Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Sistema de Monitorización de Servidores y Servicios en Infraestructuras de Investigación

Autor: Luis Garcia Capilla
Tutor: Daniel Garijo Verdejo
Cotutor: Esteban Gonzalez Guardia

Madrid, enero 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Sistema de Monitorización de Servidores y Servicios en Infraestructuras de Investigación
Enero 2025.

Autor: Luis Garcia Capilla

Tutor:
Daniel Garijo Verdejo
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Este Trabajo de Fin de Grado presenta el diseño e implementación de un sistema de monitorización y visualización para la infraestructura de servidores y servicios del grupo de investigación OEG (Ontology Engineering Group) de la Universidad Politécnica de Madrid. El objetivo principal ha sido desarrollar una solución que permita la supervisión en tiempo real del rendimiento de los sistemas, la detección temprana de problemas y el envío de alertas automáticas cuando se alcanzan valores críticos en las métricas clave, facilitando así una gestión proactiva y eficiente de los recursos.

El sistema propuesto utiliza un stack tecnológico compuesto por Telegraf, InfluxDB y Grafana, herramientas ampliamente utilizadas en entornos de monitorización debido a su flexibilidad y capacidad de personalización. Telegraf se configuró como el agente de recolección de métricas, capturando información detallada sobre el uso de CPU, la memoria, el almacenamiento, la disponibilidad de servicios web y estado de los servicios Docker mediante peticiones HTTPEn nuestro proyecto manejamos series de datos temporales, y estas métricas se almacenan en InfluxDB, una base de datos de series temporales especialmente diseñada para manejar grandes volúmenes de datos que se actualizan constantemente. Por último, Grafana se empleó como herramienta de visualización, permitiendo el diseño de paneles interactivos y dinámicos que muestran las métricas en tiempo real de manera intuitiva.

Uno de los aspectos destacados del sistema es el diseño y configuración de dashboards personalizados en Grafana, que permiten visualizar las métricas agrupadas por tipo de servicio o servidor. Se crearon dashboards específicos para monitorizar servidores individuales, servicios web y contenedores Docker, cada uno adaptado a las necesidades específicas del grupo de investigación. Estos paneles incluyen gráficas detalladas que facilitan la interpretación de los datos y permiten identificar rápidamente cualquier anomalía en el sistema.

El sistema de alertas configurado en Grafana constituye otra pieza clave del proyecto. Este sistema permite establecer umbrales para cada métrica relevante y notifica automáticamente por correo electrónico cuando se alcanzan estos valores, gracias a la configuración de un servidor SMTP. Esta funcionalidad asegura que el equipo de administración pueda reaccionar de manera oportuna ante situaciones críticas, minimizando así el impacto de posibles fallos en el servicio.

Para garantizar la replicabilidad del sistema, se elaboró una guía completa de instalación y configuración que abarca desde la descarga e instalación de las herramientas hasta la creación y personalización de dashboards y alertas. Esta documentación no solo facilita la futura implementación del sistema en otros entornos, sino que también proporciona las instrucciones necesarias para ampliar el sistema con nuevas métricas o servidores.

En resumen, este trabajo ha desarrollado una infraestructura de monitorización robusta, flexible y escalable que proporciona al grupo de investigación OEG una herramienta efectiva para supervisar el estado de sus sistemas y anticiparse a posibles problemas. La capacidad de personalización de los paneles y la rapidez en la detección de incidencias mediante alertas automáticas son características clave que aportan un valor significativo a la gestión de los recursos tecnológicos del grupo.

Abstract

This Bachelor's Thesis focuses on the design and implementation of a monitoring and visualization system for the infrastructure of servers and services at the Ontology Engineering Group (OEG) of the Polytechnic University of Madrid. The main objective was to develop a solution capable of real-time performance monitoring, early problem detection, and automatic alert generation when critical thresholds are reached, thus enabling proactive and efficient resource management.

The proposed system leverages a technological stack composed of Telegraf, InfluxDB, and Grafana, which are widely used tools in monitoring environments due to their flexibility and high customization potential. Telegraf was configured as the data collection agent, gathering detailed information on CPU usage, memory consumption, storage utilization, Docker status and service availability through HTTP requestsIn our project, we work with time-series data, and the collected metrics are stored in InfluxDB, a time-series database specifically designed to handle large volumes of frequently updated data. Finally, Grafana was employed as the visualization platform, enabling the creation of interactive and dynamic dashboards that display real-time metrics in an intuitive manner.

One of the key components of the system is the set of customized dashboards created in Grafana, which provide clear and organized visualization of metrics by service type or server. Specific dashboards were designed for monitoring individual servers, web services, and Docker containers, each tailored to the specific needs of the research group. These dashboards feature detailed graphs that facilitate data interpretation and allow for quick identification of any anomalies in the system.

