



GRADO EN INGENIERÍA DE SISTEMAS
AUDIOVISUALES Y MULTIMEDIA

Curso Académico 2018/2019

Trabajo Fin de Grado

SISTEMA DE ALERTING PARA
ARQUITECTURA DE PREVENTIVOS

Autor : Javier Baos Mora

Tutor : Dr. Gregorio Robles

Trabajo Fin de Grado

Sistema de Alerting para Arquitectura de Preventivos

Autor : Javier Baos Mora

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de
de 2019, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2019

“The first step is to establish that something is possible then probability will occur.”
- Elon Musk

*Dedicado a
mi familia y futura mujer*

Agradecimientos

Agradezco a mi madre, mi padre y hermanos por la paciencia y confianza depositada en mí a lo largo de este largo y duro camino.

Agradezco a mi futura mujer por su perseverancia e inteligencia, por estar siempre ayudándome a quitar cada piedra del camino.

Agradezco a mis abuelas por sentirse siempre orgullosas.

Agradezco a mis compañeros de biblioteca por amenizar las tardes de verano e invierno.

Agradezco a mis compañeros de trabajo por ser siempre un pozo sin fondo de conocimiento.

Por último, y no menos importante. Agradezco a mi tutor por ayudarme a escribir el final de este capítulo.

Resumen

En este proyecto se presenta una solución de mantenimiento de red, con el objetivo de evitar o mitigar las consecuencias de los fallos de un equipo, logrando en algunos casos prevenir las incidencias antes de que estas sucedan. La solución es perfectamente aplicable a cualquier capa de red y ha sido probada en la realidad.

El sistema está basado en tres módulos: base de datos, capa de visualización y módulo de alertas. Se probaron distintas soluciones software para cada módulo, optándose finalmente por: Graphite como base de datos, Grafana en la capa de visualización y Gralarmas (desarrollo propio en Django) como sistema de alertas.

Este proyecto surgió ante la necesidad de la empresa Orange de evolucionar hacia un sistema de monitorización flexible, centralizada, basada en el tiempo real y escalable. En la actualidad, el sistema está en continuo desarrollo añadiéndose nuevas funcionalidades.

Summary

This project presents a network maintenance solution, with the aim of avoiding or mitigating the consequences of equipment failures, achieving in some cases to prevent incidents before they happen. The solution is perfectly applicable to any network layer and has been tested in reality.

The system is based on three modules: database, visualization layer and alert module. Different software solutions for each module were tested and finally opted for: Graphite as database, Grafana in the visualization layer and Gralarmas (own development in Django) as alert system.

This project was born out of Orange's need to evolve towards a flexible, centralized, real-time-based and scalable monitoring system. At present, the system is in continuous development adding new functionalities.

Índice general

1. Introducción y motivación	1
1.1. Contexto	2
1.2. Open Source	2
1.3. Estructura de la memoria	3
2. Objetivos	5
2.1. Objetivo general	5
2.2. Objetivos específicos	5
2.3. Planificación temporal	6
3. Estado del arte	9
3.1. Graphite	9
3.2. Grafana	9
3.3. Django	10
3.4. Python	11
3.5. Javascript	12
3.6. HTML5	13
3.7. CSS3	14
3.8. MySQL	15
3.9. API	16

3.10. Bootstrap	17
4. Diseño, implementación y Desarrollo	19
4.1. Hardware	19
4.2. Software	22
4.2.1. Graphite	25
4.2.2. Grafana	25
4.2.3. Gralarmas	26
4.2.4. Backup	29
5. Resultados	31
6. Conclusiones	41
6.1. Consecución de objetivos	41
6.2. Aplicación de lo aprendido	42
6.3. Lecciones aprendidas	42
6.4. Trabajos futuros	43
Bibliografía	45

Índice de figuras

2.1. Cronología	7
2.2. Diagrama DevOps	8
3.1. Estructura Graphite	10
3.2. Aspecto Grafana	10
3.3. Esquema interno Django	11
3.4. Ejemplo código python	12
3.5. Ejemplo código Javascript	13
3.6. Estructura semántica HTML5	14
3.7. Formato reglas CSS	15
3.8. Ejemplo de conexiones API	16
3.9. Plantilla Bootstrap	17
4.1. DELL PowerEdge R630	19
4.2. Esquema de servidores	21
4.3. Arquitectura	24
4.4. Clase Carbon	25
4.5. Configuración Graphite en Grafana	26
4.6. Estructura Base de Datos	27
4.7. Models Django	28
4.8. Diagrama Lógico	28

5.1. urls.py	32
5.2. Dashboard	33
5.3. Selección Dashboard	33
5.4. Selección Panel	34
5.5. Selección Metrica	34
5.6. Combo Umbrales	35
5.7. Listado métricas configuradas	35
5.8. Vista principal templates	36
5.9. Combo parámetros templates	37
5.10. Vista del configurador de alarmas	38
5.11. Vista de histórico alarmas	39
5.12. forms.py	40

Capítulo 1

Introducción y motivación

Ejerciendo de becario en la empresa Orange España a media jornada en los departamentos de Despliegue Fijo y Planificación durante un año me ofrecieron una nueva beca con objetivos más ambiciosos en el departamento de Operación y Mantenimiento. En la entrevista simplemente me indicaron que las capacidades que gustaban de mi eran los conocimientos sobre Python y Django, eso me hacía tener una idea de cuál sería mi función (anteriormente realizaba funciones de seguimiento económico) y acepte el nuevo reto. Novedoso y revolucionario, se planteaba una nueva forma de mantener la red.

A mi llegada, se empleaban diversas herramientas de monitorización de diferentes *vendors*. Estas eran cerradas, con poca escalabilidad y escasa utilidad en tiempo real. Ante una incidencia surgía la necesidad de una herramienta ágil, la cual permitiera generar indicadores de gestión (KPIs) o monitorizar distintos sistemas en momentos concretos sin dependencias externas. En paralelo se estaba montando una solución a todos estos problemas basada en herramientas Open Source llamadas Graphite y Grafana, con una carencia, la capa de *alerting*. Cubrir esta carencia e implantar dicha solución global era mi cometido.

1.1. Contexto

Orange se caracteriza por tener una red muy compleja debido a las adquisiciones de distintas empresas (ya.com, amena, Jazztel, Uni2. . .) y, en consecuencia, sus redes. Por tanto, nos encontramos con una red muy heterogénea, multitud de equipos y *vendors* con sus diferentes elementos de gestión y monitorización.

Cada capa de red e incluso dentro de una misma capa se monitoriza de forma diferente y con un elemento distinto. Esto dificultaba la mucho el *troubleshooting* entre capas, una visión E2E (End to End) de servicio e incluso los *post-mortem* de las incidencias.

La unificación y centralización de toda la información permitirá correlar alarmas de diversas capas de red sacando KPIs de servicio. Cuando se identifique una degradación, se generarán alertas preventivas.

1.2. Open Source

Sin presupuesto para el proyecto, más allá de mi contratación como becario, planteo una solución desarrollada internamente basada en código abierto (Open Source). El código abierto es un modelo de desarrollo de software basado en la colaboración abierta. Se enfoca más en los beneficios prácticos (acceso al código fuente) que en cuestiones éticas o de libertad que tanto se destacan en el software libre. Para muchos el término libre hace referencia al hecho de adquirir un software de manera gratuita. Además, el hecho de estar respaldado por una comunidad garantiza la continua evolución de las herramientas de una forma ágil.

Las herramientas finalmente utilizadas fueron:

- Graphite: Fue diseñado y escrito originalmente por Chris Davis en Orbitz en 2006 como un proyecto paralelo que finalmente se convirtió en su herramienta de mo-

nitorización fundamental. En 2008, Orbitz permitió que Graphite se publicara con la licencia de código abierto Apache 2.0.

- Grafana: Sus *slogans* “The open platform for beautiful analytics and monitoring” y “The leading open source software for time series analytics” dan indicios sobre su filosofía. Actualmente es la herramienta de monitorización y análisis de *time-series* por excelencia.
- Django: Framework de código abierto escrito en Python.

1.3. Estructura de la memoria

En esta sección se introduce la estructura de la memoria:

- En el primer capítulo 1 se hace una introducción del proyecto, poniendo en contexto y profundizando en la tecnología Open Source sobre la que esta basada el proyecto.
- En el capítulo 2 se muestra el objetivo principal del proyecto y los específicos sobre los que se basó.
- A continuación se presenta el estado del arte 3, en el que se describen las tecnologías que utilizadas.
- Después se muestra el diseño, la implementación y el desarrollo 4 de la solución.
- En el capítulo 5 se incluyen los resultados del trabajo fin de grado.
- Para finalizar, se sintetizan los puntos más relevantes del proyecto en la conclusión.

