R-Snort.

- **Estado.** Muestra información en tiempo real del estado operativo tanto del agente seleccionado como del sistema central. Incluye gráficos de métricas del sistema (por ejemplo, temperatura de la CPU, uso de CPU y disco) y paneles de estado que indican si los servicios del agente están activos. También se presentan botones para reiniciar procesos como Snort (“Reiniciar Snort”) y una lista de logs archivados disponibles para descarga.

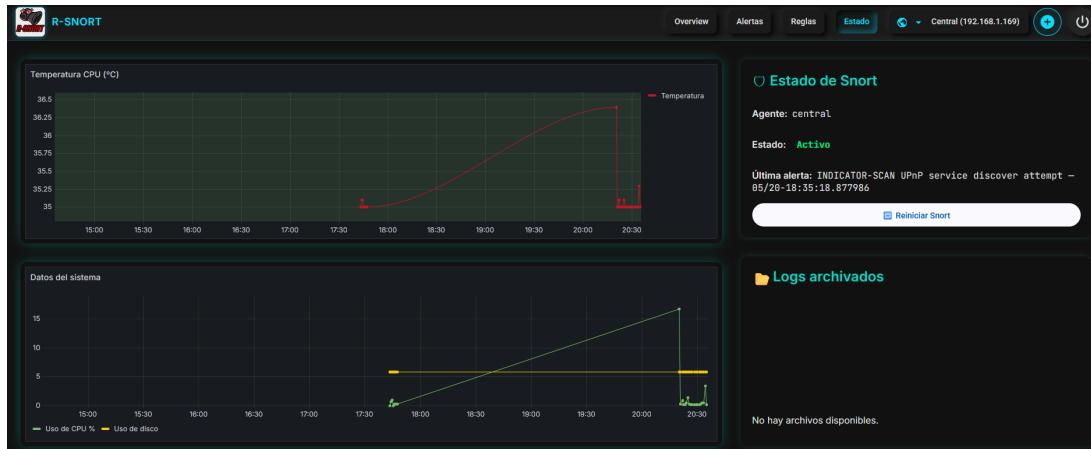


Figura 2.21: Estado del agente seleccionado.



Figura 2.22: Estado de los servicios de un agente individual.

Además de estos módulos principales, la aplicación cuenta con componentes auxiliares como un **header** (encabezado) que contiene el menú de navegación y el selector de agente, **servicios compartidos** (por ejemplo, servicios para hacer peticiones HTTP al backend central, manejo del token JWT, etc.), **guards e interceptors**, y elementos de UI reutilizables (diálogos de confirmación, notificaciones, pipes para formateo, etc.). Toda esta estructura sigue la conveniencia de Angular de separar responsabilidades: **componentes** para la presentación, **servicios** para la lógica de datos/comunicación, y **guardas/interceptores** para control de flujo y seguridad en la interfaz.

## Enrutamiento y selección dinámica de agentes

La SPA Angular define un sistema de rutas (*routing*) para navegar entre las distintas secciones (login, overview, alertas, reglas, estado). Cada ruta está asociada a un componente principal de los mencionados módulos. Por ejemplo, se configuran rutas como /login para la pantalla de autenticación, /overview para la vista general, /alertas para el panel de alertas, etc. Una vez autenticado el usuario, navegar entre estas rutas no provoca recarga de la página, sino que Angular se encarga de cargar dinámicamente el componente correspondiente dentro de la misma aplicación.

Un requerimiento particular de R-Snort es la capacidad de visualizar información de múltiples agentes de forma centralizada. Esto se resuelve en la interfaz mediante un mecanismo de **selección dinámica de agentes**. En la parte superior derecha de la aplicación (ver figuras), el usuario dispone de un menú desplegable que lista los agentes registrados. El usuario puede cambiar el agente activo seleccionándolo en ese menú, con lo cual la aplicación actualiza la información mostrada en todos los módulos (alertas, reglas, overview, estado) para reflejar los datos de ese agente.

Se pretende manejar la selección de agente mediante un **servicio global** de Angular (por ejemplo, un AgentService que almacena el agente seleccionado en memoria). En este enfoque, las rutas no cambian cuando se selecciona otro agente; más bien, los componentes consultan al AgentService para saber qué agente utilizar en sus peticiones. Un cambio en la selección actualizaría dicho servicio y los componentes reaccionarían (por ejemplo, recargando datos). Esta aproximación evita tener parámetros en todas las rutas, a costa de no tener una URL única por agente (pero dado que el objetivo es administración interactiva, eso no suele ser un problema).

En la implementación de R-Snort se ha optado por la comodidad de poder cambiar de agente sin recargar la aplicación. Al iniciar sesión, típicamente se selecciona un agente por defecto (por ejemplo, el agente "central") para mostrar. Luego, al elegir otro agente en el menú desplegable, la aplicación realiza internamente las solicitudes al backend central para ese agente y actualiza las vistas. Todos los componentes clave (alertas, reglas, estado, etc.) están preparados para reaccionar ante un cambio de agente y volver a cargar sus datos desde el servidor central.

El sistema de rutas de Angular también incorpora **guardas (guards)** para restringir el acceso a ciertas rutas. En particular, se utiliza un **guardia de autenticación** que impide el acceso a las rutas de la aplicación principal (overview, alertas, etc.) si el usuario no ha iniciado sesión. Esto se configura de forma que, si Angular detecta que no hay un token JWT válido almacenado (o que el usuario no ha pasado por login), automáticamente redirige la navegación hacia la ruta /login. A continuación se muestra un ejemplo simplificado de un guard de autenticación:

```
// Ejemplo de AuthGuard en Angular
2 @Injectable({ providedIn: 'root' })
3 export class AuthGuard implements CanActivate {
4   constructor(private authService: AuthService, private router: Router) {}
5   canActivate(): boolean {
6     if (!this.authService.isLoggedIn()) {
7       this.router.navigate(['/login']);
8       return false;
9     }
10    return true;
11  }
12 }
```

Código 2.8: Ejemplo simplificado de authGuard

En el código anterior, el guard (AuthGuard) consulta a un servicio de autenticación AuthService para verificar si el usuario está logueado. El método isLoggedIn() típicamente comprueba si existe un JWT almacenado y vigente. Si no lo está, se realiza un redirect a la página de login y se bloquea la activación de la ruta solicitada; si sí lo está, se permite el acceso. Este guard se aplica, por ejemplo, a todas las rutas bajo /agents/\* o a un grupo de rutas definidas para la parte protegida de la app.

Asimismo, para hacer transparente el uso del JWT en el frontend, se ha implementado un **interceptor HTTP**. Este interceptor global intercepta todas las peticiones HTTP salientes desde Angular hacia el backend central y añade automáticamente la cabecera Authorization: Bearer <token> con el JWT que fue almacenado tras el login. De esta manera, los componentes y servicios del frontend no tienen que preocuparse de adjuntar manualmente el token en cada llamada; el interceptor garantiza que el backend reciba el token en cada petición, manteniendo la sesión autenticada.

Si el token expirara o fuera inválido, el backend responderá con un error de no autorizado, y en respuesta la aplicación podría redirigir al usuario al login nuevamente. Esta lógica puede manejarse también en el interceptor o en un manejador centralizado de respuestas.

### Integración de Grafana para visualización de métricas

Una característica destacada de R-Snort es la integración de paneles de **Grafana** en la interfaz web para mostrar **gráficos y métricas en tiempo real**. Grafana es una herramienta especializada en visualización de datos de monitorización, y su uso permite aprovechar gráficos interactivos sin tener que implementarlos desde cero.

En el contexto de R-Snort, se ha utilizado Grafana para representar principalmente las métricas de rendimiento del sistema y **series temporales de las alertas**.

### Integración de Grafana en la interfaz Angular

La integración técnica con Grafana se realiza mediante la inserción de sus dashboards en la página Angular, aprovechando la opción de **Iframes embebidos** de Grafana que permite

compartir paneles mediante URLs especiales. En R-Snort, se utilizan `iframes` en los componentes Angular (como `OverviewComponent`) para incrustar paneles específicos. Por ejemplo, un `<iframe>` puede apuntar a una URL compartida por Grafana que renderiza únicamente la gráfica en modo `kiosk`, sin menús ni cabecera. Este método es sencillo y permite mostrar los datos en tiempo real sin complejidad adicional en el frontend.

En cuanto a la **autenticación**, Grafana puede configurarse con acceso anónimo de solo lectura o mediante API Keys incrustadas en la URL. En R-Snort, se ha garantizado que el acceso a los gráficos esté restringido de forma segura (por ejemplo, corriendo Grafana localmente o con un usuario de sólo visualización). Cuando el usuario entra en las vistas `Overview` o `Estado`, los gráficos se cargan automáticamente según el agente seleccionado, ya sea modificando el `iframe` o pasando variables dinámicas a Grafana.

Este tipo de integración aporta valor visual al frontend con mínimo esfuerzo, permitiendo combinar datos detallados con visualizaciones agregadas en una sola interfaz unificada.

### Ejemplo de interacciones en el frontend

A continuación se muestran ejemplos representativos del funcionamiento del frontend.

**Login y almacenamiento del token JWT** Cuando el usuario inicia sesión, el token JWT es recibido y almacenado, y se redirige a la vista principal:

```
1  this.authService.login(this.username, this.password).subscribe({
2    next: () => this.router.navigate(['/overview']),
3    error: () => this.loginError = true
});
```

Código 2.9: Login y navegación tras autenticación

**Carga de alertas del agente seleccionado** El componente de alertas carga las alertas desde el backend, pasando el ID del agente seleccionado:

```
1  this.alertService.getAlertas(this.agenteSeleccionado.id).subscribe(alertas => {
2    this.alertas = alertas;
});
```

Código 2.10: Carga de alertas desde el servicio

Gracias al **interceptor HTTP**, el token JWT es incluido automáticamente en la cabecera `Authorization` de cada petición, por lo que los componentes no necesitan preocuparse por ello.

### Resumen del frontend

En síntesis, el frontend de R-Snort proporciona una interfaz SPA modular, fluida y responsive que permite:

- Navegar entre módulos como `alertas`, `reglas`, `estado` o `overview` sin recarga.
- Cambiar entre agentes en tiempo real mediante un selector global.

- Consultar alertas, gestionar reglas y visualizar métricas de forma centralizada.
- Integrar paneles de Grafana para enriquecer la experiencia visual.
- Proteger rutas con **guards** y mantener autenticación mediante JWT.

Esto permite al usuario controlar múltiples sensores Snort desde una única interfaz, combinando vistas detalladas (tablas de alertas o reglas) con paneles gráficos y opciones de administración remota.

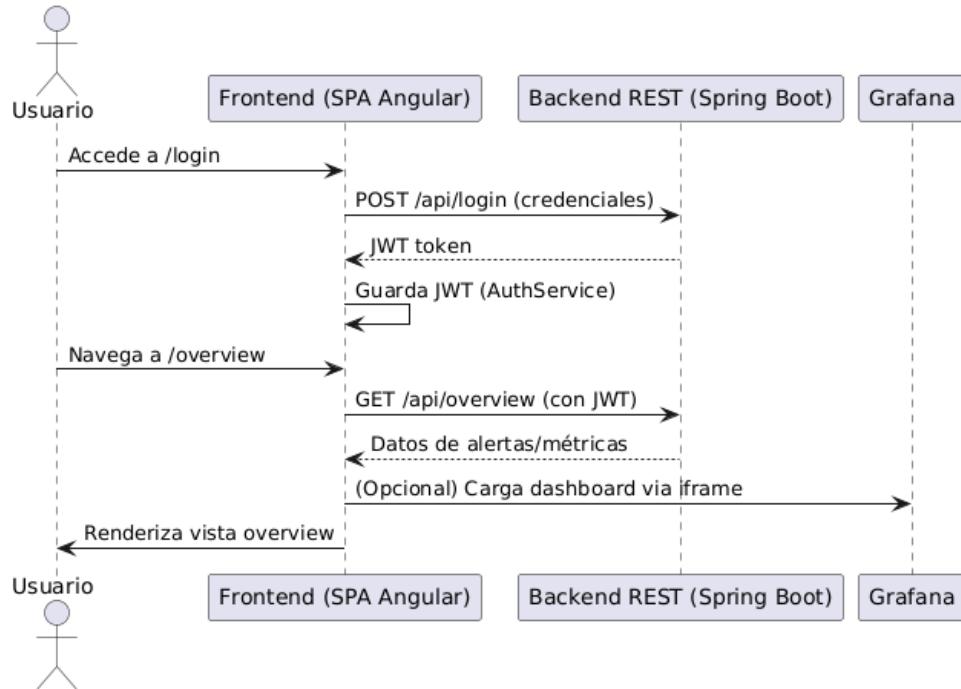


Figura 2.23: Diagrama de secuencia de interacción.

## 2.6. Script automático R-Snort

El sistema R-Snort se estructura en torno a dos componentes principales: el `snort-agent`, encargado de desplegar un nodo completo de detección de intrusiones en cada dispositivo monitorizado, y el `r-snort-central-module`, que actúa como módulo central de gestión y visualización. A continuación se describe en detalle el funcionamiento del instalador `snort-agent` y sus peculiaridades técnicas.

### 2.6.1. Instalador del agente: `snort-agent`

El script `snort-agent` automatiza la instalación y configuración de un agente R-Snort sobre sistemas Debian/Ubuntu con una instancia de Snort versión 3 ya sea con la configuración por defecto o personalizada, la base de datos local, servicios de monitorización REST y la integración con Grafana. El instalador está compuesto por varios scripts Bash y módulos Python que permiten que el agente funcione de forma autónoma tras su ejecución. El objetivo del instalador automático desde un principio se centra en usuarios con poco conocimiento técnico haciéndolo prácticamente plug and play.

**Script 00\_common.sh.** Este script inicial crea la estructura de configuración del agente, incluyendo:

- Generación de un identificador UUID persistente para identificar únicamente al agente.
- Creación de archivos de entorno (db.cnf, env.sh y env.service) que almacenan credenciales y configuración necesaria para el resto de scripts y servicios.

```
1 cat >"$DB_CNF" <<EOF
2 [client]
3 user=$DB_USER
4 password=$DB_PASS
5 database=$DB_NAME
6 host=127.0.0.1
7 EOF
```

Código 2.11: Fragmento de creación del archivo db.cnf

**Script 01\_install\_db.sh.** Este script instala MariaDB y crea el esquema de base de datos para las alertas y métricas. Las tablas alerts y system\_metrics permiten almacenar información relevante como el origen/destino del ataque, el tipo de alerta, el uso de CPU, entre otros.

**Script 02\_configure\_snort.sh.** Configura Snort para ejecutarse como servicio en segundo plano, escaneando el tráfico de red a través de la interfaz detectada automáticamente. También modifica el archivo snort.lua para incluir la salida alert\_json y define un servicio systemd. Una particularidad es que el script escoge la interfaz activada en modo promiscuo para la tarea de escuchar el tráfico de red, en caso de no haber ninguna interfaz en dicho modo escogerá escuchar a través de la interfaz usada originalmente por Snort.

```
1 [Service]
2 ExecStart=/usr/local/snort/bin/snort -q -c /usr/local/snort/etc/snort/snort.lua -i
3     ↳ $IFACE -A alert_json -l /opt/snort/logs/live
4 Restart=always
```

Código 2.12: Definición del servicio systemd para Snort

**Script 03\_log\_rotation.sh.** Implementa un mecanismo de rotación de logs usando logrotate, que evita el crecimiento descontrolado del archivo alert\_json.txt. Además, se configura un cron job para archivado diario de logs.

**Script 04\_setup\_grafana.sh.** Automatiza la instalación de Grafana, habilita el modo anónimo y crea un token de autenticación mediante un Service Account. El script es especialmente estable, incluyendo comprobaciones de salud, reintentos y validaciones en cada paso aguantando desfases que pueden ser provocados por equipos muy lentos o muy rápidos.

**Script 05\_setup\_python\_env.sh.** Crea un entorno virtual de Python donde se instalan dependencias como FastAPI, PyMySQL, requests y psutil. Esto aísla el entorno de ejecución de posibles conflictos con paquetes del sistema.

**Script 06\_install\_services.sh.** Copia los scripts Python al destino final y registra tres servicios principales:

- **rsnort-api.service.** Expone una API REST basada en FastAPI para interactuar con el agente.
- **rsnort-ingest.service.** Lee continuamente el archivo `alert_json.txt` y vuelca su contenido en la base de datos.
- **rsnort-metrics.timer.** Ejecuta un script cada 30 segundos para capturar métricas del sistema.

**Script 07\_import\_dashboard.sh.** Importa automáticamente un dashboard preconfigurado en Grafana, lo vincula con el datasource Snort–MariaDB y proporciona una visualización inmediata y profesional de alertas y métricas.

#### Scripts Python: arquitectura y lógica interna

- **agent\_api.py.** Implementa endpoints REST para acceder a alertas, métricas, reglas activas, archivos archivados y gestión de servicios. Incluye validación sintáctica de reglas antes de ser añadidas.
- **ingest\_service.py.** Monitoriza el archivo `alert_json.txt` detectando rotaciones y evita duplicados usando un hash de línea.
- **metrics\_timer.py.** Calcula la carga de CPU, uso de RAM, espacio en disco y temperatura media del sistema. Esta última es estimada si los sensores no están disponibles.

```
1 def insert_alert(rec):
2     ts_original = rec.get("timestamp", "")
3     ts_normalizado = normalize_ts(ts_original)
4     rec["timestamp"] = ts_normalizado
5     ...
6     cur.execute("""
7         INSERT INTO alerts (...),
8             VALUES (%s, ..., %s)
9         """ , valores + [AGENT_ID])
```

Código 2.13: Fragmento de inserción de alerta en ingest\_service.py

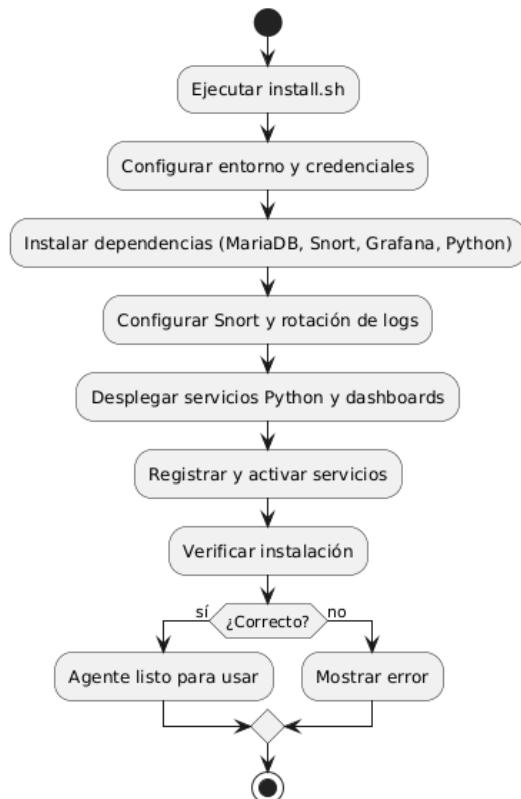


Figura 2.24: Proceso detallado de instalación del agente R-Snort.