Another significant element of the project is the alert system implemented in Grafana. This system allows the definition of thresholds for each relevant metric and sends automatic email notifications when these values are exceeded, using an SMTP server configuration. This functionality ensures that the administration team can respond promptly to critical situations, minimizing the potential impact of system failures.

To ensure the replicability of the system, a comprehensive installation and configuration guide was developed. This guide covers everything from downloading and installing the tools to creating and customizing dashboards and alerts. The documentation not only facilitates future deployment of the system in different environments but also provides the necessary instructions for expanding the system with new metrics or servers.

In conclusion, this project has developed a robust, flexible, and scalable monitoring infrastructure that offers the OEG research group an effective tool for supervising the status of their systems and anticipating potential problems. The high level of customization of the dashboards and the rapid

incident detection through automatic alerts are key features that significantly enhance the management of the group's technological resources.

Tabla de contenidos

1 Introducción

1.1 Contexto y motivación del proyecto

En un entorno cada vez más dependiente de infraestructuras digitales y servicios online, la monitorización eficiente de servidores y servicios se ha convertido en una necesidad fundamental. Los grupos de investigación, como el Ontology Engineering Group (OEG) de la Universidad Politécnica de Madrid, gestionan un volumen considerable de recursos informáticos que requieren una supervisión constante para garantizar su correcto funcionamiento. La detección temprana de problemas y la capacidad de actuar rápidamente ante incidencias son aspectos clave para asegurar la continuidad de los servicios y el rendimiento óptimo de las aplicaciones.

La motivación principal de este proyecto surge de la necesidad de implementar un sistema de monitorización integral que no solo permita visualizar en tiempo real el estado de los recursos, sino también generar alertas automáticas cuando se detecten anomalías. Para ello, se ha optado por utilizar herramientas de código abierto ampliamente utilizadas en la industria, como **Telegraf**, **InfluxDB** y **Grafana**, que ofrecen flexibilidad y escalabilidad para entornos de monitorización avanzados. La elección de estas herramientas se fundamenta en su capacidad para manejar grandes volúmenes de datos en tiempo real, su facilidad de integración y su versatilidad a la hora de crear dashboards personalizados y configurar alertas.

Este proyecto no solo representa una solución técnica a una necesidad concreta, sino también una oportunidad de aprendizaje práctico en la gestión de sistemas de monitorización, el manejo de bases de datos de series temporales y el diseño de paneles de visualización interactivos. La posibilidad de documentar y compartir esta experiencia contribuirá a que el sistema desarrollado pueda ser replicado y mejorado en otros entornos, tanto académicos como profesionales.

1.2 Objetivos generales y específicos

- **Objetivos generales**

El objetivo principal de este Trabajo de Fin de Grado es desarrollar un sistema de monitorización que permita al grupo de investigación OEG supervisar y analizar el estado de sus servidores y servicios en tiempo real. Este sistema debe ser capaz de recopilar métricas clave, visualizarlas de manera intuitiva y generar alertas automáticas que faciliten la detección de fallos o situaciones críticas.

- **Objetivos específicos**
- **Implementar un sistema de monitorización funcional**: Configurar Telegraf para la recolección de métricas y asegurar su almacenamiento correcto en InfluxDB.
- **Diseñar dashboards personalizados**: Crear paneles en Grafana que permitan visualizar de forma clara e intuitiva las métricas recogidas.
- **Configurar un sistema de alertas automáticas**: Establecer umbrales críticos para diferentes métricas y configurar notificaciones por correo electrónico a través de SMTP.

- **Documentar el proceso de instalación y configuración**: Elaborar una guía detallada que permita replicar el sistema en otros entornos y facilitar su mantenimiento y ampliación futura.

Modificaciones realizadas durante el desarrollo

Durante el desarrollo del proyecto, se realizaron numerosas modificaciones respecto a los objetivos iniciales para adaptarse a los retos y necesidades que surgieron:

1. **Optimización del sistema de monitorización de servicios web y contenedores Docker**: Inicialmente, se elaboró un script en Python que requería crear un input en Telegraf y un script por cada URL a monitorizar. Esta solución resultó adecuada para entornos pequeños, pero ineficiente para infraestructuras más grandes como la del OEG, donde se gestionan una gran cantidad de servicios web y contenedores. La solución fue desarrollar un script genérico que lee las URLs desde un archivo JSON, eliminando la necesidad de definir inputs individuales por servicio.

2. **Incorporación de filtros personalizados**: Con el fin de mejorar la capacidad de análisis y segmentación de los datos, se agregaron campos adicionales en el archivo JSON que permiten al usuario definir información como el nombre de la máquina virtual, el administrador responsable y palabras clave. Esta información se envía como tags en las métricas, permitiendo filtrar y agrupar los datos de manera eficiente en los dashboards de Grafana.