Capítulo 2

Objetivos

2.1. Objetivo general

Mi trabajo fin de grado consiste en montar una solución de monitorización para la capa de PS (Packet Switching) extensible a otras capas, escalable e intuitivo cara al usuario en el tiempo límite de duración de la beca (3 meses).

2.2. Objetivos específicos

Al inicio, el proyecto se dividió en subobjetivos tangibles, alcanzables, sintiéndome con ganas, motivación y herramientas para llevarlo a cabo:

1. Decidir, instalar y configurar Hardware. Decidir el hardware a utilizar, instalar en el CPD (Centro de Procesamiento de Datos) correspondiente y configurar con los requisitos de los aplicativos.
2. Decidir aplicativos de monitorización. Sondear en el mercado las herramientas de monitorización punteras y elegir la solución que mejor se adapte a las necesidades de proyecto.

3. Instalar y poner a punto las herramientas en los nuevos servidores. Instalar y configurar de forma óptima los aplicativos seleccionados en los servidores instalados.
4. Conectar y extraer la información de Graphite y Grafana. Decidir en que entorno se monta el sistema de alertas y sus interfaces con las dos fuentes de información. Realizar análisis de conectividades necesarias, interfaces, formatos de entrada/salida, etc.
5. Definir la base de datos. Se conoce como base de datos al conjunto de informaciones que está organizado y estructurado de un modo específico para que su contenido pueda ser tratado y analizado de manera rápida y sencilla. Es uno de los puntos más importantes para que el usuario puede encontrar aquello que busca con facilidad, a diferencia de lo que le sucedería si todos los datos estuvieran mezclados y sin ningún tipo de orden. Centrándonos en la velocidad de consulta y la flexibilidad para expandirlas y relacionar las tablas.
6. Definir la lógica de la herramienta. Acotar la funcionalidades de la herramienta, tener claro el para, el qué y el cómo.
7. Diseñar gráficamente la aplicación. Decidir el diseño gráfico buscando siempre una aplicación sencilla, rápida e intuitiva.
8. Poner en marcha la aplicación. Transferir a mantenimiento, procedimientos de backup y sesiones de formación de uso de la herramienta. Tiene una dificultad adicional, Orange cuenta con una media de edad alta en la plantilla poco habituada a utilizar este tipo de aplicativos, entra un proceso de cambio cultural.

2.3. Planificación temporal

En la realización del proyecto se ha seguido la siguiente cronología, dividida en

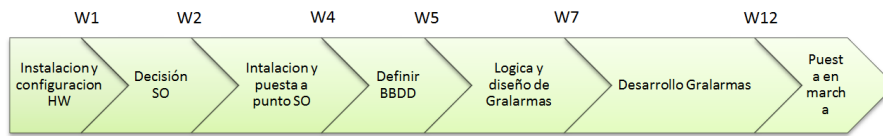


Figura 2.1: Cronología

semanas, compaginando trabajo y TFG han sido necesarios fines de semana para realizar los hitos en las fechas establecidas al inicio del mismo:

En la primera semana se instaló y configuró el hardware escogido, la fase más sencilla debido a que ya se poseía conocimiento del hardware utilizado (reutilizado) y se habían instalado mas servidores con dichas características y sistemas operativos. Después se decidieron las aplicaciones que iban a formar el sistema de monitorización, haciendo un análisis de las herramientas Open Source más potentes del mercado. A continuación, se instalaron y configuraron las herramientas Graphite y Grafana en un periodo de dos semanas seguido de la configuración de la base de datos. Llegados a estas fechas, se iniciaron las tareas principales y mas complejas, la creación de la lógica de la herramienta, el diseño de la misma y su posterior desarrollo, esta etapa duró aproximadamente mes y medio. Para finalizar, comenzó la puesta en marcha del sistema en la que aún continuamos aplicando un sistema DevOps.

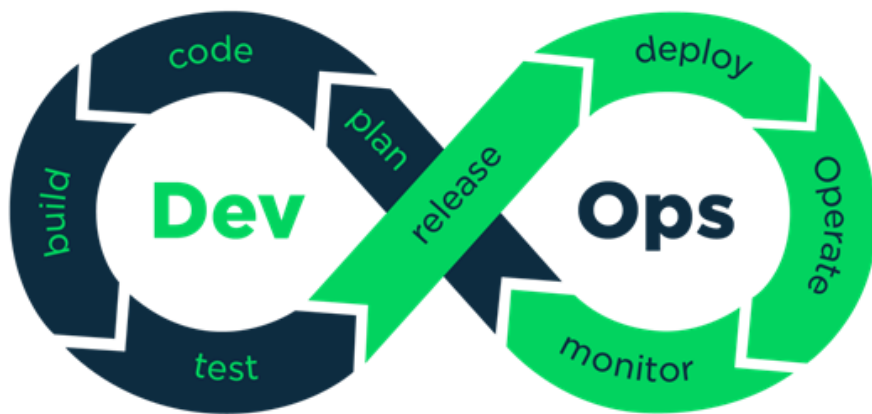


Figura 2.2: Diagrama DevOps

Capítulo 3

Estado del arte

3.1. Graphite

Graphite es una herramienta de monitorización válida tanto para infraestructuras físicas o cloud. Graphite se utiliza para realizar un seguimiento del rendimiento de sitios web, aplicaciones, servicios y servidores de red. Graphite marcó el inicio de una nueva generación de herramientas de monitorización, haciendo más fácil almacenar, recuperar, compartir y visualizar datos de *time-series*.

Graphite consiste en tres componentes software:

- Carbon: Servicio de alto rendimiento que escucha datos de *time-series*.
- Whisper: Biblioteca de base de datos simple para almacenar datos de *time-series*.
- Graphite-web: Interfaz de usuario y API para representar gráficos y paneles.

3.2. Grafana

Grafana es un software libre basado en licencia de Apache 2.0, que permite la visualización y el formato de métricas. Permite crear cuadros de mando y gráficos a partir de

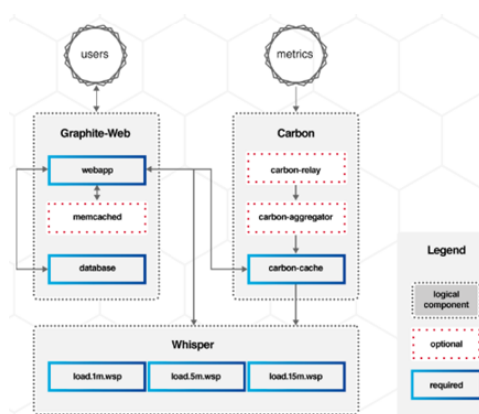


Figura 3.1: Estructura Graphite

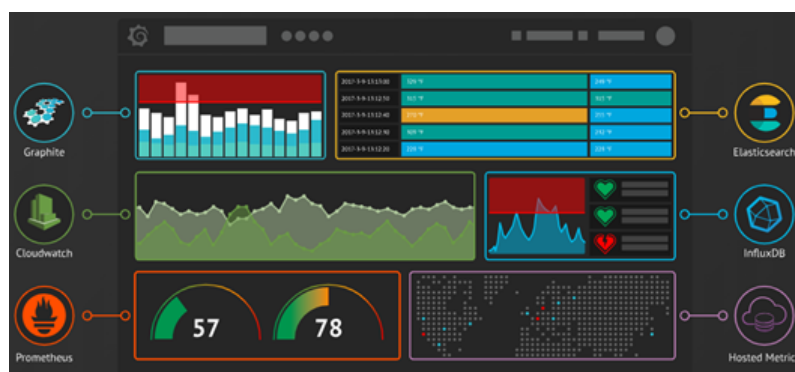


Figura 3.2: Aspecto Grafana

múltiples fuentes, incluidas bases de datos de time-series como Graphite o InfluxDB.

Es una herramienta multiplataforma sin ninguna dependencia. Está escrito en lenguaje Go y tiene un HTTP API completo. Ofrece al usuario multitud de posibilidades de forma muy intuitiva.

3.3. Django

Django es un framework de aplicaciones web gratuito y de código abierto (Open Source) escrito en Python basado en el diseño Modelo-Vista-Template.

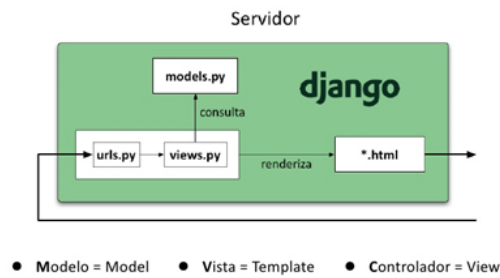


Figura 3.3: Esquema interno Django

Diseñado para ayudar a los desarrolladores a llevar las aplicaciones desde el concepto hasta su finalización lo más rápido posible. Evitando muchos errores comunes de seguridad y ofreciendo una escalabilidad rápida y flexible.