### 2.6.2. Instalador del módulo central: **r-snort-central-module**

El módulo central constituye el componente principal del proyecto, este se encargar de llevar a cabo la gestión y visualización del sistema R-Snort con todo lo que ello conlleva. Permite consolidar alertas y métricas provenientes de múltiples agentes, así como gestionar reglas, usuarios y agentes desde una interfaz web integrada. Este componente fusiona un backend en Spring Boot con un frontend en Angular, y se despliega como un servicio `systemd`.

#### Iniciación

- `00_common.sh`. Define rutas comunes, credenciales de base de datos y funciones auxiliares como la detección de IP local. Estos valores se reutilizan en todos los demás scripts.
- `01_dependencies.sh`. Instala dependencias clave del sistema:
  - **Java JDK 17 y Maven**, necesarios para compilar el backend.
  - **Node.js 18 y Angular CLI**, requeridos para construir el frontend.
  - Herramientas auxiliares como `git`, `curl` y `apache2-utils`.

#### Compilación

- `02_compile_frontend.sh`. Inyecta automáticamente la IP local del backend en los archivos de entorno de Angular, compila la aplicación en modo producción y copia los archivos resultantes al directorio estático del backend para que se empaqueten en el JAR final.

- `03_compile_backend.sh`. Compila el backend con Maven (o su wrapper si está disponible), generando un archivo `rsnort.jar` que incluye también el frontend. Este archivo se copiará posteriormente a `/opt/rsnort-backend/`.

## Configuración de base de datos

- `04_prepare_db.sh`. Garantiza la existencia de la tabla `users` en la base de datos, la cual almacena credenciales, roles y estados de activación de los usuarios que acceden a la interfaz web.
- `05_add_admin_user.sh`. Sigue por terminal el correo y contraseña del primer administrador, genera un hash `bcrypt` con `htpasswd`, e inserta el usuario directamente en la tabla `users`.

## Despliegue

- `06_setup_agents.sh`. Permite definir los agentes registrados en el sistema. Se crea el archivo `agents.json` en `/var/lib/rsnort-backend/`, incluyendo el agente central (automático) y cualquier otro agente remoto adicional que se desee agregar.
- `07_install_services.sh`. Este script empaqueta y despliega el sistema como servicio:
  - Copia el JAR generado a `/opt/rsnort-backend/`.
  - Define variables de entorno tanto para `systemd` como para shells interactivos.
  - Crea un servicio `rsnort-backend.service` que lanza el backend con Java y asegura su persistencia.

```
1 [Service]
2 ExecStart=/usr/bin/java -jar /opt/rsnort-backend/rsnort.jar
3 EnvironmentFile=/etc/rsnort-backend/rsnort.env
4 Restart=on-failure
5 ProtectSystem=full
6 PrivateTmp=true
```

Código 2.14: Fragmento del servicio `systemd` para el backend

**Script `run_all.sh`:** Pone en marcha todos los pasos anteriores con mensajes en color y ejecución secuencial para que el usuario final sea capaz de visualizar el estado de la instalación. Ejecuta cada script con los permisos adecuados (algunos como usuario regular, otros como root) y resume el proceso al finalizar proporcionando instrucciones de acceso.

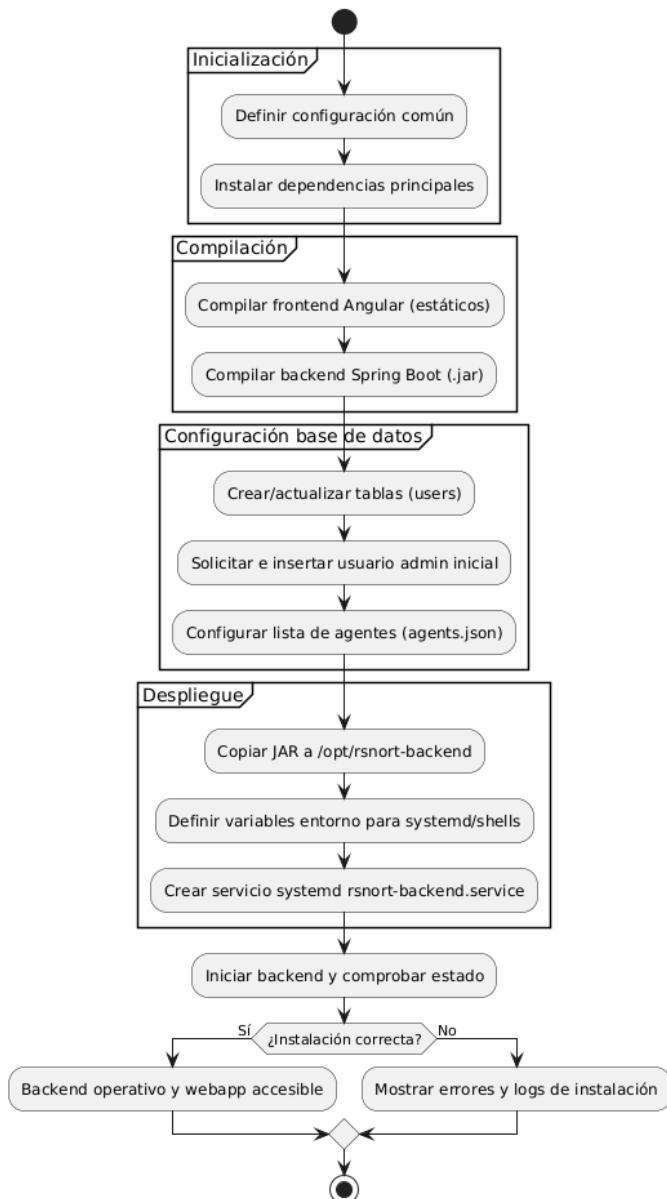


Figura 2.25: Proceso escalonado de instalación del módulo central de R-Snort.

Este instalador permite desplegar un sistema de visualización y control, que centraliza alertas y datos de múltiples agentes, ofrece una interfaz segura para la administración de reglas y usuarios, y facilita el mantenimiento y la extensión futura del sistema R-Snort.



### 3. Caso práctico: utilización del frontend de R-Snort

El sistema R-Snort se ha desplegado y evaluado en un entorno realista orientado a redes domésticas avanzadas o pequeñas oficinas (SOHO), simulando un entorno distribuido de múltiples subredes monitorizadas mediante instancias independientes de Snort 3 en Raspberry Pi 5.

#### 3.1. Entorno de trabajo

Este ecosistema puede funcionar tanto en modo **mononodo** como en modo **multinodo**, según las necesidades del entorno. En la modalidad mononodo, se instala únicamente el módulo central, que además de ofrecer la interfaz web y coordinar la lógica del sistema, ejecuta también Snort para vigilar el segmento de red local en el que se encuentra. Esta configuración es ideal para entornos más simples, como puede verse en la Figura 3.1.

Por otro lado, en entornos más complejos o distribuidos, R-Snort permite el despliegue en modo multinodo, donde se instalan agentes adicionales en diferentes segmentos de red. Cada uno de estos agentes monitoriza su propia subred y se comunica con el módulo central, que unifica las alertas, métricas y reglas en una única consola de gestión. Esta arquitectura escalable, ilustrada en la Figura 3.2, permite una cobertura más completa de la red y facilita la administración desde un punto central.

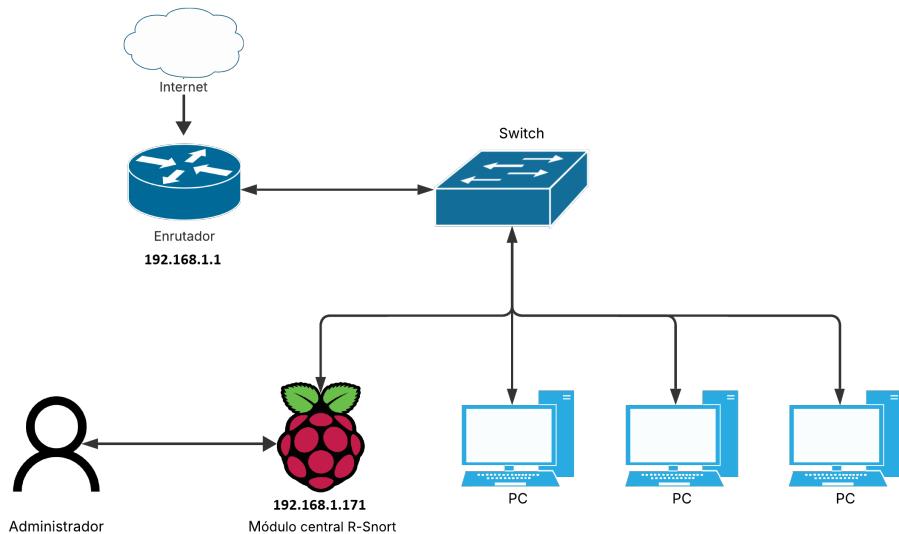


Figura 3.1: Funcionamiento del sistema R-Snort en modo mononodo.

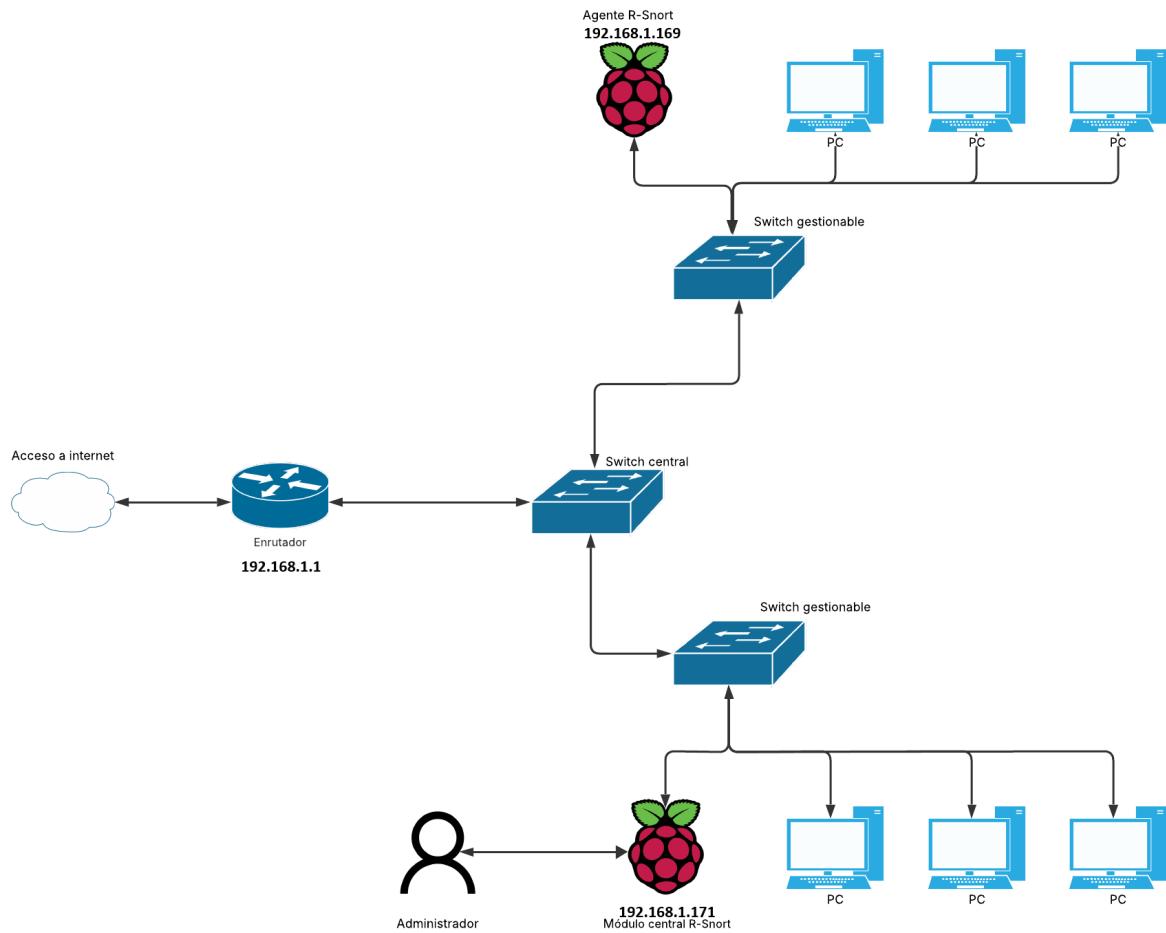


Figura 3.2: Esquema lógico de red con múltiples agentes Snort y módulo central R-Snort.

El objetivo de esta distribución es comprobar la eficacia del proyecto en un entorno lo más cercano a una red PYME o SOHO, por ello la distribución final utilizada es la multinodo. El agente se encarga de recoger las alertas de su subred asignada a través del port mirroring que copiará todo el tráfico de los demás puertos hacia el agente, de la misma manera, el módulo central recoge todo el tráfico de su correspondiente switch usando la misma técnica con la funcionalidad añadida de que el módulo central cuenta con el frontend previamente instalado que va a permitir cubrir toda la red de vigilancia.

Aunque en la práctica solo se ha empleado un agente Snort por limitaciones de infraestructura (dos si contamos el módulo central), el diseño y la implementación del sistema permiten escalar horizontalmente con facilidad, añadiendo nuevos agentes usando el instalador automático de snort-agent, posteriormente agregando la IP del nuevo agente mediante la interfaz del módulo central.

El hardware empleado para el despliegue y validación del sistema R-Snort incluye los siguientes dispositivos y componentes de red:

- **Raspberry Pi 5 (8 GB de RAM)** con Ubuntu Server 22.04 LTS (aarch64). Esta placa actúa como *módulo central* de R-Snort, alojando:
  - Snort 3.1.84.0 compilado con Hyperscan.

- `rsnort-backend` (Spring Boot) y `rsnort-frontend` (Angular).
- Base de datos MariaDB para alertas, métricas y configuración.
- Grafana 12 para dashboards de rendimiento, alertas y uso de recursos.
- Servicios systemd: `rsnort-backend.service`.

Se conecta a un switch gestionable mediante un adaptador USB 3.0 → Gigabit para garantizar ancho de banda completo y latencia mínima en la almacenamiento de métricas.

- **Raspberry Pi 3 Model B (1 GB de RAM)** con Ubuntu Server 22.04 LTS. Funciona como *agente Snort*, ejecutando:

- Snort 3.1.84.0 compilado con Hyperscan.
- Servicios systemd: `rsnort-ingest`, `rsnort-api`, `rsnort-metrics`.
- Configuración optimizada (NUMA deshabilitado, perfil ligero) para adaptarse a recursos limitados.

Está conectado al puerto mirror de un switch gestionable Tenda TEG205E (5 puertos Gigabit), que replica el tráfico de la subred al sensor.

- **Switch gestionable Tenda TEG205E** con port mirroring activado, replicando todo el tráfico ethernet de cada subred hacia el agente correspondiente sin afectar al tráfico productivo.
- **Dispositivos cliente heterogéneos** (Windows 11, Ubuntu Desktop), que generan tráfico de usuario real y ataques de prueba (p. ej. nmap, peticiones DNS, HTTP).
- **PC con Ubuntu Server 22.04 LTS (x86\_64)**. Solo se utiliza para pruebas puntuales (simulación de escenarios, análisis offline de logs, generación de tráfico controlado) y no forma parte del despliegue productivo.

Todos los componentes de R-Snort se comunican vía HTTP REST API en el puerto 9000, intercambiando alertas y métricas con la base de datos central. Grafana 12, instalado en la Raspberry Pi 5, se conecta directamente a MariaDB para ofrecer dashboards interactivos de alertas, consumo de CPU/RAM y mapas de tráfico en tiempo real.

### 3.2. Instalación del sistema

El sistema R-Snort (abreviatura de *Raspberry-Snort*) ha sido concebido como una solución ligera, modular y automatizada para proteger redes pequeñas o entornos de baja potencia, como los basados en dispositivos ARM (Raspberry Pi). Todos sus componentes han sido diseñados para ejecutarse sin dificultad en hardware con recursos limitados, sin renunciar a funcionalidades avanzadas de inspección y monitorización.

Aunque R-Snort es plenamente compatible con cualquier sistema Ubuntu/Debian de arquitectura x86\_64 o ARM64, la elección de Raspberry Pi como plataforma principal responde al objetivo de validar su comportamiento en contextos realistas de bajo consumo, alta eficiencia

energética y presupuesto reducido.

El sistema completo se compone de tres grandes bloques:

- **Instalador del complemento del TFG (R-Snort3).** Encargado de desplegar automáticamente una instancia personalizada y optimizada de Snort versión 3, incluyendo todas las dependencias, configuraciones y adaptaciones necesarias para PYMES y redes SOHO.
- **Módulo de agente (snort-agent).** Permite la captura y gestión local de alertas y métricas del sistema. Se comunica con el módulo central mediante una API REST y transmite datos a una base de datos que es leída por Grafana mediante un dashboard personalizado y adaptado a alertas de Snort 3 y métricas de sistemas Linux.
- **Módulo central (rsnort-central-module).** Centraliza la gestión del sistema, permitiendo desde una interfaz web el control de reglas, la visualización de alertas, la integración con Grafana y la supervisión de múltiples agentes distribuidos.

En el contexto de este Trabajo Fin de Grado, el desarrollo se ha centrado principalmente en los dos últimos módulos, es decir, en la creación de una interfaz web completa, distribuida y funcional mediante Spring Boot (backend) y Angular (frontend), que permita interactuar con los agentes y gestionar Snort de forma remota.