3. **Eliminación de la dependencia de Python para monitorizar contenedores Docker**: Una de las principales limitaciones detectadas fue que el script inicial para monitorizar Docker requería que Python estuviera instalado en cada servidor, lo cual no siempre es viable en entornos de producción. Para resolver este problema, se optó por utilizar el plugin nativo de Docker en Telegraf. Este plugin permite recolectar métricas técnicas (CPU, memoria, disco y red) de los contenedores, presentándolas como series temporales. De esta forma, se puede visualizar el momento exacto en que un contenedor deja de enviar datos, lo que facilita la detección de contenedores inactivos.

4. **Simplificación de la configuración de los dashboards**: Al principio, el usuario tenía que modificar manualmente los nombres de los buckets y los hosts en los archivos JSON de los dashboards, lo que resultaba poco práctico. Se resolvió este inconveniente implementando variables en Grafana, lo que permite seleccionar los buckets y hosts disponibles de manera dinámica a través de un desplegable en la interfaz.

5. **Uso del lenguaje Flux para las consultas**: Dado que InfluxDB utiliza el lenguaje de consultas Flux, fue necesario adaptar todas las queries a este formato. Flux es un lenguaje potente pero complejo, lo que implicó un esfuerzo adicional para crear consultas optimizadas que permitieran extraer la información necesaria de manera eficiente.

Estos cambios no solo mejoraron la usabilidad y escalabilidad del sistema, sino que también permitieron ofrecer una solución más robusta y adaptada a las necesidades reales del entorno.

1.3 Planificación del Proyecto

En esta sección se incluye un análisis de la cronología de las tareas que se han llevado a cabo a lo largo del proyecto. Se presenta un diagrama de Gantt que muestra las fechas estimadas de cada tarea y se detallan las distintas fases del proyecto:

1. Estudio de herramientas

i. Estudio de la herramienta Telegraf para la recolección de métricas.
ii. Familiarización con InfluxDB como base de datos de series temporales.
iii. Análisis de Grafana como herramienta de visualización de dashboards.

2. Identificación de métricas de desarrollo y dashboards

i. Definición de las métricas clave para monitorizar el rendimiento de los servidores y servicios.
ii. Selección de métricas para la supervisión de contenedores Docker.
iii. Diseño preliminar de los dashboards en Grafana.

3. Diseño de la arquitectura del sistema

i. Diseño de la arquitectura general del sistema de monitorización
ii. Planificación del flujo de datos desde Telegraf hasta Grafana
iii. Documentación del diseño arquitectónico.

4. Implementación del sistema

i. Configuración de Telegraf para la recolección de métricas del sistema.
ii. Configuración de InfluxDB para el almacenamiento eficiente de las métricas.
iii. Creación de dashboards personalizados en Grafana.
iv. Configuración del sistema de alertas en Grafana.

5. Despliegue del sistema en entorno de pruebas

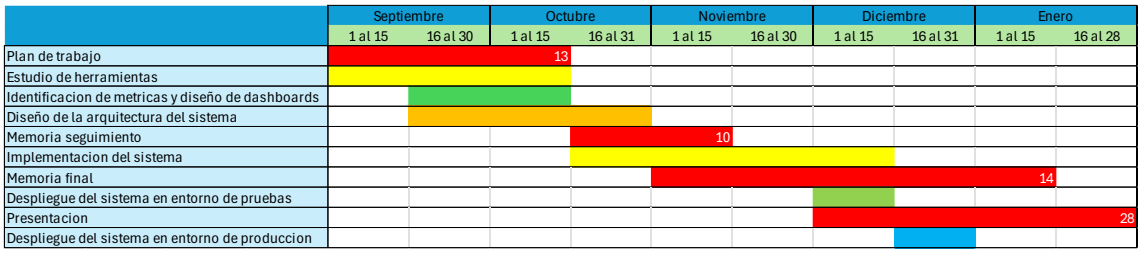
i. Instalación del sistema en un entorno controlado de pruebas.
ii. Verificación del correcto funcionamiento de la recolección de métricas.
iii. Pruebas de integridad de los dashboards y del sistema de alertas.

6. Despliegue del sistema en entorno de producción (versión inicial OEG)

i. Despliegue de la versión inicial del sistema en el entorno de producción del grupo OEG.

7. Resultados y conclusiones

i. Análisis de los resultados obtenidos durante el despliegue en producción.
ii. Redacción de la memoria final del proyecto.
iii. Elaboración de las conclusiones y propuestas de trabajo futuro.