Otras características destacables son:

- Un mapeador objeto-relacional.
- Aplicaciones "enchufables" que pueden instalarse en cualquier página gestionada con Django.
- Una API de base de datos robusta.
- Un sistema incorporado de "vistas genéricas" que ahorra tener que escribir la lógica de ciertas tareas comunes.
- Un sistema extensible de plantillas basado en etiquetas, con herencia de plantillas.
- Un despachador de URLs basado en expresiones regulares.
- Un sistema "middleware" para desarrollar características adicionales.

3.4. Python

Lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis

```

124     for db_name in lista_dashboards:
125         url = http + '/api/dashboards/db/' + db_name
126         respuesta_metricas = requests.get(url, headers=headers)
127         lista_metricas = respuesta_metricas.json()
128         numero_variables = 0
129         if 'dashboard' in lista_metricas:
130             metricas_en_json = lista_metricas['dashboard']
131
132             # Primero sacamos la variables de templating
133             if 'templating' in metricas_en_json:
134                 # imprime toda la variable de templating (util durante el desarrollo)
135
136                 variable_template = {}
137                 for variables in metricas_en_json['templating']['list']:
138                     nombre_variable = variables['name']
139                     lista_opciones = []
140                     for opciones in variables['options']:
141                         lista_opciones.append(opciones['text'])
142                     variable_template[nombre_variable] = lista_opciones
143
144                 for variables_individuales in variable_template:
145                     numero_variables += 1
146                     lista_variables.append(db_name + "---->" + variables_individuales)

```

Figura 3.4: Ejemplo código python

que favorezca un código legible haciendo más sencilla la colaboración. Soporta orientación a objetos, usa tipado dinámico y es multiplataforma. Es administrado por la Python Software Foundation y posee una licencia de código abierto, denominada Python Software Foundation License.

Python es un lenguaje de programación multiparadigma, permite varios estilos:

- Programación orientada a objetos.
- Programación imperativa.
- Programación funcional.

3.5. Javascript

Lenguaje de programación interpretado, orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Implementado principalmente como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

```

1  /*global gettext*/
2  (function($) {
3      'use strict';
4      $(document).ready(function() {
5          // Add anchor tag for Show/Hide link
6          $('#fieldset.collapse').each(function(i, elem) {
7              // Don't hide if fields in this fieldset have errors
8              if ($(elem).find('div.errors').length === 0) {
9                  $(elem).addClass("collapsed").find("h2").first().append(' <a id="fieldsetcollapse" +
10                      i + " class="collapse-toggle" href="#">' + gettext("Show") +
11                      '</a>');
12              }
13          });
14          // Add toggle to anchor tag
15          $('#fieldset.collapse a.collapse-toggle').click(function(ev) {
16              if ($(this).closest("fieldset").hasClass("collapsed")) {
17                  // Show
18                  $(this).text(gettext("Hide")).closest("fieldset").removeClass("collapsed").trigger("show.fieldset", [$(this).attr("id")]);
19              } else {
20                  // Hide
21                  $(this).text(gettext("Show")).closest("fieldset").addClass("collapsed").trigger("hide.fieldset", [$(this).attr("id")]);
22              }
23              return false;
24          });
25      });
26  })(django.jQuery);
27

```

Figura 3.5: Ejemplo código Javascript

Diseñado con una sintaxis similar a C. Aunque adopta nombres y convenciones del lenguaje de programación Java, tienen semánticas y propósitos diferentes. Todos los navegadores interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del DOM (Document Object Model).

Utilizado en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. Actualmente es ampliamente utilizado para enviar y recibir información del servidor junto con ayuda de otras tecnologías como AJAX. JavaScript se interpreta en el agente de usuario al mismo tiempo que las sentencias van descargándose junto con el código HTML.

3.6. HTML5

Quinta revisión del lenguaje de marcado básico de la web. HTML5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos.

Entre sus nuevas funcionalidades destacan:



Figura 3.6: Estructura semántica HTML5

- Incorpora etiquetas (canvas 2D y 3D, audio, vídeo) con codecs para mostrar los contenidos multimedia.
- Etiquetas para manejar grandes conjuntos de datos: Datagrid, Details, Menu y Command. Permiten generar tablas dinámicas que pueden filtrar, ordenar y ocultar contenido en cliente.
- Mejoras en los formularios.
- Drag & Drop. Nueva funcionalidad para arrastrar objetos como imágenes.

3.7. CSS3

Lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado.

Diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este, características tales como las capas, los colores y



Figura 3.7: Formato reglas CSS

las fuentes. Esta separación busca mejorar la accesibilidad, proveer más flexibilidad y control en la especificación de características de presentación, permitir que varios documentos HTML compartan un mismo estilo usando una sola hoja de estilos separada en un archivo .css, y reducir la complejidad y la repetición de código en la estructura del documento.

La especificación CSS (Cascading Style Sheets) describe un esquema prioritario para determinar qué reglas de estilo se aplican si más de una regla coincide para un elemento en particular. Estas reglas son aplicadas en cascada, de modo que las prioridades son calculadas y asignadas a las reglas.

3.8. MySQL

Sistema de gestión de bases de datos relacional. Desarrollado en su mayor parte en ANSI C y C++, MySQL es usado por muchos sitios web grandes y populares.

Permite a los desarrolladores y diseñadores, realizar cambios en sus sitios de manera simple, con tan sólo cambiar un archivo, evitando tener que modificar todo el código web. Esto se debe a que trabaja con un sistema centralizado de gestión de datos, que permite realizar cambios en un solo archivo y que se ejecuta en toda la estructura de datos que se comparte en la red.



Figura 3.8: Ejemplo de conexiones API

3.9. API

Interfaz de programación de aplicaciones, es un conjunto de rutinas que provee acceso a funciones de un determinado software.

Una API representa la capacidad de comunicación entre componentes de software. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación, generalmente entre los niveles o capas inferiores y los superiores del software.

Uno de los principales propósitos de una API es proporcionar un conjunto de funciones de uso general. De esta forma, los programadores se benefician de las ventajas del API haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio. Las API asimismo son abstractas: el software que proporciona una cierta API generalmente es llamado la implementación de esa API.



Figura 3.9: Plantilla Bootstrap

3.10. Bootstrap

Biblioteca de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales.

Bootstrap es compatible con la mayoría de los navegadores web. Existe un concepto de compatibilidad parcial que hace disponible la información básica de un sitio web para todos los dispositivos y navegadores. Por ejemplo, las propiedades introducidas en CSS3 para las esquinas redondeadas, gradientes y sombras son usadas por Bootstrap a pesar de la falta de soporte de navegadores antiguos. Esto extiende la funcionalidad de la herramienta, pero no es requerida para su uso.

Bootstrap es de código abierto y está disponible en GitHub. A diferencia de muchos frameworks web, solo se ocupa del desarrollo front-end.

Capítulo 4

Diseño, implementación y Desarrollo

4.1. Hardware

Por motivos económicos, se decide reutilizar el hardware de otra plataforma formada por tres servidores DELL PowerEdge R630.

Cuyas características técnicas se adecuan a las necesidades del proyecto:

- 10 HDD.
- Intel® Xeon® E5-2620 v4 2.1GHz,20M Cache,8.0GT/s QPI,Turbo,HT,8C/16T (85W) Max Mem 2133MHz.
- 2 CPU hasta 120W.
- 2666MT/s RDIMMs.



Figura 4.1: DELL PowerEdge R630

- 32GB RDIMM 2666MT/s Dual Rank.
- Medida: 8x2.5".Cuyas características técnicas se adecuan a las necesidades del proyecto:

Otros detalles destacados que ofrece el hardware de DELL son:

- Potencia de procesamiento: la generación más reciente de procesadores Intel® Xeon® da impulso a los entornos virtualizados y a las aplicaciones de la empresa que necesitan el máximo rendimiento.
- Memoria de gran capacidad y bajo consumo: la memoria DDR4 acelera las cargas de trabajo tales como la planificación de recursos para empresas y las aplicaciones de base de datos.
- Almacenamiento local ampliable y eficiente: la amplia variedad de opciones de almacenamiento en el servidor permite configuraciones solo flash, soluciones híbridas con asignación de niveles de fábrica, plataformas de bajo coste con capacidad de gran densidad y un almacenamiento adecuado basado en servidor con todo lo que las aplicaciones necesitan para ofrecer el mejor valor y rendimiento.
- Gestión simplificada e inteligente: las herramientas de Dell OpenManage permiten empezar a producir en menos tiempo gracias al acceso local mejorado, nuevos dispositivos móviles para la supervisión segura del centro de datos desde dispositivos manuales y nuevos procesos automatizados que ahorran tiempo y dinero en las tareas cotidianas de operación y mantenimiento.
- Eficiencia energética: las tecnologías de alimentación y refrigeración innovadoras como Dell Fresh Air 2.0 permiten que los centros de datos funcionen a temperaturas constantes de hasta 40 °C/104 °F.