No obstante, para aprovechar al máximo las capacidades del sistema, se ha optado por utilizar la instancia personalizada de Snort 3 generada por el complemento del TFG, en lugar de emplear la versión genérica del IDS. Esta elección permite reducir el consumo de recursos, aumentar la compatibilidad con entornos ARM y facilitar la integración con los módulos del sistema.

**Por tanto, como requisito previo a la instalación de los agentes, se asume que el sistema ya dispone de Snort 3 instalado correctamente**, ya sea mediante el complemento R-Snort3 o a través de una instalación manual compatible. Las instrucciones siguientes parten de esa premisa.

**Tabla 3.0: Requisitos del sistema para el despliegue de R-Snort**

Tipo de requisito	Descripción
<b>R1 - Requisito funcional</b>	El sistema debe disponer de Snort 3 previamente instalado y configurado correctamente.
<b>R2 - Requisito funcional</b>	El agente debe enviar alertas y métricas al módulo central mediante API REST.
<b>R3 - Requisito funcional</b>	El módulo central debe permitir visualizar alertas, editar reglas y gestionar agentes desde la web.
<b>R4 - Requisito funcional</b>	Los logs deben poder rotarse y almacenarse automáticamente para fines forenses.
<b>R5 - Requisito no funcional</b>	El sistema debe ejecutarse correctamente en hardware de bajo consumo como Raspberry Pi 3 o 5.
<b>R6 - Requisito no funcional</b>	La instalación debe estar automatizada y requerir intervención mínima del usuario.
<b>R7 - Requisito no funcional</b>	El sistema debe integrarse con MariaDB y Grafana sin configuraciones manuales complejas.
<b>R8 - Requisito no funcional</b>	La interfaz web debe ser accesible desde cualquier navegador moderno sin necesidad de complementos.

Tabla 3.1: Requisitos funcionales y no funcionales previos al despliegue del sistema R-Snort

### 3.2.1. Preparación de la instalación

#### Instalación del sistema operativo en las Raspberry Pi

Antes de comenzar con la instalación del ecosistema R-Snort, es importante preparar adecuadamente el entorno de trabajo. Como se ha explicado previamente, R-Snort está pensado para ejecutarse en dispositivos de bajo consumo como las Raspberry Pi, aunque también es compatible con cualquier sistema Ubuntu/Debian.

El primer paso consiste en instalar el sistema operativo en los dispositivos que actuarán como módulo central o como agentes de Snort. Para ello se utiliza **Raspberry Pi Imager** [28], una herramienta oficial y sencilla que permite seleccionar el modelo de Raspberry Pi, elegir la distribución de Linux deseada e instalarla automáticamente en una tarjeta microSD.

En este caso, el sistema operativo elegido ha sido Ubuntu Server 25.04 (64-bit), que ofrece un equilibrio excelente entre rendimiento, estabilidad y compatibilidad con las herramientas utilizadas por R-Snort.

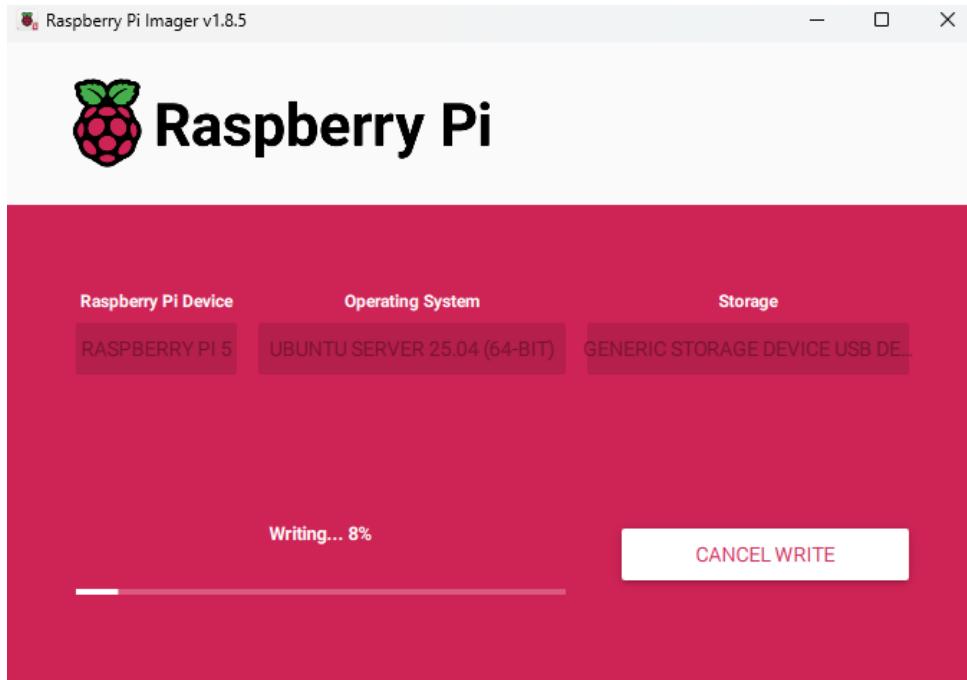


Figura 3.3: Proceso de escritura de Ubuntu Server 25.04 (64-bit) en una Raspberry Pi 5 mediante Raspberry Pi Imager.

### Conexiones físicas de la Raspberry Pi 5

Una vez instalado el sistema operativo tanto en la **Raspberry Pi 5** (que actuará como módulo central) como en la **Raspberry Pi 3 Model B** (que funcionará como agente Snort), se procede a realizar las conexiones de red necesarias para cada una de ellas.

En el caso de la Raspberry Pi 5, se conectan **dos interfaces de red físicas**. Esta decisión responde a la necesidad de separar el tráfico de administración del tráfico de monitorización. Por una parte, la interfaz principal `eth0` proporciona conectividad a internet y se utiliza para el control general del sistema. Por otra, se añade una segunda interfaz de red mediante adaptador USB Ethernet, configurada como **interfaz promiscua** y renombrada automáticamente como `enxc8a362b4a702`.

Esta segunda interfaz está especialmente diseñada para capturar todo el tráfico de red redirigido mediante *port mirroring* desde el switch gestionable. Por razones de seguridad, el script de instalación de la instancia personalizada de Snort 3 desarrollada en el complemento del TFG desactiva cualquier dirección IP en esta interfaz, dejándola exclusivamente dedicada a la escucha pasiva.



Figura 3.4: Conexión física de la Raspberry Pi 5 como módulo central.

### Configuración del switch y port mirroring

En el switch gestionable Tenda, se han establecido las conexiones necesarias para habilitar la monitorización del tráfico de red. Por un lado, se conecta el router doméstico al puerto 1, lo que proporciona conectividad a internet a todos los dispositivos conectados al switch. Por otro lado, los puertos 2 y 3 se utilizan para conectar distintos dispositivos de la red local que generarán tráfico.

La interfaz en modo promiscuo de la Raspberry Pi 5 se conecta al puerto 4 del switch mediante un adaptador USB a Ethernet. Esta interfaz será la encargada de recibir todo el tráfico replicado desde los demás puertos del switch.

Para lograr este comportamiento, se configura la opción de *port mirroring* desde la interfaz web del switch. Se establece el puerto 4 como destino del mirroring, y como fuente todos los puertos del 1 al 3, en ambas direcciones. Esta configuración garantiza que todo el tráfico entrante y saliente de los dispositivos conectados sea duplicado y enviado a la Raspberry Pi 5 para su análisis con Snort 3.

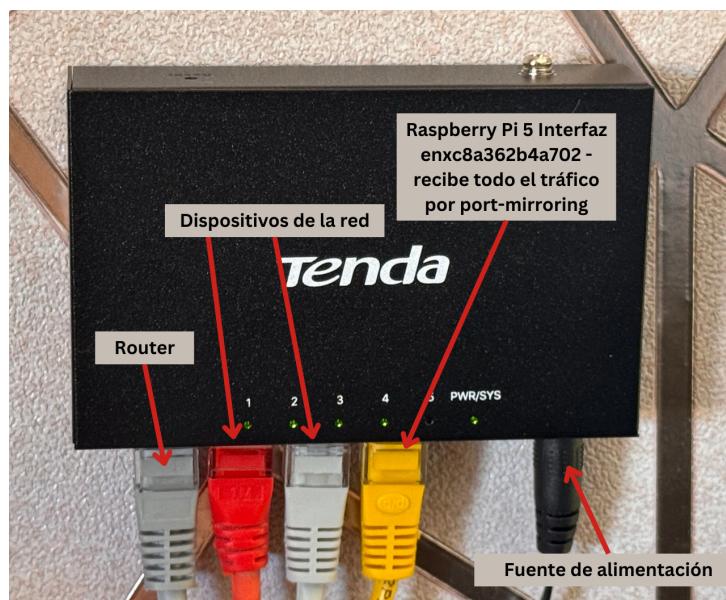


Figura 3.5: Conexiones del switch.

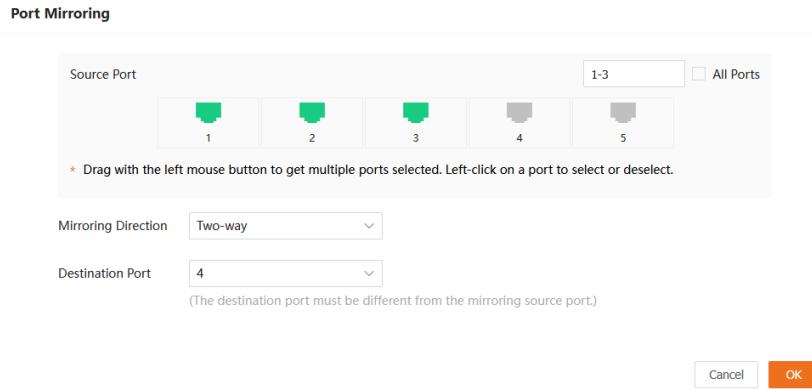


Figura 3.6: Configuración del mirroring.

### Conexión del agente en la subred secundaria

La subred secundaria es gestionada por una **Raspberry Pi 3 Model B**, que actúa como agente Snort. Debido a las limitaciones del hardware, solo dispone de una interfaz de red física (`eth0`), la cual será utilizada simultáneamente como interfaz promiscua y como salida a internet.

Aunque esta práctica no es recomendable en entornos de producción —donde se sugiere eliminar la dirección IP de la interfaz dedicada a la captura para evitar accesos no deseados— en este caso se ha considerado aceptable al tratarse de un entorno de pruebas controlado.

El instalador del complemento de Snort 3 personalizado para PYMEs está diseñado para manejar estas situaciones. En sistemas con múltiples interfaces, pregunta al usuario si desea borrar la IP de la interfaz promiscua, evitando así riesgos innecesarios. Sin embargo, si solo se detecta una interfaz, como en esta Raspberry Pi 3, la lógica del script omite esa pregunta para prevenir una pérdida accidental de conectividad.

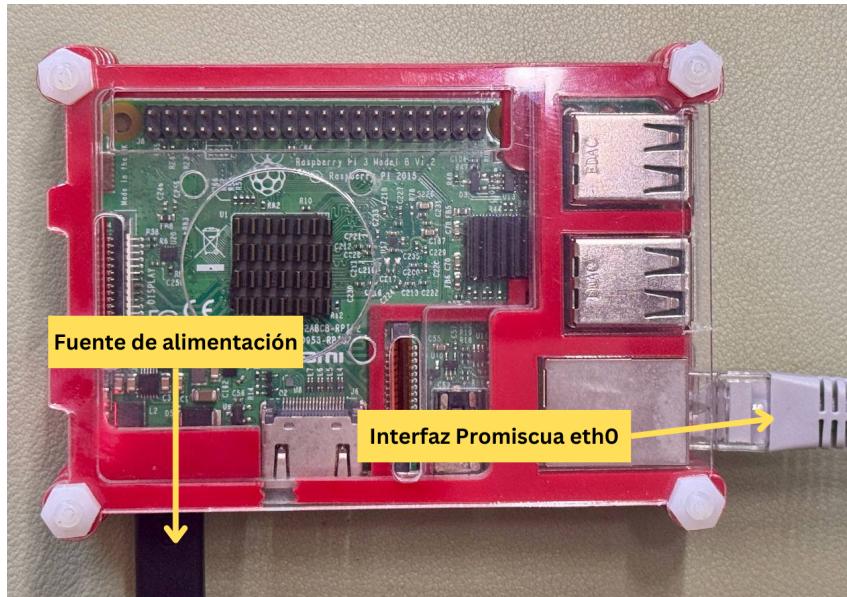


Figura 3.7: Raspberry Pi 3 actuando como agente.

### Conexión del switch de la subred secundaria

En la subred secundaria, el tráfico de red es gestionado por un segundo switch gestionable. Este switch recibe conexión directa al router para proporcionar acceso a internet a los dispositivos conectados.

La **Raspberry Pi 3 Model B**, que actúa como agente Snort, está conectada al puerto configurado para port mirroring. Su interfaz `eth0`, operando en modo promiscuo, captura todo el tráfico generado por los dispositivos de la red que están conectados al resto de puertos del switch.

Esta arquitectura permite extender la monitorización a múltiples subredes, manteniendo una estructura modular y escalable dentro del sistema R-Snort.

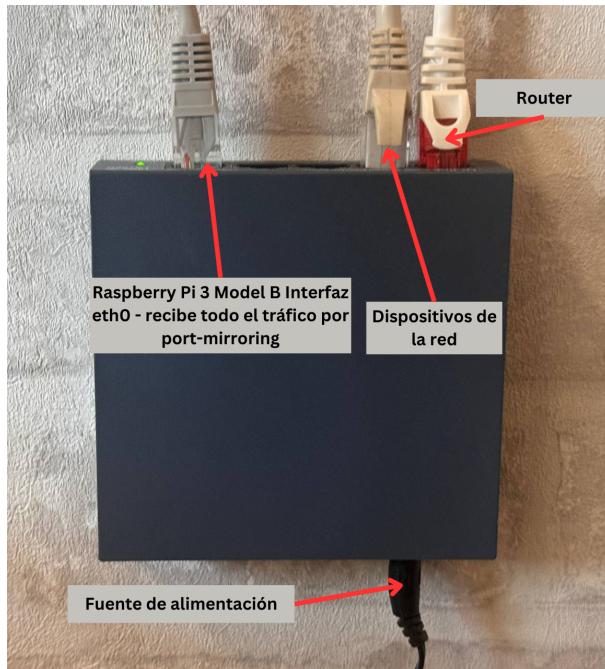


Figura 3.8: Conexiones del switch secundario.

### Instalación de Snort 3 personalizado

Para completar la preparación del sistema, es necesario instalar Snort versión 3 en ambas Raspberry Pi. Tal como se ha comentado previamente, se utilizará la instancia personalizada desarrollada como complemento de este Trabajo Fin de Grado, optimizada para entornos SOHO y compatible con arquitecturas ARM.

Esta versión incluye todas las dependencias necesarias, configuraciones de rendimiento ajustadas y soporte para bibliotecas como Hyperscan, eliminando la necesidad de instalación manual o intervención compleja por parte del usuario.

El proceso de instalación es completamente automático. Basta con clonar el repositorio correspondiente y ejecutar el script principal:

```
git clone https://github.com/deianp189/r-snort-installer.git
2   cd r-snort-installer
    sudo ./install_rsnort.sh
```

Código 3.1: Instalación de R-Snort personalizada

```
● snort-lab@r-snort:~$ git clone https://github.com/deianp189/r-snort-installer
Cloning into 'r-snort-installer'...
remote: Enumerating objects: 137, done.
remote: Counting objects: 100% (2/2), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 137 (delta 0), reused 0 (delta 0), pack-reused 135 (from 1)
Receiving objects: 100% (137/137), 74.28 MiB | 35.72 MiB/s, done.
Resolving deltas: 100% (42/42), done.
● snort-lab@r-snort:~$ cd r-snort-installer/
○ snort-lab@r-snort:~/r-snort-installer$ sudo ./install_rsnort.sh
=====
[!] Instalación R-SNORT: Wed May 21 15:06:47 CEST 2025
=====
[!] [R-SNORT] Actualizando lista de paquetes...
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Hit:1 http://ports.ubuntu.com/ubuntu-ports plucky InRelease
Hit:2 http://ports.ubuntu.com/ubuntu-ports plucky-updates InRelease
Hit:3 http://ports.ubuntu.com/ubuntu-ports plucky-backports InRelease
Get:4 http://ports.ubuntu.com/ubuntu-ports plucky-security InRelease [126 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports plucky-security/main arm64 Packages [32.4 kB]
Get:6 http://ports.ubuntu.com/ubuntu-ports plucky-security/main Translation-en [12.2 kB]
Get:7 http://ports.ubuntu.com/ubuntu-ports plucky-security/universe arm64 Packages [29.2 kB]
Get:8 http://ports.ubuntu.com/ubuntu-ports plucky-security/universe Translation-en [9836 B]
```

Figura 3.9: Inicio de la instalación del agente.

Tras unos 15 minutos en la Raspberry Pi 5, el sistema tendrá Snort 3 instalado, configurado y activo sobre la interfaz elegida por el usuario. Al finalizar, el script ofrece un resumen detallado del estado del sistema, incluyendo información sobre recursos, antivirus y adaptadores de red.

```
[✓] Servicio Snort configurado y activo.
[*] Configurador de Snort ejecutado correctamente.
[*] Comprobando estado del servicio Snort...
[*] Snort está activo y habilitado.

[*] Resumen del sistema tras la instalación:

[?] Hostname:          r-snort
[?] Uptime:            up 1 hour, 29 minutes
[?] RAM usada:         2.0Gi / 7.7Gi
[?] Swap activa:       No
[?] Espacio raíz:     4.8G usados de 117G
[?] CPU:               Cortex-A76 (4 núcleos)
[?] Snort versión:    3.1.84.0
[?] ClamAV versión:   1.4.2/27644/Wed
[?] Interfaz activa:  enxc8a362b4a702

[✓] Snort 3 está en ejecución en la interfaz: enxc8a362b4a702.
[✓] Instalación de R-Snort completada con éxito.
snort-lab@r-snort:~/r-snort-installer$
```

Figura 3.10: Resumen tras la instalación.