1.4 Estructura del documento

Este documento se organiza en cuatro capítulos principales, además de la bibliografía y los anexos:

- **Capítulo 1: Introducción**. Se presenta el contexto del proyecto, la motivación que impulsó su desarrollo, los objetivos generales y específicos, así como una descripción de la estructura de la memoria.
- **Capítulo 2: Desarrollo**. Se detalla el proceso de análisis de herramientas, el diseño de la arquitectura del sistema, la instalación y configuración de las distintas herramientas utilizadas (Telegraf, InfluxDB y Grafana), la creación de dashboards personalizados y el sistema de alertas implementado. Además, se incluye una explicación del código utilizado y de los principales desafíos enfrentados durante el desarrollo.
- **Capítulo 3: Resultados y Conclusiones**. Se exponen los resultados obtenidos con el sistema desarrollado, su impacto en la gestión de la infraestructura del grupo OEG, las conclusiones generales del proyecto y posibles líneas de trabajo futuro.
- **Capítulo 4: Análisis de Impacto**. Se realiza un análisis del impacto técnico, económico y social del sistema de monitorización implementado.

Finalmente, se incluyen la **Bibliografía** utilizada durante el desarrollo del proyecto y un anexo con los códigos mencionados en la memoria.

2 Desarrollo

2.1 Análisis de herramientas seleccionadas

En el desarrollo de este Trabajo de Fin de Grado, se ha optado por un stack tecnológico compuesto por tres herramientas principales: **Telegraf**, **InfluxDB** y **Grafana**. Estas herramientas han sido seleccionadas por su amplia adopción en entornos profesionales de monitorización y por sus capacidades avanzadas para gestionar grandes volúmenes de datos, visualizarlos de manera eficiente y generar alertas automáticas en tiempo real.

2.1.1 Analisis de Telegraf, InfluxDB, Grafana:

Telegraf:

Telegraf es un agente de recolección de métricas desarrollado por InfluxData. Su principal función es recopilar datos de diversas fuentes, como sistemas operativos, aplicaciones, bases de datos y servicios web, y enviarlos a una base de datos de series temporales. Una de las ventajas clave de Telegraf es su arquitectura modular basada en plugins, que permite extender sus funcionalidades para adaptarse a diferentes necesidades de monitorización.

En este proyecto, Telegraf se ha utilizado para recolectar métricas de:

- **Recursos del sistema**: CPU, memoria, almacenamiento y red de los servidores.
- **Servicios web**: Estado y latencia de los servicios expuestos.
- **Contenedores Docker**: Métricas técnicas de los contenedores en ejecución.

La facilidad de configuración y la amplia variedad de plugins disponibles han sido factores determinantes para su elección. Además, su capacidad para enviar métricas a InfluxDB de manera eficiente ha permitido construir un sistema de monitorización escalable.

InfluxDB

InfluxDB es una base de datos de series temporales también desarrollada por InfluxData, diseñada para almacenar y consultar grandes volúmenes de datos que varían en el tiempo. Su motor de almacenamiento está optimizado para manejar millones de puntos de datos por segundo, lo que la convierte en una opción ideal para entornos de monitorización en tiempo real.

En este proyecto, InfluxDB se ha utilizado para:

- **Almacenar las métricas recolectadas por Telegraf**.
- **Gestionar grandes volúmenes de datos de manera eficiente**.
- **Permitir consultas avanzadas mediante el lenguaje Flux**, el cual ofrece gran flexibilidad para extraer información relevante y generar dashboards personalizados.

La elección de InfluxDB se ha basado en su rendimiento superior frente a otras bases de datos tradicionales y su capacidad para integrarse de manera nativa con Telegraf y Grafana, lo que ha simplificado significativamente el desarrollo del sistema

Grafana
Grafana es una plataforma de visualización de datos que permite crear dashboards interactivos y personalizables. Su interfaz amigable y su capacidad para integrar datos de diversas fuentes la han convertido en una herramienta muy popular en el ámbito de la monitorización.

En este proyecto, Grafana se ha utilizado para:

- **Visualizar en tiempo real las métricas almacenadas en InfluxDB** mediante dashboards personalizados.
- **Configurar alertas automáticas** que notifican al equipo cuando se superan ciertos umbrales críticos.
- **Facilitar el análisis y la toma de decisiones** gracias a su capacidad para filtrar y segmentar los datos según diferentes criterios, como el tipo de servicio o el servidor monitorizado.

La posibilidad de definir variables dentro de los dashboards y de crear paneles altamente interactivos ha sido clave para ofrecer una experiencia de usuario intuitiva y eficiente. Además, la integración nativa de Grafana con InfluxDB ha permitido optimizar el flujo de trabajo y reducir los tiempos de desarrollo.