Partiendo de estos 3 servidores, se decide implementar la siguiente arquitectura física:

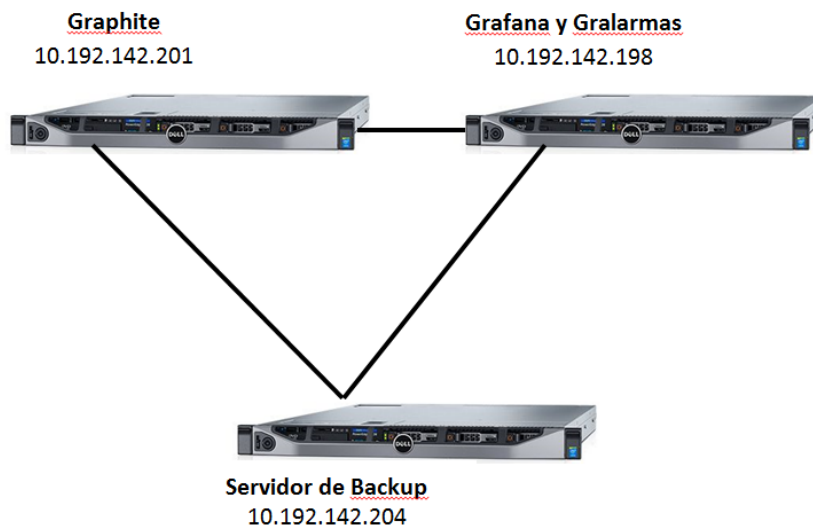


Figura 4.2: Esquema de servidores

- Un servidor para realizar backups.
- Un servidor para la base de datos de métricas (Graphite).
- Un servidor para la visualización y alerting (Grafana y Gralarmas).

Todos ellos configurados en RAID 1 (mirror) duplicando en espejo todos los datos de cada unidad de forma sincronizada a una unidad de duplicación exacta. De modo que, si se produce algún fallo o avería en alguna de las unidades, no se pierde ningún dato.

El sistema ofrece ambos tipos de escalabilidad:

- Escalabilidad vertical: añadir más recursos a un nodo particular del sistema mejora en conjunto.
- Escalabilidad horizontal: agregar más nodos mejora el rendimiento.

4.2. Software

El primer paso fue la decisión de los elementos software que cubrirán cada función:

- Solución almacenamiento: Como en todas las soluciones, se buscan entre todos los productos open source. Se compararon Elasticsearch, InfluxDB, Graphite y Prometheus.

Inicialmente se descartaron Elasticsearch y Prometheus por funcionalidad, Elasticsearch no está enfocado a análisis de *time-series* sino a logs y Prometheus requiere de la instalación de un agente en los equipos remotos para la extracción de la información algo poco recomendado por algunos vendedores. El debate surgió entre InfluxDB y Graphite, a pesar de que InfluxDB presentaba mejores prestaciones me decanté por Graphite debido a que se cubrían los requisitos de rendimiento y al mayor conocimiento interno de la herramienta, esto facilitará mucho la fase de puesta en marcha y traspaso de conocimiento.

- Solución capa de visualización: Se compararon las tres herramientas Open Source más potentes del mercado: Prometheus, Graphite y Grafana.

Finalmente, Grafana fue la elegida debido a la multitud de opciones, su sencillez y su curva de aprendizaje.

- Solución alerting: Debido a la carencia de grafana en el sistema de alerting (conocida en la decisión de la solución de la capa de visualización) se decidió desarrollar una aplicación interna basada en Django por conocimiento del framework.

A continuación, se identificaron los flujos de datos y se decidieron los formatos de comunicación entre los distintos elementos de la solución:

	Elasticsearch	InfluxDB	Graphite	Prometheus
Licencias	Open Source	Open Source	Open Source	Open Source
Intrusivo	Posibilidad de instalar agentes	No	No	Necesidad de instalar agente
Escalabilidad	Sí	Sí	Sí	Si
Tecnologías	Lucene, Java	Go, TSM	Python	Go
APIs y otros métodos de acceso	RESTful APIs, JSON API	HTTP API, JSON over UDP	HTTP API, JSON over UDP	RESTful, HTTP/JSON API
Formato de datos	Logs, Time-Series	Time-Series	Time-Series	Time-Series
Métodos de particionamiento	Sharding	Sharding	-	Sharding
Métodos de replicación	Factor de replicación seleccionable	Factor de replicación seleccionable	-	Si
Cumple performance de ingesta	Sí	Sí	Sí	Si
Conocimiento interno de la herramienta	-	-	Alto	-
Soporte	Sí	Sí	No	No

Cuadro 4.1: Comparativa Solución Almacenamiento

	Grafana	Prometheus	Graphite
Compatibilidad con Graphite	Sí	Sí	Si
Aspecto	Sobresaliente	Notable	Pobre
Plugins	Amplio abanico	Amplio abanico	Pobre
Configuración “user friendly”	Sí	No	No

Cuadro 4.2: Comparativa Solución Capa Visualización

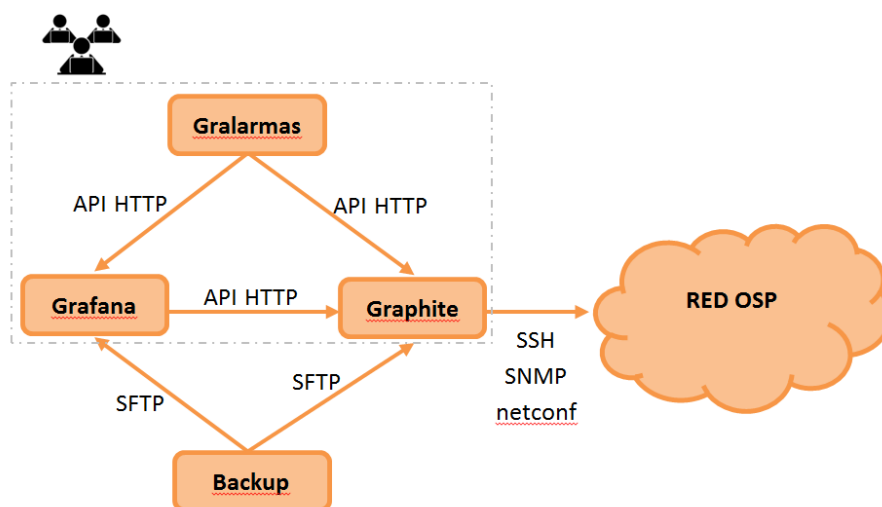


Figura 4.3: Arquitectura

```

class Carbon:
    def __init__(self, host_='10.110.212.69'):
        self.DELAY_GRAPHITE = 60
        self.CARBON_SERVER = host
        self.CARBON_PORT = 2003
        self.socket = socket(AF_INET, SOCK_STREAM)
        self.connect()

    def connect(self):
        try:
            self.socket.connect((self.CARBON_SERVER, self.CARBON_PORT))
        except IOError, e:
            print ("No se puede conectar a (%s) en el puerto (%d), %s,\n"
                  "carbon-cache.py corriendo?" %
                  (self.CARBON_SERVER, self.CARBON_PORT, str(e)))
            return
        except Exception as e:
            print ("No se puede conectar a (%s) en el puerto (%d), %s,\n"
                  "carbon-cache.py corriendo?" %
                  (self.CARBON_SERVER, self.CARBON_PORT, str(e)))
            logging.exception(e)
            return

    def disconnect(self):
        self.socket.close()

    def envia_metrice(self, metric_string):
        """ metric = 'metric_string value timestamp'
        Example:
        metric = 'foo.bar.baz 42 %d\n' % int(time())
        graphiteCarbon.send_metric(metric)
        """
        try:
            self.socket.sendall(metric_string + "\n")
        except:
            self.connect()
            self.socket.sendall(metric_string + "\n")

```

Figura 4.4: Clase Carbon

4.2.1. Graphite

Una vez claro el diseño, se procedió a la implementación de la solución de los distintos elementos. En el servidor de Graphite, además del aplicativo, se almacenarán scripts desarrollados en Python. Dichos scripts tendrán la función de conectarse a la red con diversos protocolos (ssh, netconf, snmp, ...) para extraer información. La complejidad de los scripts dependerá del protocolo y post procesado usado para cada elemento. Los scripts harán uso de una clase 4.4, mostrada en la captura posterior, desarrollada para conectarse al carbon y almacenar en formato *time-series* el valor deseado en los whisper.