### 3.2.2. Instalación del agente R-Snort

La instalación del agente R-Snort se realiza mediante un script completamente automatizado que integra y configura todos los componentes necesarios. Aunque previamente se ha detallado cómo se transforma Snort versión 3 en un sistema funcional mediante el complemento del TFG, este instalador encapsula todo el proceso en una ejecución sencilla y eficiente.

El script realiza las siguientes tareas de forma automática:

- Configura Snort 3 para almacenar sus alertas en un archivo específico.
- Lanza los servicios de ingestión y métrica que recogen estas alertas y las insertan en una base de datos MariaDB.
- Despliega una API REST basada en FastAPI, que permite al módulo central acceder a las alertas y métricas del agente.
- Instala y configura Grafana 12, incluyendo un dashboard personalizado que se importa automáticamente.
- Registra todos los servicios con systemd para que arranquen de forma persistente.

Una vez instalado, el usuario no necesita realizar configuraciones adicionales. El propio sistema indica al final de la instalación las URLs desde las que se puede acceder a la API REST y al panel de Grafana.

El proceso comienza con los siguientes comandos:

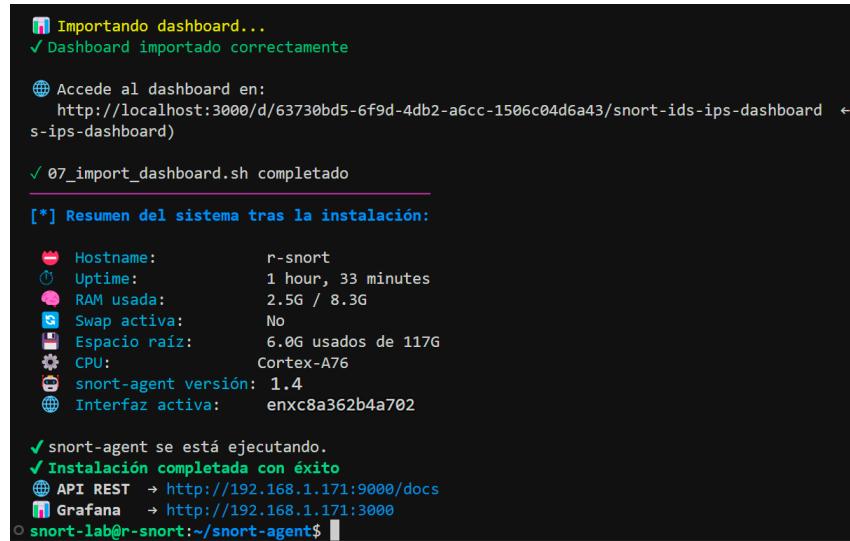
```
1 git clone https://github.com/deianp189/snort-agent.git
2 cd snort-agent
3 sudo ./install.sh
```

Código 3.2: Comandos de instalación del agente

```
● snort-lab@r-snort:~$ git clone https://github.com/deianp189/snort-agent.git
Cloning into 'snort-agent'...
remote: Enumerating objects: 138, done.
remote: Counting objects: 100% (138/138), done.
remote: Compressing objects: 100% (98/98), done.
remote: Total 138 (delta 72), reused 105 (delta 39), pack-reused 0 (from 0)
Receiving objects: 100% (138/138), 56.32 KiB | 1.15 MiB/s, done.
Resolving deltas: 100% (72/72), done.
● snort-lab@r-snort:~$ cd snort-agent/
● snort-lab@r-snort:~/snort-agent$ sudo ./install.sh
▶ Ejecutando 01_install_db.sh
✓ Añadido source /etc/rsnort-agent/env.sh a /home/snort-lab/.bashrc
⚡ Configuración completada.
✓ Archivo de conexión:      /etc/rsnort-agent/db.cnf
✓ Entorno para shell:        /etc/rsnort-agent/env.sh
✓ Entorno para systemd:     /etc/rsnort-agent/env.service
✓ Variables actuales:
  RSNORT_DB_USERNAME=rsnort
  RSNORT_DB_PASSWORD=cambio_me
  RSNORT_DB_NAME=rsnort_agent
Hit:1 http://ports.ubuntu.com/ubuntu-ports plucky InRelease
Hit:2 http://ports.ubuntu.com/ubuntu-ports plucky-updates InRelease
Hit:3 http://ports.ubuntu.com/ubuntu-ports plucky-backports InRelease
Hit:4 http://ports.ubuntu.com/ubuntu-ports plucky-security InRelease
Reading package lists... 69%
```

Figura 3.11: Inicio de la instalación del agente.

La duración estimada es de entre 5 y 8 minutos, dependiendo de la conexión a internet. El usuario simplemente debe esperar mientras el sistema realiza toda la configuración necesaria.



```

[!] Importando dashboard...
✓ Dashboard importado correctamente

🌐 Accede al dashboard en:
  http://localhost:3000/d/63730bd5-6f9d-4db2-a6cc-1506c04d6a43/snort-ids-ips-dashboard ←
  s-ips-dashboard)

✓ 07_import_dashboard.sh completado

[*] Resumen del sistema tras la instalación:

  📡 Hostname:          r-snort
  ⏳ Uptime:            1 hour, 33 minutes
  💡 RAM usada:        2.5G / 8.3G
  💾 Swap activa:      No
  📂 Espacio raíz:    6.0G usados de 117G
  🖥️ CPU:              Cortex-A76
  🚀 snort-agent versión: 1.4
  🌐 Interfaz activa:  enxc8a362b4a702

✓ snort-agent se está ejecutando.
✓ Instalación completada con éxito
🌐 API REST → http://192.168.1.171:9000/docs
🌐 Grafana → http://192.168.1.171:3000
● snort-lab@r-snort:~/snort-agent$ 

```

Figura 3.12: Finalización y resumen del agente instalado.

### 3.2.3. Instalación del módulo central de R-Snort

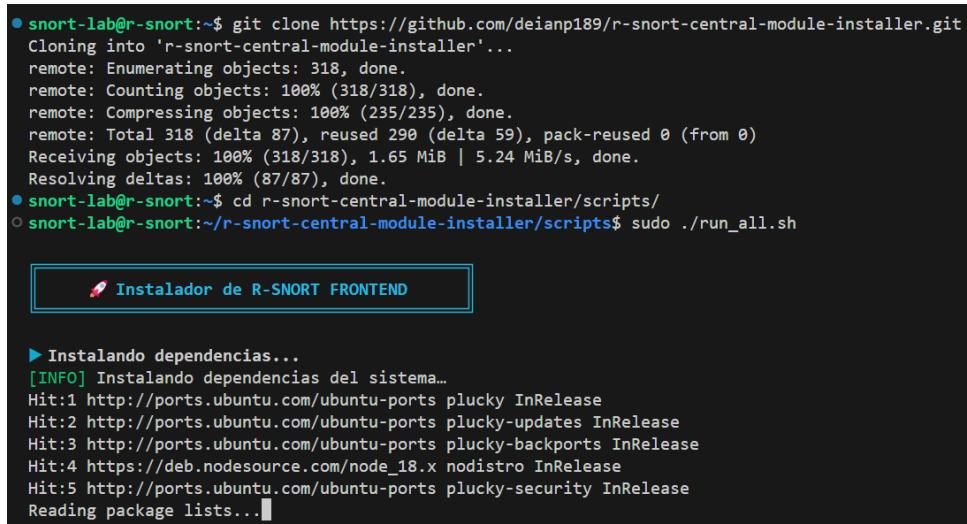
Una vez instalado el agente, puede transformarse en el **módulo central de R-Snort** ejecutando un segundo instalador. Este script despliega el sistema completo de gestión: compila el frontend desarrollado en Angular, lo integra dentro del backend en Spring Boot y lo configura como servicio persistente mediante `systemd`, asegurando su ejecución tras cada reinicio del sistema.

```

1 git clone https://github.com/deianp189/r-snort-central-module-installer.git
2 cd r-snort-central-module-installer/scripts
3 sudo ./run_all

```

Código 3.3: Comandos de instalación del agente



🚀 Instalador de R-SNORT FRONTEND

```

● snort-lab@r-snort:~$ git clone https://github.com/deianp189/r-snort-central-module-installer.git
Cloning into 'r-snort-central-module-installer'...
remote: Enumerating objects: 318, done.
remote: Counting objects: 100% (318/318), done.
remote: Compressing objects: 100% (235/235), done.
remote: Total 318 (delta 87), reused 290 (delta 59), pack-reused 0 (from 0)
Receiving objects: 100% (318/318), 1.65 MiB | 5.24 MiB/s, done.
Resolving deltas: 100% (87/87), done.
● snort-lab@r-snort:~$ cd r-snort-central-module-installer/scripts/
○ snort-lab@r-snort:~/r-snort-central-module-installer/scripts$ sudo ./run_all.sh

▶ Instalando dependencias...
[INFO] Instalando dependencias del sistema...
Hit:1 http://ports.ubuntu.com/ubuntu-ports plucky InRelease
Hit:2 http://ports.ubuntu.com/ubuntu-ports plucky-updates InRelease
Hit:3 http://ports.ubuntu.com/ubuntu-ports plucky-backports InRelease
Hit:4 https://deb.nodesource.com/node_18.x nodistro InRelease
Hit:5 http://ports.ubuntu.com/ubuntu-ports plucky-security InRelease
Reading package lists...

```

Figura 3.13: Inicio de la instalación del módulo central.

Durante la instalación, el usuario solo debe indicar sus credenciales de administrador y decidir si desea declarar más agentes manualmente. En este caso se optó por omitir esa configuración, ya que la interfaz web permite añadirlos posteriormente, lo cual resulta más cómodo y sirve para demostrar la funcionalidad del sistema.

El proceso dura aproximadamente cinco minutos y finaliza mostrando un resumen detallado del entorno, así como la dirección web desde la cual puede accederse al panel de administración.

```
▶ Ejecutando 05_add_admin_user.sh
Correo para ADMIN: dp189@inlumine.ual.es
Contraseña:
Confirmar:
[INFO] ✓ Usuario ADMIN registrado.
✓ 05_add_admin_user.sh completado

▶ Ejecutando 06_setup_agents.sh
[INFO] ✓ Agente central añadido automáticamente: central (192.168.1.171)
¿Añadir otro agente? (y/N) n
[INFO] ✓ Archivo /var/lib/rsnort-backend/agents.json creado con 1 agente(s):
    - central → 192.168.1.171
✓ 06_setup_agents.sh completado

▶ Ejecutando 07_install_service.sh
Created symlink '/etc/systemd/system/multi-user.target.wants/rsnort-backend.service' → '/etc/systemd/system/rsnort-backend.service'.
[INFO] ✓ Servicio rsnort-backend activo.
✓ 07_install_service.sh completado

[*] Resumen del sistema tras la instalación:

Hostname:          r-snort
Uptime:            2 hours, 3 minutes
RAM usada:         2.5G / 8.3G
Espacio raíz:     7.7G usados de 117G
CPU:               Cortex-A76
Node.js versión:   v18.20.8
Java versión:      17.0.15
Interfaz activa:   enxc8a362b4a702

✓ Instalación completada con éxito
🌐 Accede ahora a: http://192.168.1.171:8080
```

Figura 3.14: Finalización y resumen del módulo central.

### 3.3. Utilización y pruebas

#### 3.3.1. Configuración

Tras completar el proceso de instalación, el acceso a la interfaz web de R-Snort se realiza fácilmente introduciendo en el navegador la dirección `http://<IP-del-módulo-central>:8080` (También se muestra con exactitud al finalizar la instalación). La autenticación se efectúa mediante las credenciales proporcionadas por el usuario durante la instalación del módulo central.

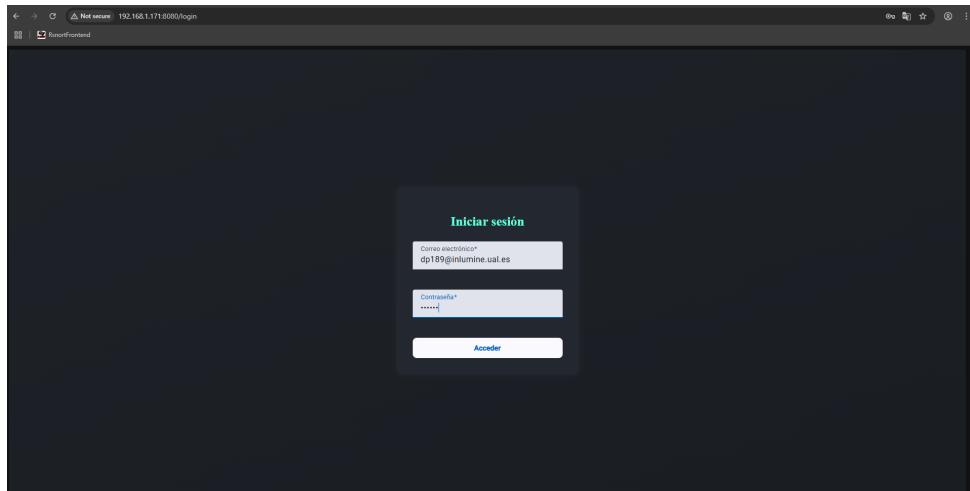


Figura 3.15: Pantalla de inicio de sesión del sistema R-Snort.

Una vez autenticado, se accede a la pestaña **Overview**, que presenta un panel general con gráficas dinámicas alimentadas por Grafana. Estas gráficas se generan a partir de los datos almacenados en MariaDB y reflejan información escogida con sumo cuidado como: número de alertas por severidad (alta, media y baja), protocolos implicados, tipos de alertas más frecuentes, y direcciones IP origen. Todo ello permite una visualización rápida y efectiva del estado de la red.



Figura 3.16: Panel general con métricas y alertas.

En la pestaña **Alertas**, el usuario puede inspeccionar de forma detallada las últimas alertas generadas, agrupadas por severidad. Esta vista incluye un panel de resumen y una tabla con todas las alertas clasificadas por fecha, origen, destino y tipo de amenaza detectada. Además, se ofrece la opción de descargar alertas individuales por agente o en su totalidad para un análisis forense posterior.



Figura 3.17: Panel de alertas clasificadas por severidad.

La pestaña **Reglas** muestra todas las reglas actualmente activas del archivo `custom.rules`, incluyendo aquellas diseñadas para detectar fuga de datos sensibles como correos electrónicos, tarjetas de crédito o números NUSS. El sistema permite añadir nuevas reglas y elimina automáticamente aquellas duplicadas o malformadas, previniendo fallos de carga en Snort.

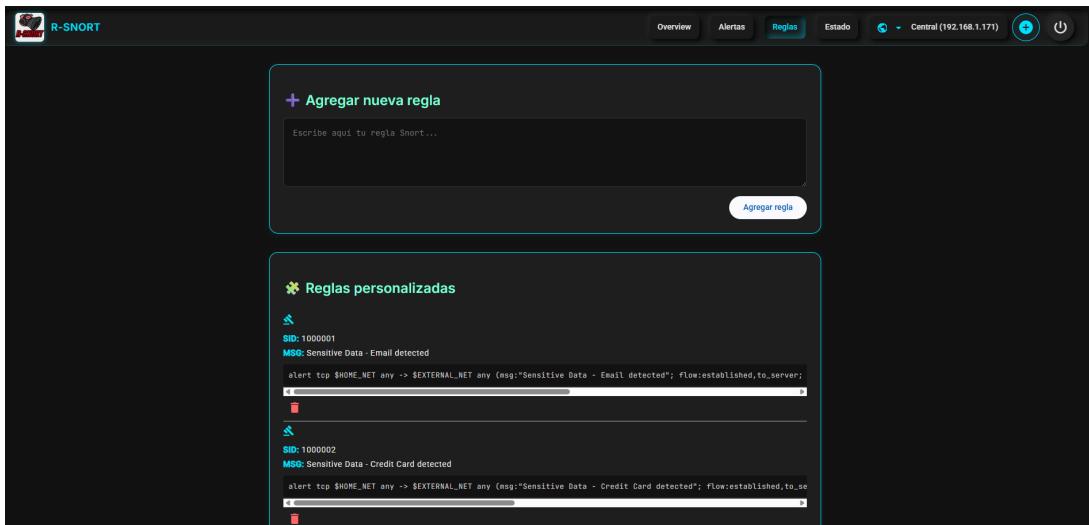


Figura 3.18: Gestión de reglas Snort desde la interfaz.

En la pestaña **Estado**, se presentan datos sobre el uso de recursos del sistema (CPU, disco, temperatura) a lo largo del tiempo, gracias a la integración con Grafana. Además, se muestra el estado actual de Snort, la última alerta capturada y la posibilidad de reiniciar los servicios involucrados (snort, ingest, metrics).

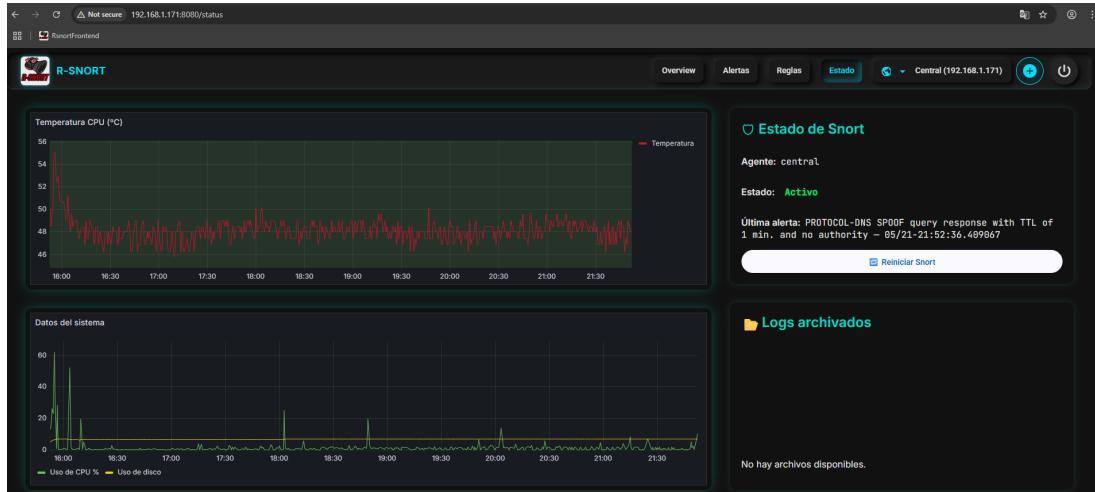


Figura 3.19: Monitorización de temperatura, CPU y estado del servicio.