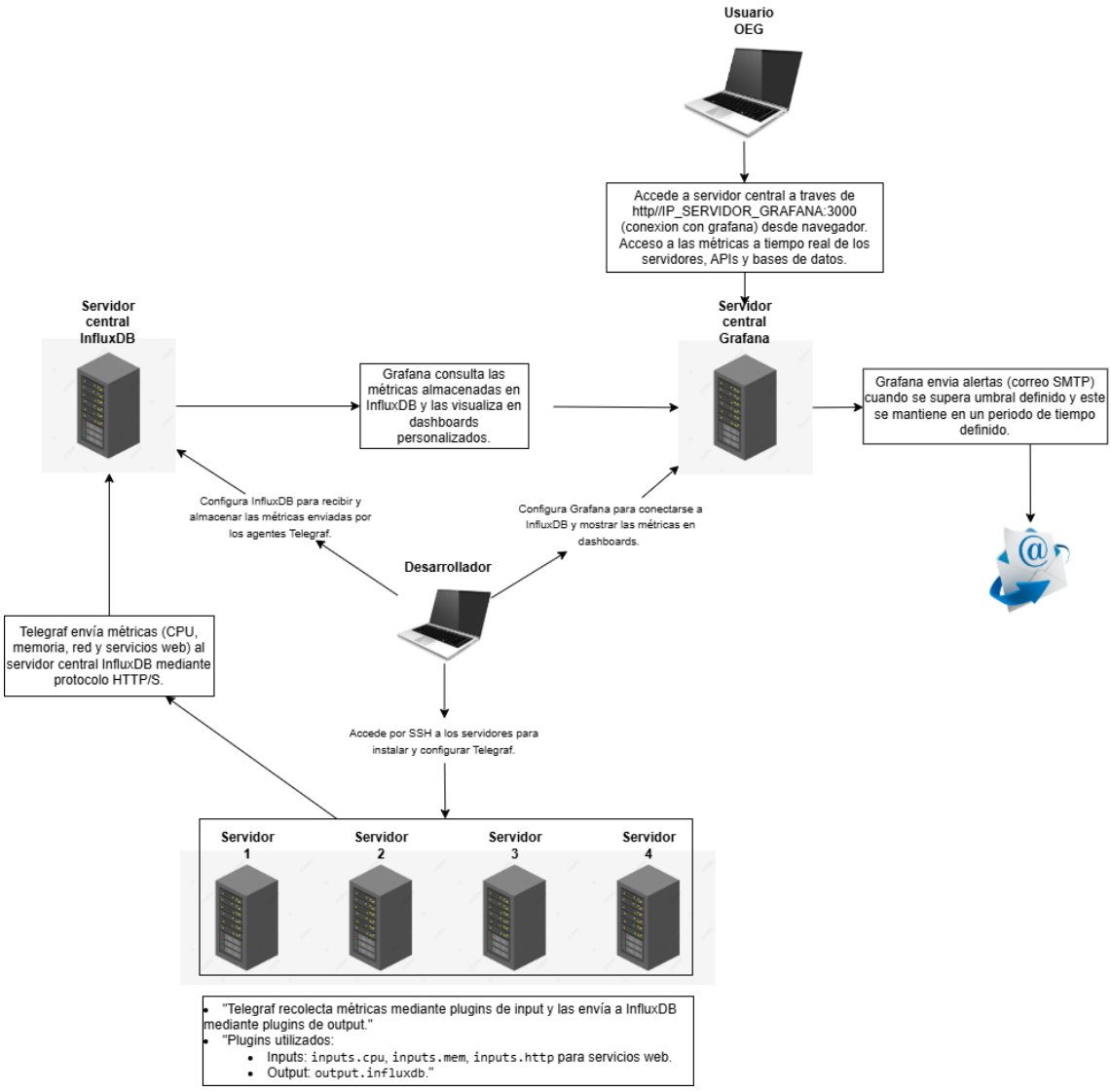
2.1.2 Justificación de la elección del stack
La combinación de **Telegraf**, **InfluxDB** y **Grafana** ha demostrado ser una solución robusta, flexible y escalable para abordar los requisitos del proyecto. Cada herramienta desempeña un rol específico y complementario:

1. **Telegraf** se encarga de la recolección de datos desde diversas fuentes, garantizando que se cubran todas las necesidades de monitorización.
2. **InfluxDB** proporciona un almacenamiento eficiente y optimizado para datos de series temporales, permitiendo consultas avanzadas y rápidas.
3. **Grafana** facilita la visualización y el análisis de los datos, además de ofrecer funcionalidades avanzadas de alertas y personalización de dashboards.

Este stack tecnológico ha permitido desarrollar un sistema de monitorización que cumple con los objetivos planteados al inicio del proyecto: una herramienta que no solo permite visualizar el estado de los recursos en tiempo real, sino que también facilita la detección temprana de problemas y la toma de decisiones informada.

2.2 Arquitectura del Sistema
Diagrama de arquitectura

El siguiente diagrama ilustra la arquitectura del sistema, mostrando la interacción entre los servidores monitorizados, los servidores centrales y los usuarios que acceden al sistema de monitorización.



2.2.1 Descripción del flujo de trabajo entre las herramientas

La arquitectura del sistema de monitorización está diseñada para centralizar la recolección y visualización de métricas clave en dos servidores centrales independientes: uno dedicado a **InfluxDB** y otro a **Grafana**. Esta configuración permite distribuir la carga de trabajo y mejorar la eficiencia en el procesamiento y visualización de datos.