4.2.2. Grafana

La capa de visualización de Graphite (graphite-web) es pobre en funcionalidades y poco intuitiva, por ello se decidió utilizar Grafana. En el mismo se configura el acceso

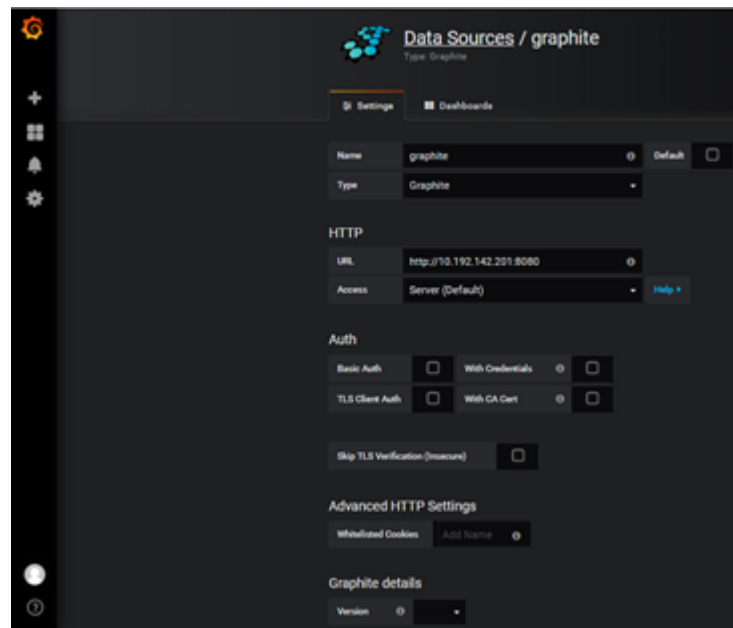


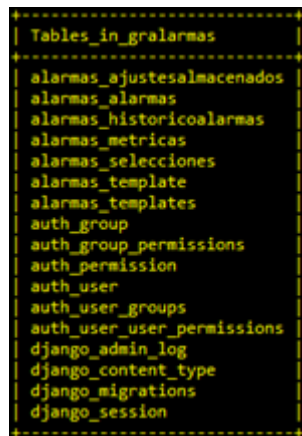
Figura 4.5: Configuración Graphite en Grafana

a la base de datos (Graphite) de forma sencilla una vez abierta la conectividad entre los servidores.

4.2.3. Gralarmas

Para cubrir la carencia de Grafana en el sistema de alertas se desarrolla una web en interno. La cual se conecta mediante API HTTP a Grafana para extraer el formato de la métrica, los dashboards, los paneles,... y a Graphite para, con la información extraída de Grafana, consultar el valor de la métrica.

En la fase de lógica y diseño decidí que el aplicativo permitirá: Configurar alarmas sobre las métricas, crear templates para agilizar aún más la definición de alarmas, listar las alarmas configuradas, editar ajustes sobre las alarmas, mantener un histórico de alarmas, editar algunos ajustes del aplicativo y un dashboard en el que se muestran las alarmas activas.



Tables in gralarmas
alarmas_ajustesalmacenados
alarmas_alarmas
alarmas_historicoalarmas
alarmas_metricas
alarmas_selecciones
alarmas_template
alarmas_templates
auth_group
auth_group_permissions
auth_permission
auth_user
auth_user_groups
auth_user_user_permissions
django_admin_log
django_content_type
django_migrations
django_session

Figura 4.6: Estructura Base de Datos

Los datos extraídos de ambas herramientas se almacenan en una base de datos MYSQL (models django) con la siguiente estructura:

- alarmas_ajustesalmacenados: Donde se almacenan los ajustes configurados.
- alarmas_alarmas: Donde se almacenan las alarmas activas.
- alarmas_historicoalarmas: Donde se almacenan las alarmas cesadas.
- alarmas_metricas: Donde se almacenan las métricas configuradas con los umbrales.
- alarmas_selecciones: Relacionada con alarmas_metricas. Algunas métricas tienen opciones seleccionables, dichas opciones se almacenan en esta tabla.
- alarmas_template: Donde se almacena el nombre de los templates.
- alarmas_templates: Relacionada con alarmas_template. Donde se almacenan los ajustes de cada template.

En segundo plano, se aplica una lógica de chequeo de estados de las alarmas en la que se comprobará cada 5 minutos si una métrica es alarmada siguiendo el siguiente diagrama:

```
9      +class Metricas(models.Model):...
25
26
27      +class Template(models.Model):...
32
33
34      +class Templates(models.Model):...
41
42
43      +class TemplatesForm(ModelForm):...
47
48
49      +class Alarmas(models.Model):...
78
79
80      +class Selecciones(models.Model):...
85
86
87      +class HistoricoAlarmas(models.Model):...
98
99
100     +class AjustesAlmacenados(models.Model):...
104
```

Figura 4.7: Models Django

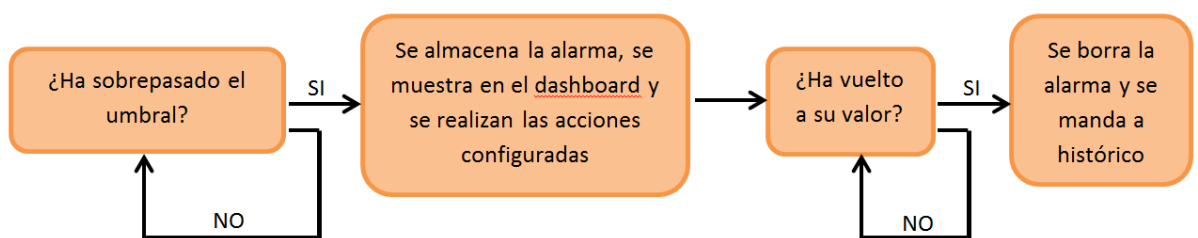


Figura 4.8: Diagrama Lógico

4.2.4. Backup

Se establecen en el servidor de backup procesos en el crontab de la máquina para hacer backups diarios a las 00:00 de la base de datos MySQL, de los archivos de configuración de los aplicativos Graphite y Grafana, de los whisper y del repositorio de scripts python.

Capítulo 5

Resultados

Los entornos de Grafana y Graphite se han estabilizado y se ha realizado un continuo proceso de *fine tuning* de los aplicativos debido al incremento notable de métricas configuradas, el uso de los usuarios y nuevas necesidades provenientes de la web Gralarmas.

El aspecto final de la web de Gralarmas es una interfaz sencilla e intuitiva que cubre las necesidades del cliente. El diseño base consta de una barra de navegación horizontal (*navbar*) con la versión actual de software, el acceso a la administración de la web y un acceso a la ayuda; un panel de navegación a la izquierda (*side-menu*) en la que se muestra el acceso al dashboard, añadir métrica/alarma, metricas configuradas, templates, configurar alarmas, historial de alarmas y ajustes; y una zona central con el contenido dependiendo de la selección del menú.

Cada opción del menú se implementa en el *ulrs.py* un enlace y sus posibles extensiones, asociado a su vista correspondiente.

En la página principal se muestra un *Dashboard* distribuido como se ve en la imagen adjuntada a continuación. La parte superior es un resumen de las métricas configuradas (cuadrado verde), las alarmas warning (cuadrado azul), mayor (cuadrado amarillo) y criticas (cuadrado rojo). La parte inferior muestra un listado de las alarmas activas caracte-

```

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^gralarmas/dashboards/(.*)/panel/(.*)/alarmas_umbral/exportar/(.*)/', exportar_alarmas),
    url(r'^gralarmas/dashboards/(.*)/panel/(.*)/alarmas_umbral/', alarmas_umbral),
    url(r'^gralarmas/dashboards/(.*)/panel/(.*)/', metricas),
    url(r'^gralarmas/dashboards/(.*)/', paneles),
    url(r'^gralarmas/', index),
    url(r'^alarmero/', alarmero),
    url(r'^dashboard/', dashboard),
    url(r'^historico/(.*)/', historico_dashboard),
    url(r'^historico/', historial_alarmas),
    url(r'^templates/(.*)/editar_template/(.*)/', editar_templates),
    url(r'^templates/(.*)/', templates),
    url(r'^templates/', templates),
    url(r'^alarmas/(.*)/edicion_umbrales/', editar_umbrales),
    url(r'^alarmas/(.*)/OVO', alarmas_OVO),
    url(r'^alarmas/(.*)/', alarmas_dashboard),
    url(r'^alarmas/', comprobar_alarmas),
    url(r'^ajustes/', configurar),
    url(r'^ayuda/', ayuda),
]

```

Figura 5.1: urls.py

rizadas por: dashboard de grafana, nombre de la métrica, ultimo valor, estado (warning, mayor o critical) y la hora y fecha de inicio de la alarma. Finalmente, en el centro se mostrara la gráfica de la alarma que se seleccione o en su defecto la primera alarma, en el caso de no existir alarma como sucede en este caso, se muestra un error 404.