Justo debajo, aparecen tarjetas individuales por cada agente declarado. En este caso se muestra el módulo central. Desde aquí se puede reiniciar cada servicio por separado o eliminar el agente (con doble confirmación). Esta vista también permite validar si Snort está en ejecución, cuántas alertas ha generado y cuál fue la última detectada.

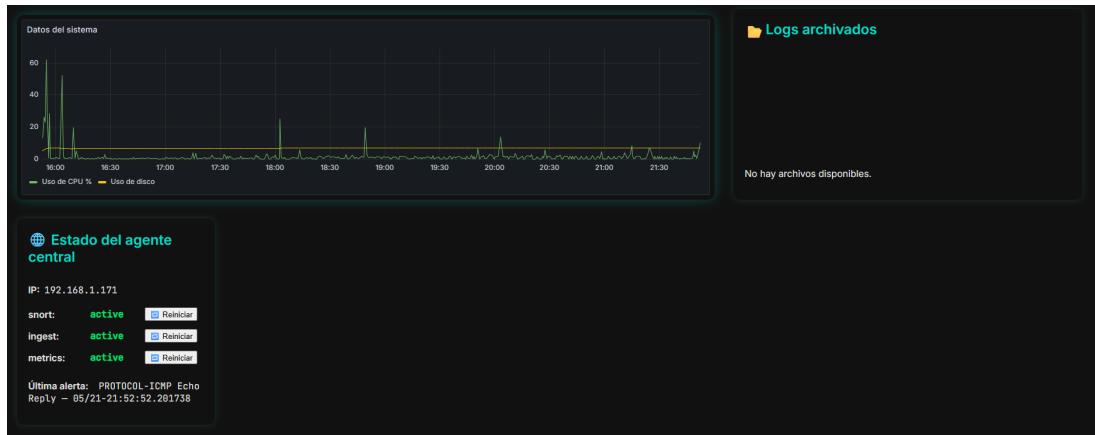


Figura 3.20: Estado detallado del agente central.

El menú desplegable de selección de agente permite cambiar rápidamente de instancia monitorizada. Inicialmente solo está presente el módulo central, pero se pueden añadir más agentes en caliente.

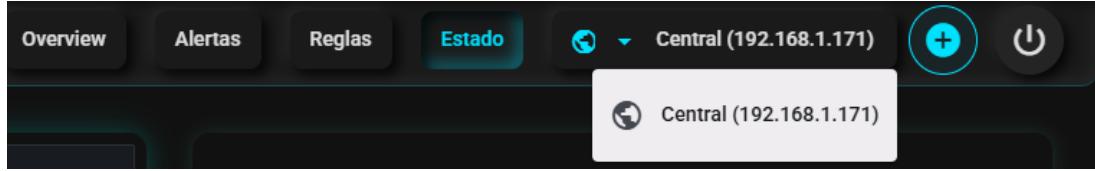


Figura 3.21: Menú de selección de agentes activos.

Para registrar un nuevo agente, basta con pulsar el botón + de la esquina superior derecha. Se abrirá un formulario donde se especifica el nombre del agente y su dirección IP. El sistema previene duplicados automáticamente.

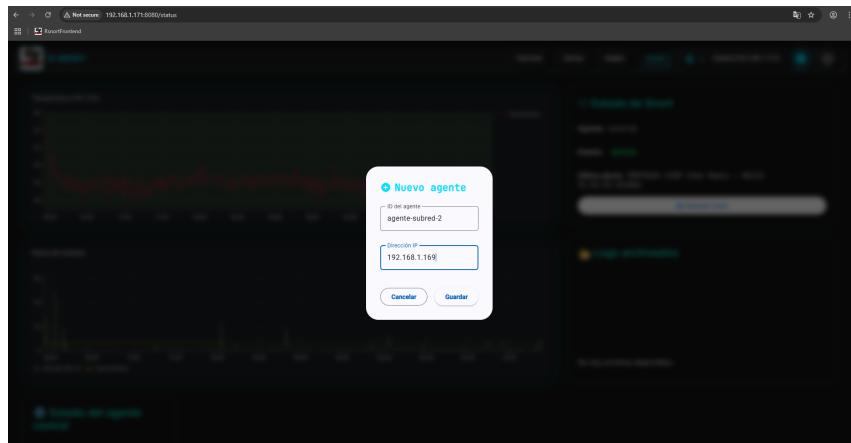


Figura 3.22: Formulario para añadir un nuevo agente Snort.

Una vez añadido, el nuevo agente aparecerá en el desplegable y el dashboard se actualizará dinámicamente con sus métricas y alertas. En el ejemplo mostrado, se ha conectado un segundo agente en la subred 2, el cual genera menos tráfico por haber estado inactivo durante parte del día.

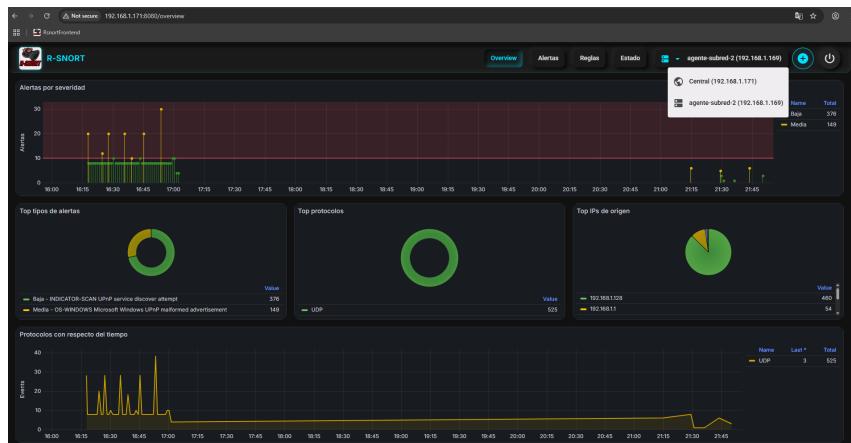


Figura 3.23: Visualización de métricas del agente secundario.

Finalmente, en la vista de **Estado** ahora se muestran múltiples tarjetas, una por cada agente activo. Cada una de ellas permite gestionar de forma independiente los servicios del agente y consultar su última alerta. Además, se incluye la opción de eliminar el agente de forma segura mediante una doble confirmación visual.

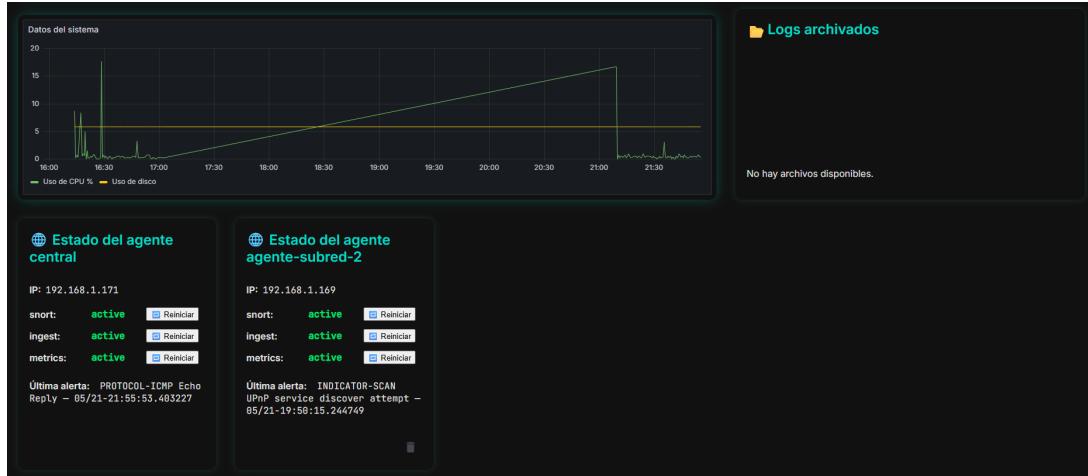


Figura 3.24: Gestión avanzada de agentes Snort desde la interfaz.

### 3.3.2. Pruebas de rendimiento

Gracias a que el propio diseño del módulo central de R-Snort cuenta con métricas del sistema que se actualizan en tiempo real vamos a aprovechar estas con el fin de hacer unas pruebas de rendimiento bajo uso normal o moderado del sistema, es decir simplemente escuchando la red sin ningún tipo de ataque ni alteración que pueda disparar las alertas y posteriormente haremos un envío de datos masivos para tratar de comprobar la estabilidad del sistema y como responde ante dicho envío de datos.

#### Monitorización del rendimiento en funcionamiento normal

Para analizar el comportamiento del sistema R-Snort durante su funcionamiento ordinario, se ha optado por utilizar directamente los paneles de Grafana integrados en la interfaz web del módulo central. Esta decisión no solo permite evitar herramientas externas, sino que además ilustra la capacidad del propio sistema para autodiagnosticarse en tiempo real, un aspecto clave en entornos de monitorización profesional.

Las métricas analizadas han sido las siguientes:

- **Temperatura de la CPU.** Como se aprecia en la Figura 3.25, la temperatura del sistema se mantiene entre los 46 y 51 grados durante un intervalo prolongado de uso normal. Estas cifras se encuentran dentro de los márgenes aceptables para una Raspberry Pi 5 en carga moderada, lo cual indica una refrigeración adecuada y ausencia de sobrecalentamientos.
- **Uso de CPU (%).** Tal como muestra la Figura 3.26, el consumo de CPU se mantiene mayoritariamente por debajo del 10 %, con algunos picos aislados que no superan el 30 %. Este comportamiento es coherente con un sistema que opera de manera eficiente, sin procesos en bucle ni sobrecargas injustificadas.

- **Uso de disco.** También representado en la Figura 3.26, el uso del disco se estabiliza en torno al 7 % de capacidad total. Esta métrica sugiere que los logs y la base de datos de alertas no saturan el almacenamiento durante la operación diaria del sistema.

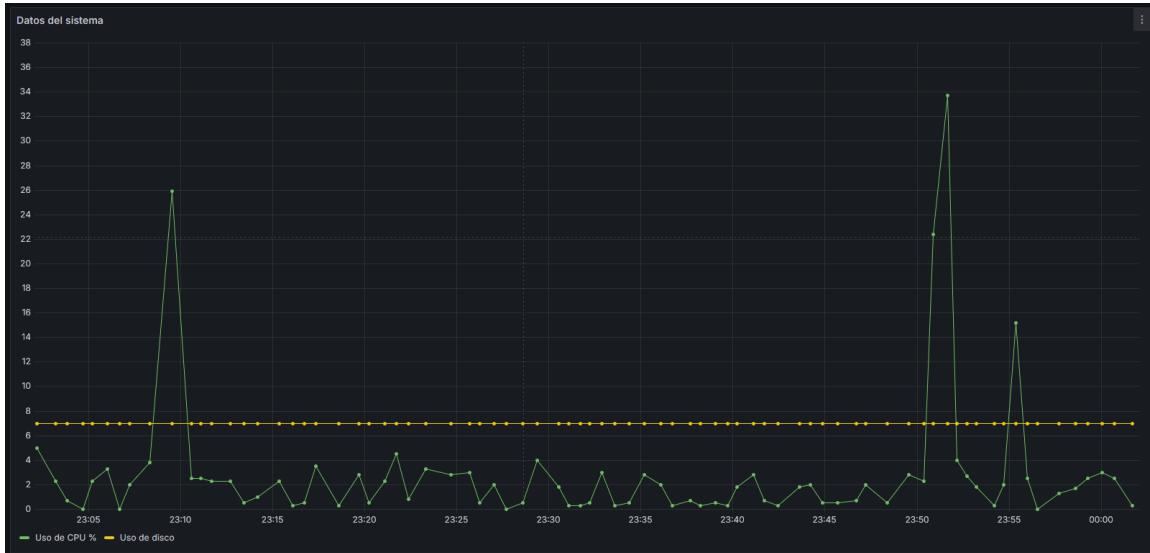


Figura 3.25: Temperatura de la CPU durante funcionamiento normal.

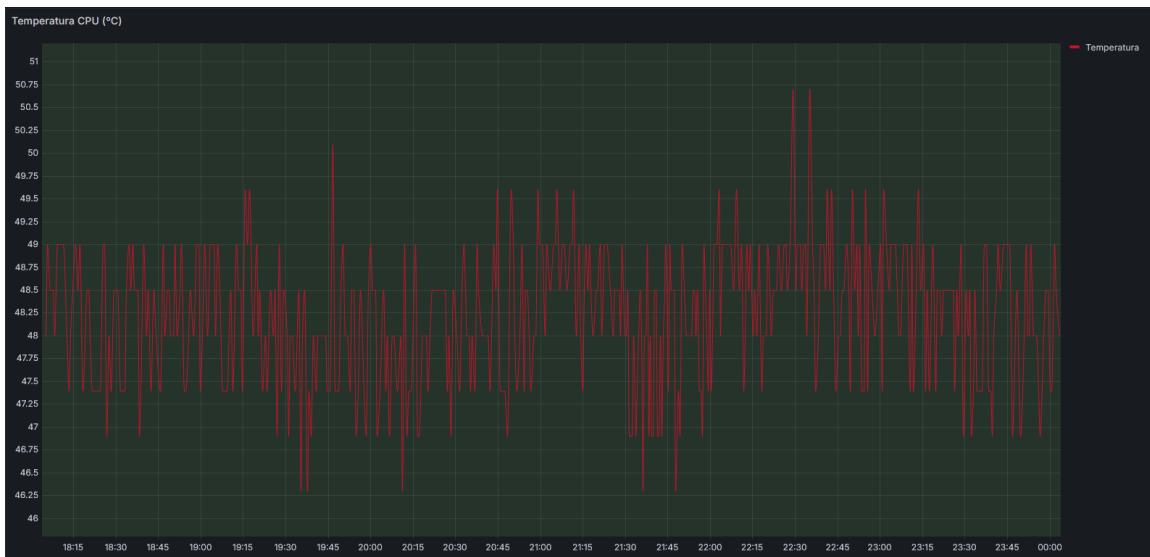


Figura 3.26: Consumo de CPU y uso de disco durante funcionamiento normal.

En conjunto, estos resultados permiten afirmar que R-Snort presenta un comportamiento estable y eficiente en condiciones de uso normales. Las gráficas no solo validan la correcta instalación de los scripts, sino también la eficacia de los servicios en ejecución, sin que se produzcan cuellos de botella ni compromisos en la estabilidad del sistema.

### Análisis de rendimiento en situación de estrés

Para evaluar el comportamiento del sistema R-Snort ante un escenario de uso intensivo, se ha llevado a cabo una prueba de estrés basada en el envío masivo de paquetes desde varias

terminales empleando `iperf3` contra la interfaz promiscua del agente central. Esta situación simula un ataque de denegación de servicio distribuido (DDoS), en el cual se pretende saturar tanto la red como los recursos del sistema, y evaluar la capacidad de detección y resistencia de la Raspberry Pi 5 actuando como nodo central.

### Descripción de las pruebas de estrés

Durante el experimento se han abierto múltiples instancias concurrentes de `iperf3`, generando tráfico simultáneo a través de diferentes puertos. A su vez, se ha mantenido Snort 3 activo con sus correspondientes preprocesadores y reglas personalizadas, recogiendo y analizando cada paquete de red. Las métricas de rendimiento se han recogido mediante Grafana, obteniendo series temporales de las siguientes variables clave:

- Uso de CPU (%).
- Uso de disco (%).
- Temperatura de la CPU (°C).
- Alertas registradas por severidad y tipo.

### Resultados observados

- En la Figura 3.27, correspondiente al tráfico normal previo al ataque, se observa un número moderado de alertas, con predominancia de alertas de severidad baja o media, originadas principalmente por tráfico ICMP o DNS.
- En la Figura 3.28, tras iniciar la prueba de carga, se detecta un incremento inmediato y sostenido del uso de CPU, alcanzando picos superiores al 60 % de uso. Sin embargo, el uso de disco permanece estable, gracias al sistema de rotación de logs y al diseño eficiente del almacenamiento.
- La Figura 3.29 muestra el incremento progresivo de la temperatura del sistema, alcanzando máximos en torno a los 62 grados (con refrigeración) durante la fase más intensa del ataque. A pesar del aumento térmico, el sistema mantuvo la estabilidad operativa sin pérdidas de servicio.

### Conclusiones

Estas pruebas demuestran que el sistema R-Snort es capaz de soportar picos de carga elevados durante períodos prolongados sin comprometer la integridad del sistema. El uso sostenido de CPU y el aumento de la temperatura son respuestas esperadas bajo estas condiciones, pero en ningún momento se observaron reinicios, errores del servicio o pérdida de conectividad. Tampoco se detectaron indicios de pérdida de paquetes ni omisiones en la detección de alertas relevantes durante los ensayos, lo que sugiere una capacidad de procesamiento estable incluso bajo estrés.

Últimas alertas							
Hora	Alerta	Origen	Destino	Puerto	Clase de alerta	Severidad	
2025-05-22 00:54:45	PROTOCOL-ICMP Unusual PING detected	192.168.1.131	142.250.184.4		Alerta de protocolo	Media	
2025-05-22 00:54:45	PROTOCOL-ICMP PING	192.168.1.131	142.250.184.4		Alerta de protocolo	Baja	
2025-05-22 00:54:45	PROTOCOL-ICMP Echo Reply	142.250.184.4	192.168.1.131		Alerta de protocolo	Baja	
2025-05-22 00:54:37	INDICATOR-SHELLCODE ssh CRC32 overflow filler	192.168.1.134	192.168.1.171	22	Tráfico desconocido	Alta	
2025-05-22 00:54:37	Sensitive Data - Email detected	192.168.1.134	192.168.1.171	22	Tráfico desconocido	Media	
2025-05-22 00:54:30	PROTOCOL-DNS SPOOF query response with TTL of 1 min. and no author	46.6.113.34	192.168.1.156	65200	Alerta de protocolo	Media	
2025-05-22 00:54:30	PROTOCOL-DNS SPOOF query response with TTL of 1 min. and no author	46.6.113.34	192.168.1.156	51196	Alerta de protocolo	Media	

Figura 3.27: Alertas registradas durante tráfico normal.