El flujo de trabajo general del sistema se describe a continuación:

1. **Recolección de métricas mediante Telegraf**: Cada servidor de la infraestructura cuenta con un agente **Telegraf** instalado. Este agente recolecta métricas clave, como el uso de CPU, memoria, almacenamiento y el estado de los contenedores Docker y servicios web. Telegraf utiliza plugins de input y output para obtener y enviar estas métricas.

2. **Envío de métricas a InfluxDB**: Los agentes Telegraf envían las métricas recolectadas al servidor central que aloja **InfluxDB** utilizando protocolos HTTP/S. InfluxDB actúa como la base de datos de series temporales, almacenando los datos de manera estructurada para su posterior consulta.
3. **Visualización de métricas en Grafana**: El servidor que aloja **Grafana** se conecta a InfluxDB para extraer y visualizar las métricas en tiempo real mediante dashboards personalizados. Grafana permite filtrar y segmentar las métricas por diferentes criterios, como el nombre del servidor, el tipo de servicio o palabras clave definidas por el usuario.
4. **Alertas automáticas**: Grafana está configurado para generar alertas automáticas cuando las métricas superan ciertos umbrales críticos. Estas alertas se envían por correo electrónico mediante un servidor SMTP configurado en Grafana, lo que permite una respuesta rápida ante incidencias.

2.2.2 Estructura General de la Arquitectura

1. **Servidor Central con InfluxDB**:
   - Encargado de almacenar las métricas recibidas de los agentes Telegraf.
   - Proporciona un almacenamiento eficiente de grandes volúmenes de datos en series temporales.
2. **Servidor Central con Grafana**:
   - Dedicado a la visualización de las métricas almacenadas en InfluxDB.
   - Permite la creación de dashboards personalizados y la configuración de alertas automáticas.
3. **Servidores Monitorizados (Agentes Telegraf)**:
   - Cada servidor monitorizado tiene un agente Telegraf instalado que recolecta métricas del sistema operativo y servicios web.
   - Los agentes envían las métricas al servidor central de InfluxDB utilizando el protocolo HTTP/S.
4. **Usuarios del Sistema**:
   - Los miembros del grupo OEG acceden a los dashboards de Grafana desde cualquier navegador web mediante la URL del servidor central de Grafana.
   - Pueden visualizar el estado de los servidores y servicios en tiempo real, así como recibir alertas automáticas por correo electrónico.

2.2.3 Flujo de Trabajo de la Arquitectura

Instalación y Configuración de Agentes Telegraf

- Se accede a cada servidor mediante SSH para instalar el agente Telegraf.
- Telegraf se configura para recolectar métricas específicas como uso de CPU, memoria, disco y disponibilidad de servicios.
- Los datos recolectados se envían al servidor central en intervalos regulares, lo que permite un monitoreo continuo del estado de cada servidor.

Configuración del Servidor Central con InfluxDB y Grafana

- En el servidor central se instala InfluxDB, que recibe y almacena los datos enviados por cada agente Telegraf.
- Grafana se configura para conectarse a InfluxDB y mostrar las métricas en tiempo real a través de dashboards personalizados.
- Los dashboards permiten filtrar métricas por servidor y servicio, facilitando el análisis detallado de cada componente de la infraestructura.

Configuración de Alertas y Notificaciones

- Grafana se configura para enviar alertas cuando ciertos valores umbral son superados, como el alto uso de CPU o la falta de espacio en disco.
- Las alertas se envían mediante correo electrónico utilizando la configuración SMTP de Grafana.

2.3 Instalación y Configuración

Esta sección describe de manera general el proceso necesario para instalar y configurar el sistema de monitorización desarrollado, compuesto por las herramientas **InfluxDB**, **Telegraf** y **Grafana**. El objetivo principal de esta guía es proporcionar una referencia que permita a cualquier usuario reproducir el sistema en su propio entorno, ya sea en infraestructuras del grupo de investigación **OEG** o en entornos similares.
La guía de instalación y configuración detallada está disponible en un **repositorio de GitHub**, donde se presenta de manera estructurada y paso a paso, asegurando una implementación sencilla y guiada. Esta documentación cubre los siguientes aspectos:

1. **Instalación de herramientas básicas**:
    - Se explican los pasos necesarios para instalar **InfluxDB**, **Telegraf** y **Grafana**, herramientas clave para la recolección, almacenamiento y visualización de métricas en tiempo real.
2. **Configuración de cada herramienta**:
    - Se detallan los ajustes clave de cada herramienta, como la configuración de **Telegraf** para el envío de métricas a **InfluxDB** y la conexión de **Grafana** a la base de datos para visualizar dichas métricas.
3. **Creación de alertas automáticas**:
    - La guía incluye instrucciones para configurar alertas automáticas en **Grafana**, las cuales notifican al usuario cuando se detectan umbrales críticos en las métricas monitorizadas.
4. **Personalización de dashboards**:
    - Se explican los pasos para crear y personalizar dashboards en **Grafana**, permitiendo una visualización intuitiva y en tiempo real de las métricas relevantes.