La segunda opción es “Añadir métrica/alarma”, aquí se configurará la alarma en varios pasos: se seleccionará el dashboard equivalente a Grafana; se seleccionará el panel en el que se encuentra la métrica que queremos alarmar; se seleccionará la métrica/métricas que queremos configurar, variables si fueran necesarias y el tipo de alarma que queremos, si ya existiera un umbral configurado sobre esa métrica se mostraría un tick en la columna correspondiente. Una vez seleccionado todo apretaríamos el botón azul y para finalizar la creación de la alarma, indicaremos los valores de los umbrales deseados para alarmas warning, mayor y critical.

A continuación, se muestra la opción de “Métricas configuradas” en el que se listarán las métricas/alarmas configuradas en una tabla. La tabla está formada por cinco columnas: dashboard, panel, métrica y dos columnas en las que se permite la edición de los umbrales de la métrica (circulo azul con lápiz en el interior) o la supresión de la misma (círculo rojo con x).

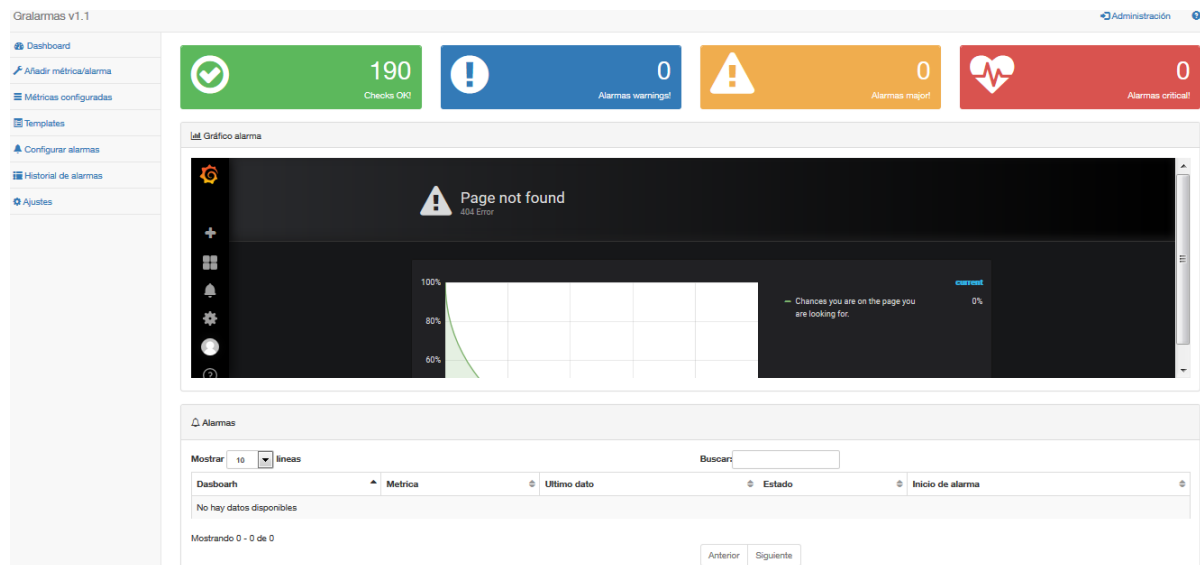


Figura 5.2: Dashboard

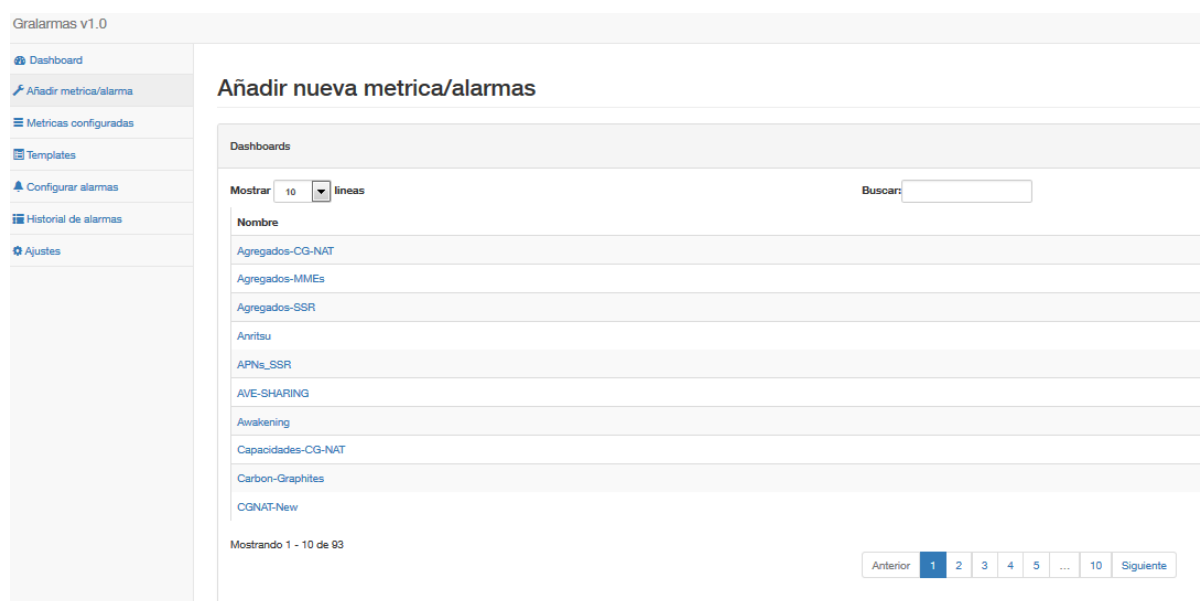


Figura 5.3: Selección Dashboard

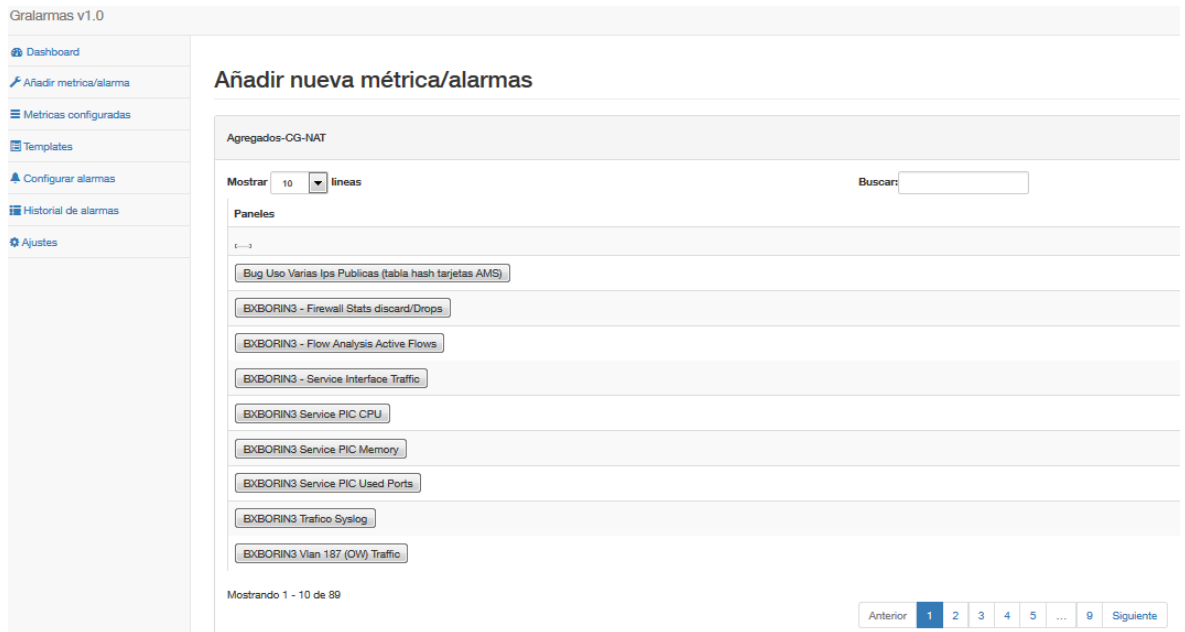


Figura 5.4: Selección Panel

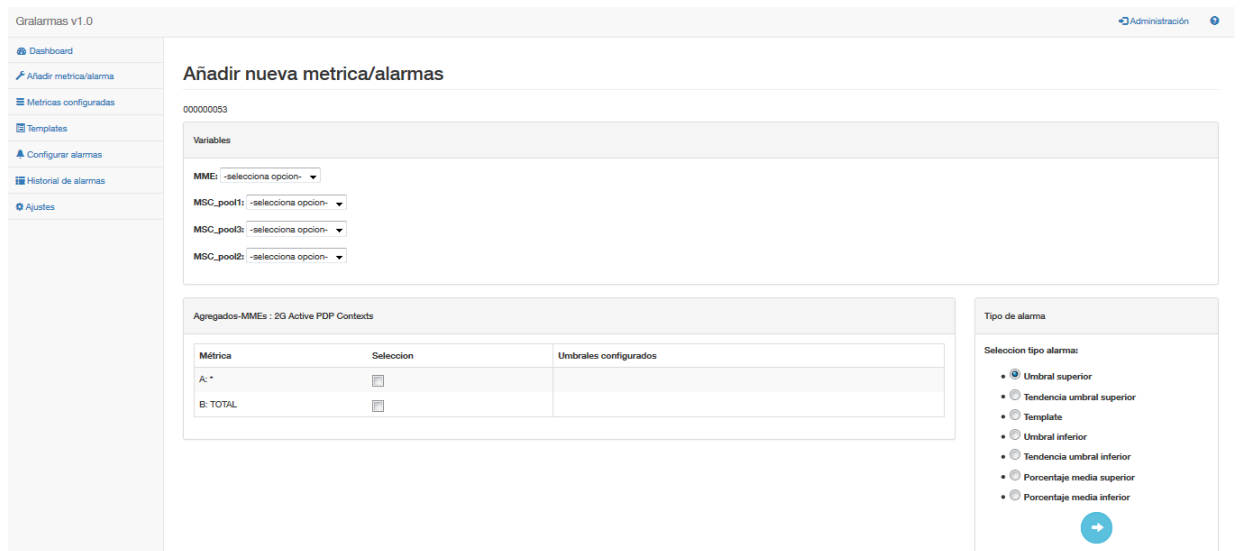


Figura 5.5: Selección Métrica

Gralarmas v1.0

- Dashboard
- Añadir metrica/alarma
- Metricas configuradas
- Templates
- Configurar alarmas
- Historial de alarmas
- Ajustes

Añadir nueva metrica/alarmas

Agregados-CG-NAT : BXBORIN3 - Firewall Stats disoard/Drops

Metricas seleccionadas:
A#aliasByNode(perSecond(junos.BXBORIN3.swf_statistics.*.total_drops), 3, 5)

Configuracion de umbrales:

Valor warning: 0

Valor mayor: 0

Valor critica: 0

* Mantener umbral "0" por defecto

➔

Figura 5.6: Combo Umbrales

Gralarmas v1.0 Administración

- Dashboard
- Añadir metrica/alarma
- Metricas configuradas
- Templates
- Configurar alarmas
- Historial de alarmas
- Ajustes

Metricas configuradas

Mostrar: 10 líneas Buscar:

Dashboard	Panel	Metrica			
Agregados-CG-NAT	Velocidad de los interfaces de gestion	A#aliasByNode(junos.BXBORIN3.Velociad_interface.ge-0_2_1, 1, 3)	✓	✗	
Agregados-CG-NAT	Velocidad de los interfaces de gestion	F#aliasByNode(junos.MXMERIN4.velociad_interface.ge-11_2_0, 1, 3)	✓	✗	
Agregados-CG-NAT	Velocidad de los interfaces de gestion	E#aliasByNode(junos.MXMERIN3.Velociad_interface.ge-0_2_1, 1, 3)	✓	✗	
Agregados-CG-NAT	Velocidad de los interfaces de gestion	H#aliasByNode(junos.SEPFRIN3.Velociad_interface.ge-0_2_1, 1, 3)	✓	✗	
Agregados-CG-NAT	Velocidad de los interfaces de gestion	G#aliasByNode(junos.MXULFRIN3.Velociad_interface.ge-0_2_1, 1, 3)	✓	✗	