Figura 3.28: Porcentaje de uso de CPU y disco durante el ataque.

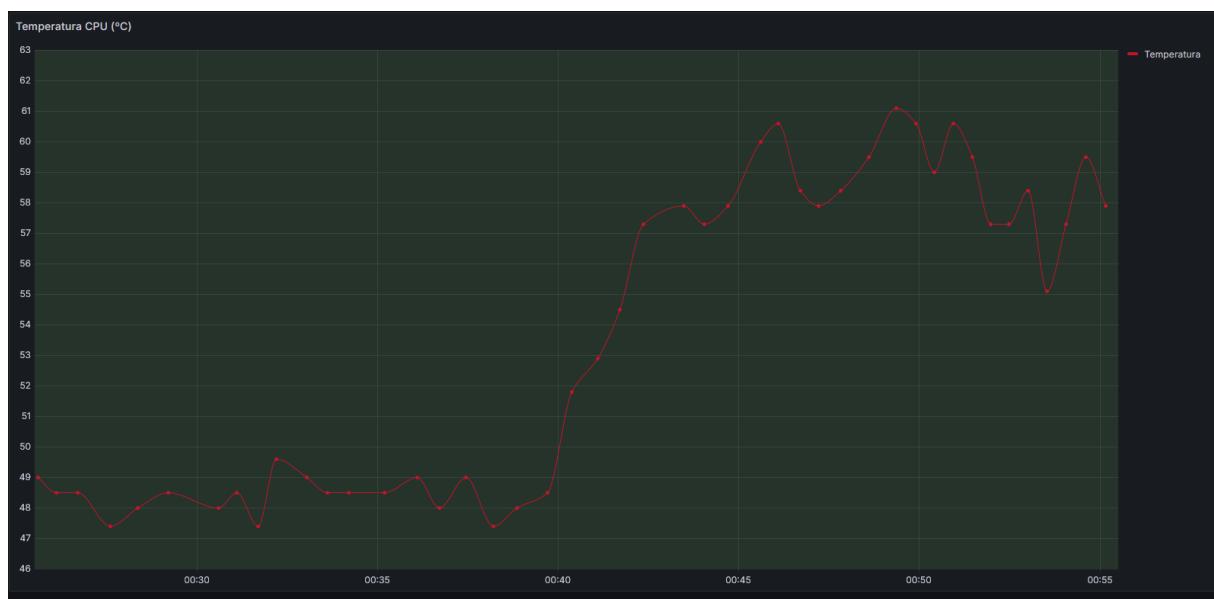


Figura 3.29: Temperatura de la CPU durante la fase de estrés.

Para este ataque masivo con el objetivo de someter al sistema R-Snort a condiciones extremas y evaluar su capacidad de detección y estabilidad bajo situaciones de estrés, se diseñó un

script que ejecuta múltiples ataques en paralelo. Este script fue lanzado desde una máquina externa WSL y permitió generar un volumen de tráfico malicioso elevado, simulando un escenario realista de intrusión.

Las pruebas se han realizado activando simultáneamente diferentes vectores de ataque: un *TCP SYN Flood*, un *ICMP Flood*, un escaneo agresivo con *nmap*, tráfico constante con *iperf3*, y un intento limitado de fuerza bruta sobre el servicio *ssh* utilizando *hydra*. Cada ejecución del script puede adaptarse para utilizar diferentes puertos de iperf, lo que permite lanzarlo desde varias terminales en paralelo sin conflictos.

El siguiente fragmento muestra el contenido del script utilizado:

```
1 # PUERTO UNICO POR INSTANCIA
2 PUERTO_IPERF="${1:-5201}"
3
4 echo "[+] ATAQUE desde terminal con IPERF en puerto $PUERTO_IPERF"
5 echo "====="
6
7 # 1. TCP SYN Flood
8 echo "[1] Iniciando TCP SYN Flood con hping3..."
9 sudo hping3 -S --flood -p 80 $R_SNORT_IP &
10
11 # 2. ICMP Flood
12 echo "[2] Iniciando ICMP Flood..."
13 sudo hping3 -1 --flood $R_SNORT_IP &
14
15 # 3. Escaneo nmap agresivo
16 echo "[3] Lanzando escaneo Nmap agresivo..."
17 sudo nmap -sS -sU -T4 -A $R_SNORT_IP --reason --script "default" -p- &
18
19 # 4. Tráfico continuo con iperf3 (en puerto variable)
20 echo "[4] Ejecutando iperf3 -c $R_SNORT_IP -p $PUERTO_IPERF ..."
21 iperf3 -c $R_SNORT_IP -p $PUERTO_IPERF -t 60 -b 100M &
22
23 # 5. Ataque de fuerza bruta con hydra (limitado)
24 echo "[5] Simulando fuerza bruta con Hydra..."
25 hydra -l admin -p admin ssh://$R_SNORT_IP -t 4 -V -f &
26
27 echo "====="
28 echo "Todos los ataques están en ejecución. Observar las gráficas."
```

Código 3.4: Script para generar ataques masivos contra R-Snort

Este enfoque permitió verificar no solo la capacidad del sistema para generar alertas ante ataques múltiples, sino también analizar el rendimiento de la Raspberry Pi bajo carga, así como la eficacia del almacenamiento de alertas y la visualización en tiempo real mediante Grafana.

### 3.3.3. Pruebas funcionales

Para el apartado de pruebas se pretende verificar la estabilidad y eficiencia de la interfaz frontend de R-Snort ante un volumen extremo de alertas generado durante las pruebas de estrés con tráfico malicioso y las demás funcionalidades del ecosistema R-Snort.

### Registro de alertas masivas

Durante los benchmarks realizados en el apartado *3.3.2 Benchmark de rendimiento*, se generaron decenas de miles de alertas mediante ataques simulados que incluían DDoS, escaneos agresivos y envíos masivos de paquetes ICMP, TCP y UDP con herramientas como hping3, iperf3 y nmap. Como resultado, Snort detectó y registró más de **250.000 alertas** en un intervalo de pocas horas.

La interfaz de R-Snort, tanto en la vista de alertas como en el panel de Grafana incrustado, fue puesta a prueba visualizando este gran volumen de información. Se evaluaron los tiempos de carga, la fluidez al cambiar entre vistas y la capacidad de la base de datos para manejar consultas masivas sin cuellos de botella visibles.

### Resultado

A pesar del gran volumen de datos, la aplicación web se comportó de forma fluida y sin fallos. En la vista Alertas, se mostraron correctamente los conteos por severidad, así como las tablas paginadas con miles de resultados. Por otro lado, los dashboards de Grafana embebidos en la sección Overview actualizaron en tiempo real los gráficos de tendencias por severidad, tipos de alerta, IPs de origen y protocolos.

Este comportamiento confirma que R-Snort es capaz de operar eficazmente incluso en escenarios de saturación de red, cumpliendo así uno de sus objetivos clave como herramienta profesional para entornos empresariales.

#### Componentes implicados:

- Backend de ingestión de alertas (`ingest_service.py`)
- API REST del agente y del módulo central
- Base de datos MariaDB para almacenamiento de alertas
- Frontend Angular (tablas, visualización por severidad)
- Dashboards de Grafana incrustados (Overview)



Figura 3.30: Dashboard de Grafana mostrando más de 250.000 alertas tras una prueba de estrés. Se observan tendencias por severidad, protocolo e IPs de origen.



Figura 3.31: Panel de alertas en la aplicación web con visualización correcta de alertas severas, medias y bajas tras las pruebas de carga.

## Recorrido de las funciones

En esta sección se describen algunas de las funcionalidades adicionales del sistema R-Snort implementadas en la interfaz web, que no habían sido abordadas o lo fueron muy superficialmente en apartados anteriores. Estas funcionalidades están orientadas a facilitar el uso y mantenimiento del sistema por parte de un administrador de red, permitiendo gestionar reglas, supervisar el estado de los agentes, y acceder a los registros generados por Snort.

### Descarga de logs archivados

Una de las funcionalidades destacadas es el sistema de rotación y archivado de logs implementado en los agentes. Este sistema previene que los archivos de alertas crezcan indefinidamente, lo cual es especialmente útil en entornos con tráfico elevado. Los archivos rotados se comprimen automáticamente y se almacenan como copias de respaldo, accesibles desde la interfaz. Se ha incorporado un botón para descargar estos logs archivados, lo cual resulta especialmente útil para tareas de análisis forense o revisión posterior.

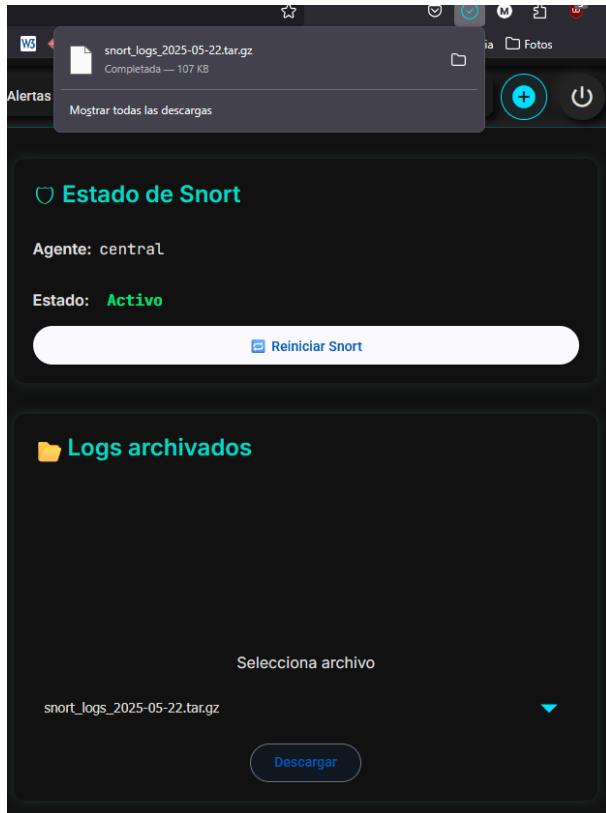


Figura 3.32: Descarga de archivos de logs archivados generados por el sistema de rotación.

### Exportación de alertas activas en CSV

Además de los logs comprimidos, el sistema permite exportar las alertas activas en formato .csv, tanto para un agente específico como para todos los agentes de la red. Esta funcionalidad permite al administrador exportar la información y analizarla con herramientas externas como hojas de cálculo o entornos de análisis de datos.

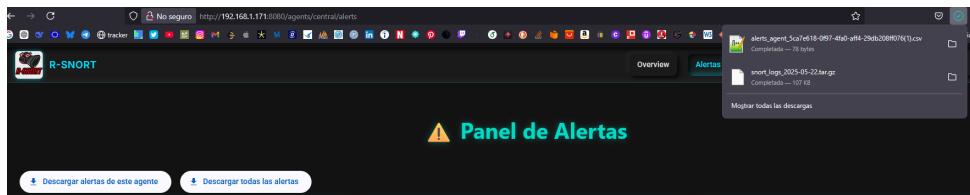


Figura 3.33: Opciones de descarga de alertas activas por agente o globales en formato CSV.

### Añadir reglas personalizadas a Snort

En el módulo de gestión de reglas, el sistema permite añadir nuevas reglas personalizadas al archivo `custom.rules` del agente. El formulario de inserción verifica automáticamente que la regla no esté duplicada y que su sintaxis sea válida. En caso de error, se notifica al usuario de forma clara. A continuación, se muestra la creación de una nueva regla para detectar intentos de acceso externo al puerto 22 (SSH).

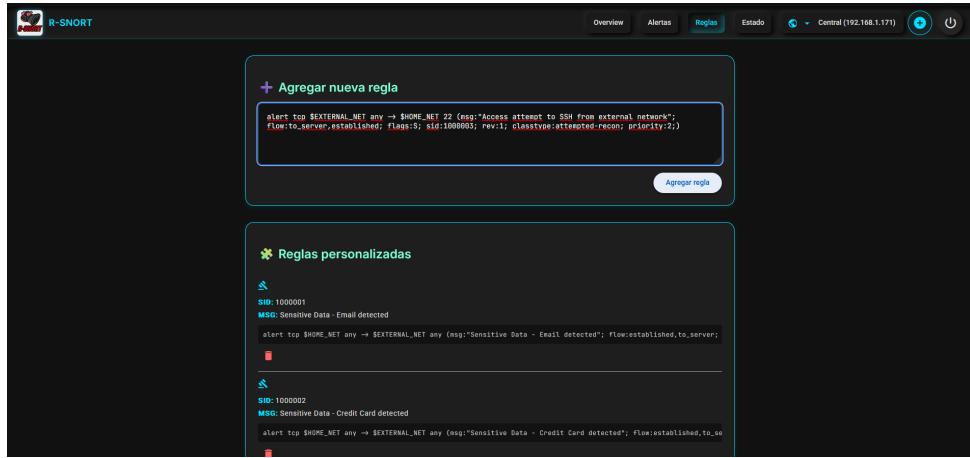


Figura 3.34: Inserción de una nueva regla de detección de conexiones SSH no autorizadas.

Una vez agregada, la regla aparece lista junto a las demás reglas activas. Esto permite al administrador confirmar su incorporación y tener una visión clara de las políticas de detección vigentes en el sistema.

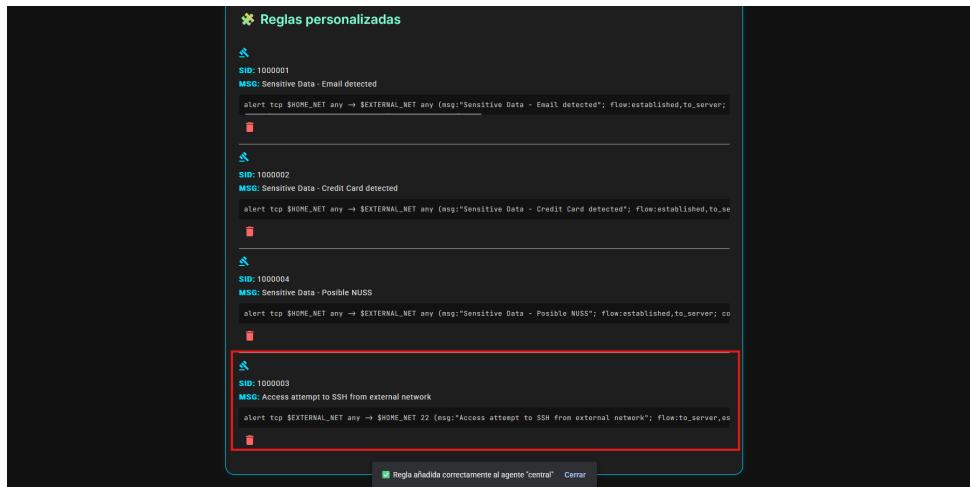


Figura 3.35: Regla personalizada añadida correctamente al sistema del agente central.

### Eliminación segura de reglas

Del mismo modo, el sistema permite eliminar reglas personalizadas de forma controlada. Cada eliminación requiere confirmación explícita por parte del usuario para evitar modificaciones accidentales.

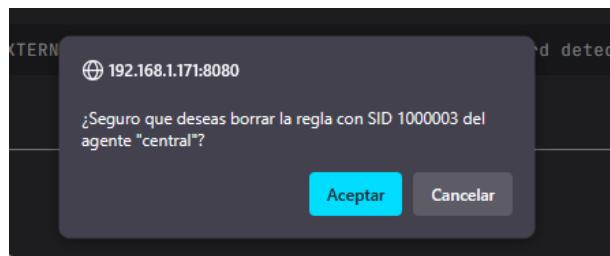


Figura 3.36: Confirmación de eliminación de una regla personalizada.

### Gestión de agentes y eliminación con verificación

En cuanto a la gestión de agentes, el sistema permite también eliminar instancias previamente registradas. Dado que esta acción puede afectar la supervisión de una subred completa, el sistema solicita una doble confirmación, reforzando así la seguridad en las operaciones críticas.

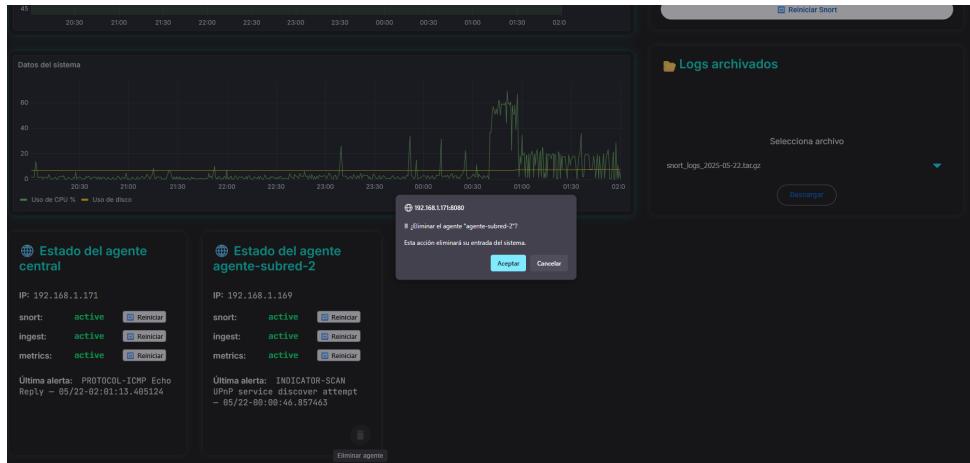


Figura 3.37: Verificación antes de eliminar un agente de la interfaz.

### Supervisión de servicios y reinicio desde la interfaz

Por último, el panel de estado permite visualizar en tiempo real la disponibilidad de los servicios principales de cada agente (Snort, ingestión de alertas y métricas del sistema). En caso de que algún servicio se encuentre caído o inactivo, el administrador puede reiniciarlo directamente desde la interfaz sin necesidad de acceso por terminal. Esto proporciona una herramienta de recuperación rápida ante fallos.



Figura 3.38: Estado de los servicios de los agentes y opción de reinicio remoto.

Estas pruebas funcionales demuestran que el sistema R-Snort no solo es capaz de detectar tráfico sospechoso y generar alertas, sino que también proporciona un conjunto de herramientas

completas para la administración y supervisión de agentes, reglas y registros, mejorando la capacidad operativa y reduciendo la necesidad de intervención manual.