Esta guía es fundamental para asegurar que cualquier equipo técnico pueda desplegar correctamente el sistema de monitorización y ajustarlo según sus necesidades específicas. Para consultar los detalles completos de instalación y configuración, se puede acceder al repositorio de GitHub mediante el siguiente enlace:
https://github.com/luisGarciiaa/monitorizacion-grafana-influx-telegraf.git

2.3.1 Añadir un Nuevo Servidor o Servicio al Sistema de Monitorizacion

2.3.1.1 Añadir un Nuevo Servidor

Añadir un nuevo servidor al sistema de monitorización ahora es un proceso sencillo. Solo necesitas configurar **Telegraf** en el servidor y asegurarte de que envíe las métricas al bucket de **InfluxDB**. Los dashboards de Grafana ya están

preparados para reconocer y mostrar los datos de cualquier servidor configurado en la base de datos.

Pasos:

1. Instala **Telegraf** en el nuevo servidor siguiendo la guía de instalación.

2. Configura telegraf.conf con las métricas básicas (CPU, memoria, disco).

3. Verifica que los datos se están enviando correctamente al bucket de **InfluxDB**.

2.3.1.2 Añadir un Nuevo Servicio o Contenedor Docker

Con el sistema actualizado, ya no es necesario crear scripts individuales o modificar configuraciones complejas. Ahora, simplemente edita los archivos JSON en la carpeta scripts para añadir nuevos servicios o contenedores Docker.

1.Monitorización de Servicios Web (URLs)

1. **Editar el archivo servicios.json**:
    - Abre el archivo en la carpeta scripts.
    - Añade una nueva entrada con los siguientes campos: json { "url": "http://example.com", "nombre": "mi_servicio", "keywords": "frontend,api,web" }
2. **Campos requeridos**:
    - **url**: La dirección del servicio web a monitorizar.
    - **nombre**: Un identificador único y descriptivo para el servicio.
    - **keywords**: Palabras clave asociadas para facilitar los filtros en Grafana.
3. **Guardar los cambios**:
    - Guarda el archivo. Telegraf procesará automáticamente el nuevo servicio.

2.Monitorización de Contenedores Docker

1. **Editar el archivo dockers.json**:
    - Abre el archivo en la carpeta scripts.
    - Añade una nueva entrada con los siguientes campos: json { "nombre": "mi_contenedor", "alias": "mi_contenedor_alias", "keywords": "backend,database,docker" }
2. **Campos requeridos**:
    - **nombre**: El nombre del contenedor Docker que deseas monitorizar.
    - **alias**: Un identificador único y descriptivo para el contenedor.
    - **keywords**: Palabras clave asociadas para facilitar los filtros en Grafana.

3. **Guardar los cambios**:
    - Guarda el archivo. Telegraf procesará automáticamente el nuevo contenedor.

Beneficios del Nuevo Enfoque

- **Simplificación**: Ya no es necesario crear scripts personalizados para cada servicio o contenedor.
- **Flexibilidad**: Los servicios y contenedores se configuran en un solo lugar (los archivos JSON).
- **Automatización**: Telegraf detectará y procesará automáticamente las nuevas entradas sin necesidad de reiniciar.

2.4 Alertas en Grafana

2.4.1 Explicación de las Alertas

Las alertas en Grafana son un componente clave del sistema de monitorización desarrollado, ya que permiten detectar y notificar automáticamente cualquier anomalía o incidencia en la infraestructura monitorizada. Gracias a su capacidad de supervisar múltiples métricas en tiempo real, las alertas ofrecen una solución proactiva que ayuda a mantener la estabilidad y el rendimiento del sistema.

1. **Finalidad de las Alertas**

El propósito de las alertas es doble:

a) **Prevenir problemas mayores**: Las alertas permiten identificar posibles fallos antes de que afecten gravemente al sistema o a los usuarios.
b) **Optimizar la gestión de recursos**: Al notificar automáticamente cualquier incidencia, el equipo técnico puede tomar decisiones informadas para mejorar el uso de los recursos disponibles.

Estas alertas se configuran estableciendo umbrales críticos para diferentes tipos de métricas, como el uso de CPU, memoria y espacio en disco. Cuando se superan estos umbrales, se activa una notificación que llega al equipo responsable a través de canales configurados, como correo electrónico o plataformas de mensajería como Slack.