Agregados-CG-NAT	Velocidad de los interfaces de gestion	C#aliasByNode(junos.EXQFRIN3.velociad_interface.ge-11_2_0, 1, 3)	✓	✗	

Figura 5.7: Listado métricas configuradas

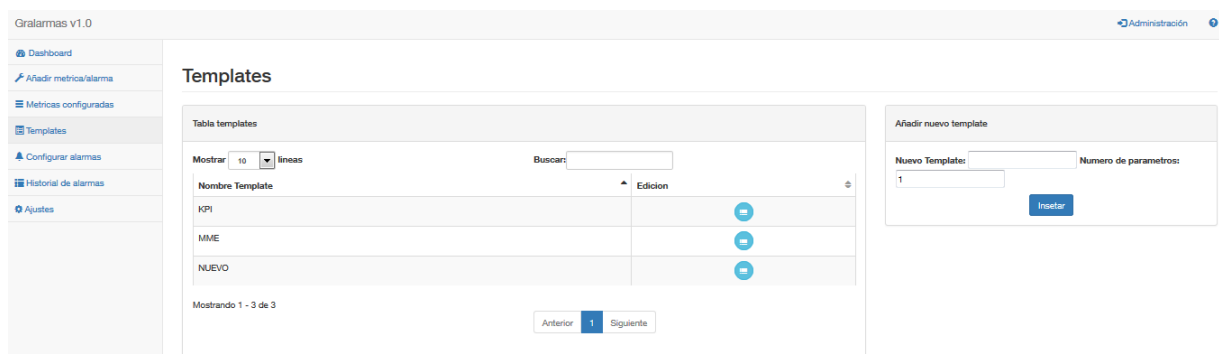


Figura 5.8: Vista principal templates

En la opción de “templates” se mostraran y configurarán los templates. En el cuadro de añadir nuevo template se añadirá el nombre y el número de parámetros necesarios. Dichos parámetros son el número de tipo de umbrales que queremos configurar. Para finalizar, seleccionaremos el botón de edición (circulo azul con líneas blancas) y configuraremos los parámetros.

Una de las vistas más importantes es “Configurar alarmas”, en ella se listan las alarmas permitiendo seleccionar las siguientes opciones para cada una de ellas:

- Envío SMS: True o False. Seleccionas si deseamos alertas a través de SMS.
- Envío OVO: True o False. Seleccionas si deseamos alertas a través de OVO (Herramienta interna).
- Checks alerta: número de veces que tiene que sobrepasar un valor el umbral para que la métrica sea alarmada.
- Checks OK: número de veces que tiene que sobrepasar un valor el umbral para que la métrica sea cesada.
- Periodo alerta: seleccionar periodo de activación. 24x7
- Periodo diurno.

Gralarmas v1.0

- Dashboard
- Añadir métrica/alarma
- Métricas configuradas
- Templates
- Configurar alarmas
- Historial de alarmas
- Ajustes

Templates

Editar templates

Nombre template:	KPI
Tipo umbral:	1
Valor warning:	40,0
Valor mayor:	60,0
Valor critico:	80,0
Nombre template:	KPI
Tipo umbral:	4
Valor warning:	20,0
Valor mayor:	15,0
Valor critico:	10,0

Guardar valores

Figura 5.9: Combo parámetros templates

- Margen OK (%): porcentaje del umbral para garantizar el OK.

En la vista de “Alarmas históricas” se muestra un listado de las alarmas históricas. Dicha vista está formada por una tabla con siete columnas, dashboard, panel, métrica, el último dato con el que fue alarmado, estado, fecha y hora de inicio de la alarma y fecha y hora de fin de la alarma.

Como última opción están los “Ajustes”, donde podremos rellenar modificar algunas configuraciones básicas:

- IP servidor grafana: IP del servidor grafana con el que nos conectaremos.
- Puerto servidor grafana: puerto de grafana escuchando para establecer la comunicación.
- API key: key para atacar a la API de Grafana.
- Botón cargar BBDDs: actualizar la conectividad con el servidor de grafana.
- Modo visualización métricas: formato de visualización de las métricas. Alias, métrica completa o wildcard.