### 3.4. Resumen

El capítulo muestra un caso práctico exitoso del sistema R-Snort implementado en un entorno realista para pequeñas oficinas (SOHO). Se ha utilizado una infraestructura sencilla pero al mismo tiempo viable, formada por un módulo central basado en Raspberry Pi 5 y un agente secundario en Raspberry Pi 3, ambos conectados a switches gestionables que facilitan la monitorización eficiente del tráfico mediante port mirroring.

El módulo central integra de manera el backend en Spring Boot, la interfaz web desarrollada en Angular, la base de datos MariaDB y el dashboard interactivo en Grafana, proporcionando una solución completa de gestión centralizada. Por su parte, el ambos agentes ejecutan una versión optimizada de Snort 3 (desarrollada en el Complemento del TFG), adaptada específicamente para dispositivos de bajo consumo.

La instalación se destaca por ser completamente automatizada gracias a scripts especializados, lo que simplifica considerablemente la implementación en entornos reales. Las pruebas efectuadas han demostrado de manera concluyente que R-Snort maneja sin problemas grandes volúmenes de alertas (superiores a 250.000), manteniendo un rendimiento estable y una visualización fluida y dinámica de la información crítica.

Finalmente, las pruebas funcionales adicionales han confirmado la eficacia y fiabilidad del sistema, destacando funcionalidades importantes en el ámbito de la ciberseguridad y la administración de sistemas como la rotación y archivado automático de logs, la exportación sencilla de alertas, la gestión dinámica de reglas personalizadas y la administración remota intuitiva de servicios. Todo ello refuerza claramente el valor práctico del proyecto R-Snort como herramienta profesional, accesible y eficaz para la seguridad y supervisión de redes empresariales pequeñas y medianas.



# 4. Resultados y discusión

## 4.1. Resultados obtenidos

El sistema R-Snort, tras su despliegue en un entorno simulado de red tipo SOHO y bajo condiciones de uso realista, ha permitido explorar y validar de forma práctica tanto la viabilidad técnica del ecosistema como su utilidad operativa para pequeñas y medianas empresas. A continuación se presentan los resultados obtenidos durante la fase experimental, diferenciando entre los aspectos de funcionalidad, rendimiento, experiencia de usuario y capacidad de adaptación.

### 4.1.1. Eficacia en la detección de amenazas

Durante la batería de pruebas funcionales, el sistema fue capaz de identificar un amplio rango de incidentes de seguridad: desde escaneos básicos de puertos y ataques ICMP, hasta patrones de exfiltración de datos y tráfico anómalo en protocolos DNS y SNMP. Las alertas generadas por Snort 3 fueron correctamente recogidas por el backend, almacenadas en la base de datos y presentadas en tiempo real a través del frontend (véase Figuras 3.16 y 3.17). Cabe destacar que la integración con Grafana permitió visualizar tendencias y picos de actividad, lo que facilita la rápida identificación de eventos anómalos o posibles intentos de actividad maliciosa o intrusión además de filtración de datos. Esta capacidad de monitorización continua es importante para el entorno PYME, donde la inmediatez en la detección puede suponer la diferencia entre contener un incidente o sufrir un compromiso mayor.

En las pruebas orientadas a la gestión de reglas, el sistema demostró flexibilidad: fue posible añadir, modificar y eliminar reglas personalizadas desde la interfaz gráfica, reflejándose los cambios en el comportamiento del sensor sin requerir intervención manual en consola. Sin embargo, debe reconocerse que, aunque la interfaz es intuitiva, la comprensión de la lógica subyacente a cada regla aún requiere cierto conocimiento técnico, lo que podría limitar la autonomía del usuario final en escenarios menos formados.

### 4.1.2. Rendimiento y escalabilidad

En términos de rendimiento, la solución mostró un comportamiento estable ante tráfico normal y bajo cargas elevadas inducidas por ataques simulados (véanse Figuras 3.25 a 3.29 y el panel de Grafana 3.29). Durante las pruebas de estrés, la Raspberry Pi 5 mantuvo un uso de CPU y temperatura dentro de márgenes aceptables, sin caídas del servicio ni pérdida de alertas. El almacenamiento de eventos alcanzó volúmenes superiores a 250.000 alertas, gestionadas correctamente gracias al sistema de rotación y archivado automático de logs.

No obstante, se identifican ciertos límites en la capacidad de procesamiento: bajo ataques DDoS prolongados, el retardo en la inserción de nuevas alertas aumentó ligeramente, y se evidenció la importancia de ajustar tanto el número de reglas activas como los parámetros de preprocesamiento según el hardware disponible. Así, aunque el sistema es adecuado para el entorno objetivo (redes pequeñas o medianas), su escalabilidad está condicionada por la arquitectura de Raspberry Pi; para escenarios empresariales de mayor tamaño sería necesario migrar

a hardware más potente o implementar mecanismos de balanceo y paralelización de sensores.

#### 4.1.3. Usabilidad y experiencia de usuario

Uno de los objetivos principales de R-Snort era lograr una solución accesible a usuarios sin especialización profunda en ciberseguridad. En este sentido, la interfaz web desarrollada cumple de forma satisfactoria: permite la visualización clara y filtrada de alertas, la gestión de reglas y la monitorización de métricas relevantes del sistema. Las funcionalidades de descarga de logs, visualización por severidad y gráficos de actividad aportan valor añadido frente a otras alternativas existentes.

Sin embargo, el análisis de la experiencia de usuario sugiere áreas de mejora: aunque la curva de aprendizaje es más suave que en soluciones como Sguil, algunos flujos de configuración avanzada podrían simplificarse aún más para perfiles no técnicos. La gestión de incidencias y la correlación de eventos entre múltiples agentes, aunque implementada de forma básica, puede enriquecerse con mecanismos automáticos de priorización y alerta proactiva.

#### 4.1.4. Limitaciones y consideraciones

Resulta imprescindible señalar varias limitaciones detectadas durante el desarrollo y la validación del sistema:

- **Dependencia del hardware.** El rendimiento general está fuertemente condicionado por las capacidades de la Raspberry Pi. Si bien esta elección se justifica por su bajo coste, no es una solución universal para todas las PYMEs; aquellas con mayor carga de red deberán valorar una plataforma hardware superior.
- **Complejidad técnica residual.** Aunque se ha minimizado la necesidad de intervención manual, la instalación inicial y la resolución de ciertos errores pueden requerir conocimientos técnicos que exceden los de un usuario común. Es recomendable reforzar la documentación y considerar el desarrollo de asistentes de instalación.
- **Gestión de falsas alertas.** Como en cualquier IDS basado en firmas, existe el riesgo de generación de falsos positivos, especialmente cuando las reglas activas no están bien ajustadas al entorno real. Se recomienda realizar una fase de calibración antes de un despliegue definitivo.
- **Integración y futuro mantenimiento.** La modularidad del sistema permite futuras ampliaciones, pero depende de la compatibilidad de las versiones de Snort, Grafana y otros componentes open source. Es importante monitorizar estos cambios y actualizar el software regularmente para evitar obsolescencia o incompatibilidades.

### 4.2. Discusión

En conjunto, los resultados avalan la hipótesis de partida: es posible construir un sistema de detección y monitorización profesional, accesible y de bajo coste, capaz de cubrir las necesidades básicas de ciberseguridad en PYMEs. R-Snort se posiciona como una alternativa viable frente a soluciones comerciales, especialmente para organizaciones con recursos limitados y sin personal

demasiado especializado.

Ahora bien, la comparación con otras plataformas, tanto históricas como actuales (BASE, T-Pot, Sguil), evidencia que R-Snort no busca competir en funcionalidades avanzadas propias de un SIEM empresarial, sino ofrecer una solución equilibrada entre simplicidad, capacidad de adaptación y coste reducido. La integración de Grafana supone un salto cualitativo en la visualización de datos frente a la mayoría de alternativas open source tradicionales, pero también introduce nuevas dependencias técnicas que deben ser gestionadas.

Finalmente, el mayor valor de R-Snort radica en su capacidad de **democratizar** la ciberseguridad, acercando herramientas profesionales a un segmento empresarial tradicionalmente desatendido. Los resultados experimentales y la validación en entorno realista demuestran que, con un mantenimiento adecuado y una configuración inicial correcta, la solución propuesta puede suponer una mejora sustancial de la postura de seguridad en entornos PYME y SOHO. El reto futuro será mantener ese equilibrio entre sencillez, estabilidad y capacidad de adaptación a un panorama de amenazas en constante evolución.



# Conclusiones

El desarrollo y validación de R-Snort han permitido confirmar la viabilidad de una solución de monitorización y detección de intrusiones profesional, accesible y asequible orientada específicamente a las necesidades de las PYMEs españolas. El trabajo ha abarcado las fases fundamentales de un proyecto tecnológico real: desde la detección y justificación de la problemática, pasando por el diseño arquitectónico, la implementación técnica y la evaluación experimental, hasta la crítica y reflexión sobre los resultados alcanzados.

Uno de los logros más significativos ha sido demostrar que es posible dotar a pequeñas empresas de una herramienta avanzada de ciberseguridad, integrando tecnologías open source de alto nivel en una plataforma de bajo coste como la Raspberry Pi. La solución resultante además de ofrecer capacidades de detección frente a una amplia gama de amenazas y comportamientos sospechosos, sino que facilita su gestión y visualización a través de una interfaz intuitiva y amigable para usuarios no especializados. De este modo, R-Snort contribuye a reducir la brecha de ciberseguridad que actualmente sufren las PYMEs frente a grandes organizaciones.

Entre los principales puntos positivos cabe destacar:

- **Accesibilidad y facilidad de uso.** La interfaz web y los mecanismos automáticos de despliegue simplifican la puesta en marcha y el manejo del sistema, haciendo viable su adopción incluso por personal sin formación específica en ciberseguridad.
- **Eficacia técnica.** R-Snort ha demostrado su capacidad para detectar múltiples vectores de ataque y exfiltración de datos en tiempo real, generando alertas y métricas que facilitan el análisis y la reacción rápida ante incidentes.
- **Escalabilidad modular.** La arquitectura permite añadir nuevos agentes, ampliar la base de reglas y personalizar la monitorización en función de las necesidades de cada red, sin depender de soluciones propietarias ni de licencias costosas.
- **Bajo coste.** El uso de hardware asequible y software libre reduce drásticamente la barrera de entrada, facilitando la democratización de la ciberseguridad.

No obstante, el análisis crítico realizado durante el proyecto y la discusión de resultados invitan a matizar este optimismo. Las principales limitaciones identificadas incluyen la dependencia del hardware (que restringe el rendimiento bajo cargas extremas), la necesidad de ciertos conocimientos técnicos durante la instalación o resolución de incidencias, y el riesgo de falsos positivos en la detección, especialmente en entornos poco calibrados. Esto subraya la importancia de un mantenimiento adecuado, una buena documentación y, eventualmente, la posibilidad de migrar a plataformas de mayor potencia para escenarios más exigentes.

En perspectiva, R-Snort no aspira a competir con soluciones SIEM empresariales de gran escala, sino a ofrecer una alternativa realista que permita a las PYMEs protegerse de forma proactiva frente al creciente número de amenazas. El sistema, tal y como ha sido diseñado, es

extensible y adaptable, y sienta las bases para futuras evoluciones ya sea integrando inteligencia artificial para la correlación de eventos, añadiendo respuestas automáticas, o extendiendo la cobertura a dispositivos IoT.

En última instancia, este trabajo ejemplifica que la innovación no siempre requiere grandes presupuestos, sino una adecuada integración de tecnologías y un enfoque centrado en las necesidades reales del usuario. Haber llevado a cabo este proyecto ha supuesto, además de un reto técnico, un ejercicio de madurez personal y profesional, reforzando la convicción de que la ciberseguridad debe ser un derecho accesible y no un privilegio reservado a unos pocos.

# Trabajo futuro

Como ocurre en la mayoría de proyectos de un cierto tamaño, varias ideas y funcionalidades se han quedado en el tintero, principalmente por limitaciones de tiempo y de infraestructura. Algunas de ellas ya cuentan con bases en el código o el diseño, y podrían retomarse en el futuro para mejorar R-Snort y hacerlo más completo.

Una de las mejoras más evidentes sería la gestión avanzada de usuarios. Aunque la base de datos y parte del backend ya están preparados para admitir múltiples cuentas y roles, finalmente no se ha habilitado la opción de tener varios perfiles con distintos permisos en la plataforma. Implementar este sistema permitiría asignar tareas y responsabilidades de manera más granular, y acercaría la herramienta a entornos colaborativos y profesionales.

Otra función interesante sería la detección automática de agentes en la red local. La idea era que el instalador del módulo central pudiera escanear la red, localizar dispositivos activos y comprobar si alguno expone el endpoint '/docs' en el puerto 9000 (propio del API de los agentes). Esto permitiría añadir nuevos sensores de forma prácticamente automática, sin intervención manual. Sin embargo, se descartó porque podía generar errores o falsas detecciones, especialmente en redes donde varios equipos utilizan el mismo puerto para otros servicios, pero con los ajustes correctos se podría llevar a la práctica.

También se contempló implementar la verificación de correo electrónico al registrar el usuario administrador. Aunque se trata de una función básica en términos de seguridad, su integración requería desplegar y mantener un servidor de correo, lo que complicaba la instalación y aumentaba la dependencia de servicios externos. Por eso, de momento, se ha optado por un sistema más sencillo que evita estos requisitos.

Otra idea atractiva fue el diseño de paneles geográficos, mostrando sobre un mapa mundi el origen de las direcciones IP desde las que se reciben alertas. Aunque esta funcionalidad aporta un valor visual y contextual importante, resultó poco práctica para un entorno local como el de R-Snort. Para funcionar, necesitaba acceder a servicios de geolocalización de IPs, muchos de los cuales son de pago o requieren un registro manual que no es automatizable fácilmente en el instalador y se quería evitar tener que registrar al usuario en páginas de terceros.

Por último, otra mejora planteada sería el asistente de creación automática de reglas. El objetivo sería facilitar al usuario la configuración de nuevas reglas sin tener que aprender la sintaxis de Snort: bastaría con seleccionar las variables y condiciones deseadas desde la interfaz, y el sistema generaría automáticamente la regla correspondiente. Esto haría la herramienta mucho más accesible para perfiles no técnicos.

En resumen, el margen de mejora y expansión de R-Snort es amplio. Las ideas aquí mencionadas —y otras que surgirán con el tiempo y el uso real del sistema— muestran que aún hay recorrido para hacer de esta solución algo más versátil y amigable para todos los usuarios.



# Bibliografía

- [1] ENISA. *Cybersecurity for SMEs*. Agencia de la Unión Europea para la Ciberseguridad. Disponible en: <https://www.enisa.europa.eu/topics/awareness-and-cyber-hygiene/smes-cybersecurity#:~:text=Small%20and%20medium,the%20European%20Commission%20SMEs%20report>. Último acceso: mayo de 2025.
- [2] 20Minutos. *Google avisa que España tiene un problema gordo: el 43 % de los ciberataques son a PYMES*. Publicado en 20minutos.es, 2024. Disponible en. <https://acortar.link/3ygPF7> Último acceso: mayo de 2025.
- [3] Telefónica Cyber Security Tech. *Informe sobre la situación de la ciberseguridad en 2023*. Plataforma Tecnológica Española de Tecnologías Disruptivas (PTE Disruptive), 2023. Disponible en: [https://ptedisruptive.es/wp-content/uploads/2023/12/Informe-situacion-ciberseguridad-2023\\_compressed.pdf#:~:text=Telef%C3%B3nica%20Cyber%20Security%20Tech%2C%20el,empresarial%20en%20la%20era%20digital](https://ptedisruptive.es/wp-content/uploads/2023/12/Informe-situacion-ciberseguridad-2023_compressed.pdf#:~:text=Telef%C3%B3nica%20Cyber%20Security%20Tech%2C%20el,empresarial%20en%20la%20era%20digital). Último acceso: mayo de 2025.
- [4] INCIBE-CERT. *Sector PYMES NIS2*. Instituto Nacional de Ciberseguridad de España. Disponible en: <https://www.incibe.es/incibe-cert/sectores-estrategicos/pymes-nis2>. Último acceso: mayo de 2025.
- [5] A. Alcaide, “Sistemas IDS, IPS, HIDS, NIPS, SIEM – ¿Qué son?” *Blog de A2Secure*, 12 febrero 2019. [En línea]. Disponible: <https://www.a2secure.com/blog/ids-ips-hids-nips-siem-que-es-esto/>.
- [6] Wikipedia, “NIDS (Network Intrusion Detection System),” *Wikipedia en español*, última edición: 23 mayo 2023. [En línea]. Disponible: <https://es.wikipedia.org/wiki/NIDS>.
- [7] K. Scarfone y P. Mell, *Guide to Intrusion Detection and Prevention Systems (IDPS)*, NIST Special Publication 800-94, National Institute of Standards and Technology, 2007.
- [8] A. Tatistcheff, “Snort 3: Rearchitected for Simplicity and Performance,” *Cisco Security Blog*, 2021. [En línea]. Disponible: <https://blogs.cisco.com/security/snort-3-rearchitected-for-simplicity-and-performance>.
- [9] Sakura Sky, “Supporting Snort 3 and above,” *Sakura Sky Blog*, 2020. [En línea]. Disponible: <https://www.sakurasky.com/blog/supporting-snort/>.
- [10] Grafana Labs, “SNORT 3, using JSON alert on latest GRAFANA dashboard,” *Grafana Labs Community Forums*, mensaje de usuario barukcic, 27 junio 2020. [En línea]. Disponible: <https://community.grafana.com/t/snort-3-using-json-alert-on-latest-grafana-dashboard/32798>.
- [11] Snort.org, “GUIs for Snort,” *Snort Blog*, 20 enero 2011. [En línea]. Disponible: <https://blog.snort.org/2011/01/guis-for-snort.html>.

- [12] Help Net Security, “Snorby: Modern Snort IDS frontend,” 7 diciembre 2010. [En línea]. Disponible: <https://www.helpnetsecurity.com/2010/12/07/snорby-modern-snort-ids-frontend/>.
- [13] BASE Project, *SourceForge*, disponible en: <https://sourceforge.net/projects/secureideas/>
- [14] D. Gullett, “Re: Snort Report 2.0 Beta Released,” mensaje en *Snort-users Mailing List*, 18 junio 2010. [En línea]. Disponible: <https://seclists.org/snort/2010/q2/898>.
- [15] Wikipedia, “Aanval,” *Wikipedia, The Free Encyclopedia*, 2023. [En línea]. Disponible: <https://en.wikipedia.org/wiki/Aanval>.
- [16] Insecure.org, “Sguil – Open Source Network Security Monitoring,” *SecTools Top Network Security Tools*, 2006. [En línea]. Disponible: <https://sectools.org/tool/sguil/>.
- [17] Iszi, “Snort’s great, but BASE isn’t. What are some alternative front-ends?” *Security StackExchange*, pregunta y respuestas, febrero 2011. [En línea]. Disponible: <https://security.stackexchange.com/q/2041>.
- [18] pfSense Project. “pfSense: The Open Source Firewall and Router Platform”. [En línea]. Disponible en: <https://www.pfsense.org>. [Consulta: 18 May 2025].
- [19] CISA. “Snort Intrusion Detection System”. Cybersecurity & Infrastructure Security Agency. [En línea]. Disponible en: <https://www.cisa.gov/resources-tools/services/snort>. (Referencia a BASE como interfaz web para Snort).
- [20] Deutsche Telekom Security. “T-Pot: The All In One Multi Honeypot Platform (GitHub README)”. [En línea]. Disponible en: <https://github.com/telekom-security/tpotce>. [Consulta: 18 May 2025].
- [21] Russ Combs and Jon Munshaw, *The major differences that set Snort 3 apart from Snort 2*, Snort Blog, Cisco Talos, 5 de agosto de 2020.
- [22] FastAPI Framework, *FastAPI - Fast, modern web framework for building APIs with Python 3.7+*, Documentación oficial, [En línea]. Disponible: <https://fastapi.tiangolo.com>. [Último acceso: May 2025].
- [23] Arun KL, *Turn Your Raspberry Pi as a IDS/IPS Box and Hunt for Potential Intrusions on Your Network*, TheSecMaster (blog), 28 de junio de 2024.
- [24] Grafana Labs, *Grafana - The open source analytics & monitoring solution*, [En línea]. Disponible: <https://grafana.com>. [Último acceso: May 2025].
- [25] M. Jones, J. Bradley, N. Sakimura, *RFC 7519: JSON Web Token (JWT)*, IETF, mayo 2015.
- [26] Grafana Labs. (s.f.). \*Snort (Community) Dashboard for Grafana\*. Recuperado de <https://grafana.com/grafana/dashboards/11191-snort/>
- [27] Grafana Labs. *Snort IDS/IPS Dashboard*. Dashboard público ID 11191. [En línea]. Disponible en: <https://grafana.com/grafana/dashboards/11191-snort/>.

## BIBLIOGRAFÍA

---

- [28] Raspberry Pi Foundation. *Raspberry Pi Software*. Disponible en: <https://www.raspberrypi.com/software/>. Accedido el 21 de mayo de 2025.



# I. Descripción de la plataforma R-Snort

Este anexo ofrece una visión general y práctica de la plataforma R-Snort desarrollada durante el TFG, actuando como guía de referencia para usuarios y administradores que deseen desplegar, gestionar o ampliar el sistema. Se describen los componentes principales, el flujo de funcionamiento, los requisitos técnicos y los pasos básicos de uso y administración.

## I.1. ¿Qué es R-Snort?

R-Snort es una solución integral de monitorización y detección de intrusiones en red, diseñada específicamente para pequeñas y medianas empresas (PYMEs) y entornos SOHO. El sistema se basa en el motor Snort 3, una Raspberry Pi 5 como hardware principal y una interfaz web para facilitar su uso, todo ello reforzado con dashboards de Grafana para la visualización de métricas y alertas en tiempo real. La filosofía de R-Snort es ofrecer un sistema profesional, estable y sencillo de gestionar, incluso para usuarios con poca experiencia técnica.

## I.2. Componentes principales de la plataforma

La arquitectura de R-Snort está compuesta por tres grandes bloques:

- **Agente R-Snort (sensor).** Se trata de una Raspberry Pi que ejecuta Snort 3 configurado para monitorizar el tráfico de red, generar alertas y transmitirlas al backend. El agente puede instalarse en cualquier punto estratégico de la red, habitualmente conectado a un puerto espejo del switch principal.
- **Backend central (Spring Boot).** Actúa como intermediario y cerebro del sistema. Recibe las alertas de los agentes, las almacena en una base de datos MariaDB, expone una API REST segura para consultas y operaciones, y gestiona la lógica de negocio (usuarios, reglas, métricas, etc.).
- **Frontend web (Angular).** Proporciona una interfaz gráfica intuitiva y accesible. Permite visualizar alertas en tiempo real, consultar el estado de los agentes, gestionar reglas, descargar logs, supervisar métricas y acceder a paneles de Grafana integrados.

## I.3. A continuación se muestra el flujo:

- El agente R-Snort inspecciona el tráfico de red y detecta eventos sospechosos utilizando reglas personalizadas y comunitarias.
- Las alertas se generan en formato JSON, y los servicios del sistema se encargan de almacenarlas en una base de datos, para posteriormente enviarlas automáticamente al backend central.
- El backend pone las alertas a disposición de la interfaz web y de Grafana, donde son interpretadas en distintos paneles mediante consultas SQL.

- El usuario accede al frontend, desde donde puede visualizar alertas, configurar reglas, descargar registros históricos o consultar métricas de uso y estado del sistema.

## I.4. Requisitos mínimos

- **Hardware.** Raspberry Pi 4/5 (recomendado), microSD de al menos 32 GB para el sistema operativo y inicio de la instalación, sin embargo con el paso del tiempo se requerirá almacenaje en discos duros y conexión por cable Ethernet a la red a monitorizar. Para el backend central, se recomienda un equipo con al menos 2 GB de RAM.
- **Software:** Ubuntu Server 22.04/24.04 o Debian reciente, Snort 3.1.84+, MariaDB, Spring Boot (Java 17+) y Node.js 18+ (para Angular).
- **Red.** Acceso a un puerto espejo (port mirroring) del switch o router para capturar el tráfico a analizar.

## I.5. Instalación y despliegue básico

1. **Instalar el agente.** Ejecutar el script ‘snort-agent’ en la Raspberry Pi. El proceso instala Snort 3, sus dependencias y configura el sistema para el arranque automático además de definir la base de datos MariaDB y la implantación del dashboard de Grafana.
2. **Desplegar el backend central.** Ejecutar el instalador ‘r-snort-central-module’ en el servidor central. Se configuran los servicios Spring Boot, compilación del Frontend y modificaciones para MariaDB de manera automática.
3. **Configurar la red y agentes.** Asegurarse de que los agentes pueden comunicarse con el backend y están conectados a los puertos correctos.
4. **Acceder a la interfaz web.** Desde cualquier navegador, acceder a la IP del backend central en el puerto configurado (por defecto, 4200/443).
5. **Registrar usuarios y configurar reglas.** Crear el usuario administrador, añadir agentes si es necesario y empezar a gestionar reglas de detección y políticas de alertas.

## I.6. Funcionalidades principales

- Visualización de alertas en tiempo real y por histórico.
- Filtros por severidad, tipo, agente y origen/destino.
- Edición y despliegue de reglas Snort desde la interfaz web.
- Paneles de métricas y gráficas de Grafana integrados en la aplicación.
- Descarga de logs y registros archivados para análisis forense.
- Gestión y monitorización de múltiples agentes en distintas subredes.
- Sistema de roles y usuarios (en futuras versiones).

## I.7. Consejos de administración y mantenimiento

Para asegurar la fiabilidad a largo plazo del sistema R-Snort y garantizar una protección continua frente a amenazas, se recomienda seguir una serie de buenas prácticas de mantenimiento. Estas acciones contribuyen tanto a la estabilidad del sistema como a su capacidad de adaptación ante cambios en la red o nuevas vulnerabilidades:

- Mantener actualizado el sistema operativo, así como las versiones de Snort y sus dependencias, con el fin de aprovechar mejoras de seguridad, corrección de errores y nuevas funcionalidades.
- Realizar copias de seguridad periódicas de la base de datos y los archivos de configuración, de modo que se pueda recuperar rápidamente el sistema ante fallos o pérdidas de información.
- Revisar y ajustar las reglas de detección con regularidad, adaptándolas a las características específicas del entorno de red para minimizar falsos positivos y mejorar la precisión.
- Consultar frecuentemente los paneles de Grafana para detectar tendencias inusuales, cuellos de botella o posibles incidentes de seguridad que requieran atención.
- Documentar todos los cambios relevantes realizados en el sistema, ya sea en configuraciones, reglas o infraestructura, facilitando así futuras auditorías, depuración de errores o escalado del sistema.

## I.8. Recursos y soporte

Toda la documentación técnica, scripts de instalación, ejemplos de reglas y paneles de Grafana, así como posibles actualizaciones futuras, se encuentran en el repositorio oficial de R-Snort (ver enlaces en la bibliografía y Anexo II). Para consultas adicionales, sugerencias o contribuciones, se anima a los usuarios a contactar con el autor o colaborar mediante pull requests en el repositorio.

—  
Esta descripción busca que cualquier usuario, aunque no sea experto, pueda hacerse una idea clara de cómo funciona y cómo desplegar la plataforma, sirviendo también como punto de partida para quienes quieran profundizar o contribuir a su mejora.



## II. Repositorios y guía rápida

A continuación se describen los repositorios oficiales de R-Snort y los pasos mínimos para descargar e instalar cada uno de los componentes principales. Toda la documentación extendida, scripts y archivos de configuración están disponibles en los propios repositorios de GitHub.

### II.1. Repositorio del Agente R-Snort

**Repo:** <https://github.com/deianp189/snort-agent>

Para clonar y lanzar la instalación automática del agente en una Raspberry Pi o servidor Ubuntu/Debian compatible:

```
git clone https://github.com/deianp189/snort-agent.git  
2 cd snort-agent  
sudo ./install.sh
```

The screenshot shows the README.md file of the Snort Agent GitHub repository. The title is "Snort Agent". The description states: "Sistema modular para convertir Snort 3 en un agente gestionado vía REST API con ingesta automática, métricas del sistema y visualización en Grafana." The "Descripción" section includes a bulleted list of features: "Ingesta automática de alertas desde alert\_json.txt", "Base de datos SQLite con alertas y métricas", "API REST (FastAPI) con documentación Swagger", "Dashboards automáticos en Grafana (con acceso anónimo)", and "Scripts modulares para instalación completa y sin intervención". The "Características principales" section includes a bulleted list: "Despliegue 'one-click' compatible con Raspberry Pi y Ubuntu Server", "Dashboard de Grafana configurado automáticamente usando la variable \${snort}", "API REST para consultar alertas, métricas, reglas y reiniciar Snort", "Servicio Python de ingestión en tiempo real y recolección de métricas del sistema", and "Logrotate + cron configurado por defecto para rotación y backup".

Figura II.1: Ejemplo de documentación y guía de inicio rápido disponible en el repositorio del agente.

Este script instala todas las dependencias, configura Snort 3, la API REST, el sistema de logs y los dashboards de Grafana.

## II.2. Repositorio del Módulo Central (WebApp)

**Repo:** <https://github.com/deianp189/r-snort-central-module-installer>

Para desplegar el módulo central (backend Spring Boot, frontend Angular y servicios asociados) en un servidor Ubuntu Server:

```
1 git clone https://github.com/deianp189/r-snort-central-module-installer.git
2 cd r-snort-central-module-installer/scripts
3 sudo ./run_all.sh
```



Figura II.2: Vista del repositorio y estructura de archivos del módulo central de R-Snort.

Este comando compila e instala automáticamente el backend, el frontend, la base de datos MariaDB, Snort 3 y los dashboards de Grafana, dejando el sistema listo para ser usado.

## II.3. Consejos rápidos y soporte

A continuación, se recopilan algunas recomendaciones prácticas para facilitar la instalación y el uso de R-Snort, especialmente dirigidas a usuarios con experiencia básica en sistemas Linux. También se indican los canales disponibles para obtener soporte o resolver incidencias durante el despliegue:

- Es recomendable ejecutar la instalación como `root` o con `sudo` para evitar problemas de permisos.

- Revisa los archivos README.md de cada repositorio para acceder a detalles avanzados, instrucciones de personalización y resolución de errores frecuentes.
- Para soporte adicional, utiliza el sistema de *issues* o *pull requests* de GitHub, o consulta los enlaces a la documentación técnica que acompañan cada proyecto.



Figura II.3: Ejemplo de funcionalidades destacadas y guía visual en el README de la WebApp.

*Nota:* Los comandos y rutas mostrados son los mínimos necesarios para una instalación básica. Si se desea personalizar el sistema, añadir agentes adicionales o modificar la configuración de reglas y métricas, consulte los apartados correspondientes en los propios repositorios.

---

Esta guía pretende ofrecer la vía más directa para poner en marcha R-Snort, tanto en modo agente como en su módulo centralizado, ayudando al usuario a familiarizarse visualmente con los recursos del repositorio y reduciendo las barreras técnicas para usuarios poco experimentados.



# III. Guía para escribir reglas Snort 3

Este anexo ofrece un *manual de instrucciones* para que cualquier usuario pueda redactar sus propias reglas en Snort 3. Se parte de cero, pero se incluye todo lo esencial para que las reglas resulten funcionales.

## III.1. Estructura general de una regla

En Snort 3 una regla se compone de dos partes:

1. **Cabecera (header)** – define *qué* tráfico interesa.
2. **Opciones (rule options)** – especifican *por qué* ese tráfico debe generar una alerta.

```
1 alert <proto> <src_ip> <src_port> -> <dst_ip> <dst_port> (
2   msg:"Texto descriptivo";
3   content:"cadena o patrón";
4   sid:1000001;
5   rev:1;
)
```

Código III.1: Plantilla mínima de regla Snort 3

## III.2. Cabecera

- **Acción:** alert, log, pass, drop, reject o sdrop.
- **Protocolo:** tcp, udp, icmp, ip, http2, etc.
- **Direcciones/puertos:** pueden ser IP, rangos (192.168.1.0/24), variables (\$HOME\_NET) o any.
- **Dirección de flujo:** -> (unidireccional) o <> (bidireccional).

### Variables habituales

Variable	Uso típico
\$HOME_NET	Red interna protegida
\$EXTERNAL_NET	Todo lo que no sea \$HOME_NET
\$HTTP_PORTS	Puertos 80, 8080, 8000, ...

## III.3. Opciones más comunes

msg Texto que aparecerá en la alerta.

content Cadena (ASCII o hex) que debe encontrarse en el paquete.

`pcre` Expresión regular Perl compatible.

`flow` Dirección–estado de la conexión (`to_server`, `established...`).

`metadata` Etiquetas libres (`service http`, `policy security...`).

`classtype/priority` Agrupación y severidad.

`sid` Identificador único de la regla ( $\geq 1\ 000\ 000$  para reglas locales).

`rev` Versión de la regla (se incrementa con cada cambio).

### III.4. Ejemplo comentado paso a paso

```

1  alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (
2     msg:"FTP anonymous login attempt";
3     flow:to_server,established;
4     content:"USER anonymous";
5     nocase;
6     pcre:"/^USER\s+anonymous\s*/smi";
7     metadata:service ftp, policy security;
8     classtype:attempted-recon;
9     sid:1000002;
10    rev:1;
)

```

Código III.2: Regla que detecta intentos de login FTP anónimos

1. **Cabecera:** tráfico `tcp` externo → interno en el puerto 21.
2. `flow` restringe a sesiones ya establecidas hacia el servidor.
3. `content` y `pcre` confirman la cadena exacta del paquete.
4. `nocase` ignora mayúsculas/minúsculas en `content`.

### III.5. Buenas prácticas

- Reserva rangos de `sid` distintos para tus propias reglas (p. ej. 1 000 001–1 009 999).
- Añade `metadata` con autor, fecha o referencia a CVE.
- Mantén los `rev` consecutivos; nunca reutilices un `sid`.
- Prioriza `flow` y `content` antes de expresiones `pcre` para maximizar el rendimiento.
- Agrupa reglas relacionadas en ficheros temáticos (`local.rules`, `web.rules`, `iot.rules...`).

### III.6. Validación y despliegue

1. Guarda la regla en /etc/snort/rules/local.rules.
2. Ejecuta la comprobación sintáctica:

```
1 sudo snort -c /etc/snort/snort.lua --dry-run
```

3. Si no hay errores, reinicia Snort:

```
1 sudo systemctl restart snort
```

### III.7. Recursos recomendados

- Documentación oficial Snort 3 – <https://docs.snort.org>
- Emerging Threats Rules – <https://rules.emergingthreats.net>
- Guía de desarrollo de reglas (Cisco Talos) – <https://talosintelligence.com>

Con estos fundamentos, el lector dispone de todo lo necesario para comenzar a crear reglas personalizadas en Snort 3 y fine-tunear su IDS de acuerdo con los requisitos de su entorno.





## Resumen/Abstract

La protección efectiva de redes domésticas y pequeñas empresas sigue siendo una asignatura pendiente por la complejidad y el coste de las soluciones tradicionales. Este Trabajo de Fin de Grado presenta una plataforma integral compuesta por un agente ligero y una aplicación web centralizada, diseñada para facilitar la detección y gestión de intrusiones sin requerir experiencia técnica.

El sistema automatiza la instalación, permite monitorizar múltiples sensores desde una interfaz intuitiva, y ofrece almacenamiento y visualización avanzada de alertas mediante dashboards personalizados.

La integración de procesos automáticos y la compatibilidad con hardware asequible garantizan una solución fiable y accesible, orientada a democratizar la ciberseguridad en entornos con recursos limitados.

Effective protection of home and small business networks remains a challenge due to the complexity and cost of traditional solutions. This Final Degree Project introduces a comprehensive platform combining a lightweight agent and a centralized web application, designed to simplify intrusion detection and management without requiring technical expertise.

The system automates installation, enables monitoring of multiple sensors through an intuitive interface, and offers advanced alert storage and visualization with customize dashboards.

The integration of automated processes and compatibility with affordable hardware ensure a reliable and accessible solution, aiming to democratize cybersecurity in resource-constrained environments.