Gralarmas v1.0

Dashboard

Añadir métrica/alarma

Métricas configuradas

Plantillas

Configurar alarmas

Historial de alarmas

Ajustes

Administración

Configurar alarmas

Tabla alarmas

Mostrar 10 líneas

Buscar

Selección	Dashboard	Panel	Métrica	Envío SMS	Envío OVO	Checks alerta	Checks OK	Periodo alerta	Margen OK (%)
<input type="checkbox"/>	CG-NAT	Huawei Eth-trunk Bandwidth Utilization	MXULFPS1.Ethtrunk3.Input	True	False	3	2	24x7	10
<input type="checkbox"/>	CG-NAT	Huawei Eth-trunk Bandwidth Utilization	MXULFPS1.Ethtrunk3.Output	True	False	3	2	24x7	10
<input type="checkbox"/>	CG-NAT	Huawei Eth-trunk Bandwidth Utilization	MXULFPS2.Ethtrunk3.Input	True	False	3	2	24x7	10
<input type="checkbox"/>	CG-NAT	Huawei Eth-trunk Bandwidth Utilization	MXULFPS2.Ethtrunk3.Output	True	False	3	2	24x7	10
<input type="checkbox"/>	CG-NAT	Huawei Eth-trunk Bandwidth Utilization	VXQULFPS1.Ethtrunk4.Input	True	False	3	2	24x7	10
<input type="checkbox"/>	CG-NAT	Huawei Eth-trunk Bandwidth Utilization	VXQULFPS2.Ethtrunk4.Output	True	False	3	2	24x7	10
<input type="checkbox"/>	CG-NAT	Huawei Eth-trunk Bandwidth Utilization	MXFULFPS1.Ethtrunk3.Input	True	False	3	2	24x7	10
<input type="checkbox"/>	CG-NAT	Huawei Eth-trunk Bandwidth Utilization	MXFULFPS1.Ethtrunk3.Output	True	False	3	2	24x7	10

Mostrando 1 - 10 de 190

Anterior

12345...19

Siguiente

Figura 5.10: Vista del configurador de alarmas

Grallamas v1.0

Dashboard

Añadir métrica/alarma

Métricas configuradas

Templantes

Configurar alarmas

Historial de alarmas

Ajustes

Alarmas históricas

Historial de alarmas

Mostrar

10

líneas

Buscar

Dashboard	Panel	Métrica	Ultimo dato (Mbps)	Estado	Inicio de alarma	Fin de alarma
Agregados-MMIES	Nbr Disconnected eNodeB MMIES	AND80L03	35.0	Mayor	13 de Enero de 2019 a las 00:15	13 de Enero de 2019 a las 04:30
Agregados-MMIES	Nbr Disconnected eNodeB MMIES	CAT07L05	35.0	Mayor	13 de Enero de 2019 a las 00:15	13 de Enero de 2019 a las 04:30
Agregados-MMIES	Nbr Disconnected eNodeB MMIES	CAT80L03	35.0	Mayor	13 de Enero de 2019 a las 00:15	13 de Enero de 2019 a las 04:30
Agregados-MMIES	Nbr Disconnected eNodeB MMIES	MAD08L04	33.0	Mayor	13 de Enero de 2019 a las 00:15	13 de Enero de 2019 a las 04:30
Agregados-MMIES	Nbr Disconnected eNodeB MMIES	MAD80L01	35.0	Mayor	13 de Enero de 2019 a las 00:15	13 de Enero de 2019 a las 04:30
Agregados-MMIES	Nbr Disconnected eNodeB MMIES	MAD80L01	33.0	Mayor	13 de Enero de 2019 a las 00:15	13 de Enero de 2019 a las 04:30
Agregados-MMIES	Nbr Disconnected eNodeB MMIES	VAL80L03	35.0	Mayor	13 de Enero de 2019 a las 00:15	13 de Enero de 2019 a las 04:30
Agregados-MMIES	Nbr Disconnected eNodeB MMIES	AND80L03	38.0	Mayor	13 de Enero de 2019 a las 23:30	13 de Enero de 2019 a las 23:45
Agregados-MMIES	Nbr Disconnected eNodeB MMIES	CAT07L05	38.0	Mayor	13 de Enero de 2019 a las 23:30	13 de Enero de 2019 a las 23:45
Agregados-MMIES	Nbr Disconnected eNodeB MMIES	CAT80L03	35.0	Mayor	13 de Enero de 2019 a las 23:30	13 de Enero de 2019 a las 23:45

Mostrando 1 - 10 de 4,048

Anterior

1

2

3

4

5

...

405

Siguiente

Figura 5.11: Vista de histórico alarmas

```
21
22 +class FormularioUmbralTendenciaTemplate(forms.Form):...
28
29 +class FormularioUmbrals(forms.Form):...
33
34 +class FormularioError(forms.Form):...
36
37 +class FormularioTendencia(forms.Form):...
41
42 +class FormularioNuevoTemplate(forms.Form):...
44
45 +class FormularioTemplate(forms.Form):...
```

Figura 5.12: forms.py

- Visualización rows: True o False. Mostrar la row de Grafana a la que pertenece un panel.
- Periodo diurno: horas en las que se alertara si se selecciona la opción “periodo diurno” en la configuración de la alarma.
- Método de envió: seleccionar si se va a indicar una lista telefónica o números.
- Teléfonos o lista: rellenar teléfono o listado.

Cada uno de los formularios mostrados en las vistas previas forma parte de una clase en el *forms.py*.

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

Aunque continuamos con la puesta en marcha de la solución y con desarrollo y evolución continua, los objetivos se han cumplido con creces y el resultado es bastante satisfactorio. Se ha conseguido una monitorización de la capa de PS novedosa y revolucionaria. El TFG se ha llevado a cabo en un tiempo límite de 3 meses, en el cual se han ido realizando entregas de valor al responsable a la finalización de cada objetivo.

En el proyecto abordado se ha realizado las siguientes tareas:

- Traslado e instalación hardware en CPD.
- Configuración hardware, instalación de sistema operativo e instalación de paquetes necesarios de los tres servidores.
- Instalación y configuración óptima de Grafana.
- Instalación y configuración óptima de Graphite.
- Revisión y apertura de conectividades entre servidores y con equipos remotos.
- Desarrollo de scripting en Python y documentación de los mismos.

- Desarrollo web en Django.
- Configuración de Backup.

6.2. Aplicación de lo aprendido

En esta sección se enumeran las asignaturas relacionadas y aplicadas en el proyecto:

1. Arquitectura de Internet: En ella se adquirieron las nociones básicas en arquitecturas de protocolos en redes de ordenadores, centrándose en los protocolos TCP/IP, empleada en la realización del diseño y arquitectura de la solución.
2. Informática I / II: Aunque en estas asignaturas no se impartía ninguno de los lenguajes utilizados en el proyecto, sí se obtuvieron los conocimientos básicos de programación: tipos de datos, variables, control de flujo, ciclos, estructura de datos, funciones,...
3. Protocolos para la Transmisión de Audio y Vídeo en Internet: A lo largo de la asignatura se aprendieron los principales elementos del proyecto, el lenguaje de programación Python y el framework Django.
4. Sistemas Telemáticos para Medios Audiovisuales: Esta asignatura completó la formación básica en el campo de las redes de ordenadores, las competencias y conocimientos adquiridos en Arquitectura de Internet, los reforzó y completó. En sus contenidos se incluye el estudio de los dispositivos de interconexión de redes, los protocolos de encaminamiento y la seguridad en redes de ordenadores empleados en el proyecto.

6.3. Lecciones aprendidas

En el presente proyecto se han adquirido las siguientes competencias profesionales:

1. Mejora de conocimientos de Python.
2. Mejora de conocimientos de Django.
3. Mejora de conocimientos de Linux.
4. Conocimiento sobre herramientas Open Source (Grafana y Graphite).
5. Manejo de Bootstrap.
6. Elaboración de un documento técnico.

6.4. Trabajos futuros

Ningún proyecto software termina, como se hizo mención en la sección 2.3 se esta aplicando metodología DevOps. Con una continua evolución del software según necesidades de cliente.

Algunos puntos de mejora identificados son:

- Redundancia local, redundancia dentro del mismo CPD pero en un rack distinto.
- Redundancia geográfica, redundancia en otro CPD.
- Análisis de logs, añadir la funcionalidad de analizar logs ademas de series de tiempo.
- Dockerizar la solución, hacer la solución mas potente y flexible añadiendo dockers.
- Agregación de alarmas y machine learning, añadir lógicas mas potentes.

Bibliografía

<https://grafana.com/>

<https://graphiteapp.org/>

<https://www.dell.com/es-es>

<https://www.djangoproject.com/>

<https://www.w3schools.com/> document