2. **Tipos de Alertas Implementadas**

a) Alertas de Rendimiento

Estas alertas están diseñadas para monitorizar el uso de los recursos clave del sistema y garantizar que el rendimiento se mantenga dentro de los límites aceptables. Las principales métricas monitorizadas son:

- **CPU**: Se genera una alerta cuando el uso de la CPU supera un umbral predefinido durante un período de tiempo determinado. Esto permite al equipo técnico identificar picos de carga y tomar medidas correctivas antes de que afecten al rendimiento global.
- **Memoria**: Las alertas de memoria se activan cuando el consumo de memoria excede un valor crítico. Estas alertas son especialmente importantes en entornos donde los recursos de memoria son limitados.
- **Disco**: Se configuran alertas para advertir cuando el espacio disponible en disco se reduce a niveles peligrosos, evitando así posibles problemas de almacenamiento.

b) Alertas de Disponibilidad

Este tipo de alertas garantiza que los servicios y contenedores desplegados estén operativos en todo momento. Se configuran para supervisar:

- **Servicios web**: Las alertas se activan si un servicio web deja de responder o si devuelve códigos de error específicos, como los errores 500 (fallo del servidor) o 404 (recurso no encontrado).
- **Contenedores Docker**: Se monitoriza el estado de los contenedores Docker, generando alertas cuando un contenedor pasa a estado inactivo, pausado o detenido. Estas alertas son esenciales en entornos que dependen de arquitecturas basadas en microservicios.

Alertas Personalizadas

Además de las alertas estándar, se han configurado alertas específicas adaptadas a las necesidades del grupo de investigación **OEG**. Estas alertas personalizadas supervisan métricas concretas definidas por el equipo, garantizando que se cubran todas las áreas críticas de su infraestructura.

3. Canales de Notificación

En este proyecto, las alertas han sido configuradas inicialmente para enviarse a través de **correo electrónico**, permitiendo al equipo técnico recibir notificaciones directas y detalladas sobre cualquier incidencia. Sin embargo, Grafana ofrece soporte para múltiples canales adicionales, lo que permite una gran flexibilidad a la hora de gestionar las notificaciones. Si en el futuro se requiere ampliar los canales de notificación, Grafana permite configurarlos de manera sencilla, asegurando que las alertas lleguen por el medio más adecuado según la criticidad de la incidencia.

4. Importancia de las Alertas

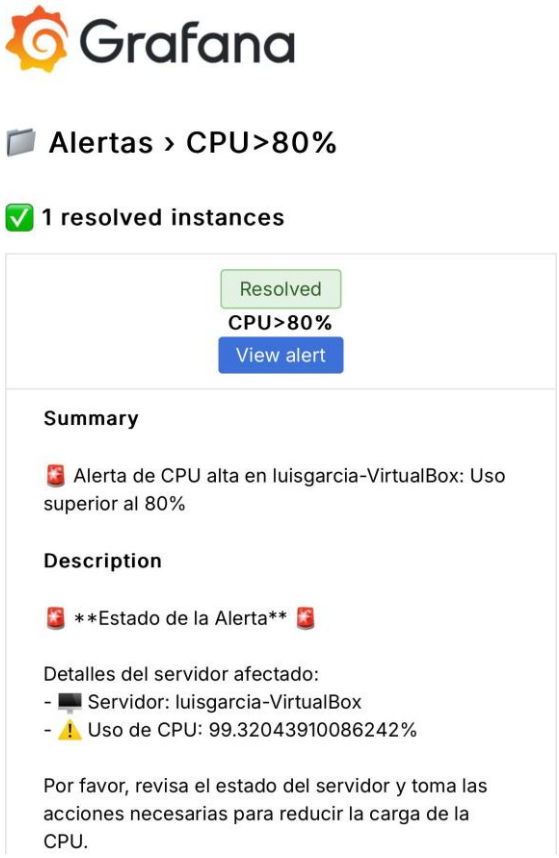
Las alertas desempeñan un papel fundamental en la operativa diaria del sistema, ya que contribuyen a:

- **Reducir tiempos de inactividad**: Al detectar problemas de forma anticipada, se pueden aplicar soluciones antes de que el fallo afecte al sistema en producción.
- **Garantizar la continuidad del servicio**: La rápida notificación de incidencias permite al equipo técnico actuar con celeridad y minimizar el impacto en los usuarios finales.
- **Mejorar la eficiencia operativa**: Al disponer de información detallada y en tiempo real sobre el estado del sistema, se optimiza el uso de los recursos y se facilita la toma de decisiones estratégicas.

2.4.2 Casos de Uso de las Alertas en el Sistema de Monitorización

A continuación, se presentan diversos casos de uso reales de estas alertas, ilustrados mediante ejemplos de escenarios prácticos obtenidos durante las pruebas del sistema. Estos ejemplos muestran cómo las alertas ayudan a identificar problemas en tiempo real, notifican al equipo técnico y confirman la resolución de los incidentes.

- **Caso de Uso 1: Alerta de CPU > 80%**



Escenario: Durante las pruebas, se configuró una alerta para que se activara cuando el uso de CPU de un servidor superara el 80%. Al detectar un uso continuado superior a este umbral, se disparó la alerta.

Mensaje recibido:

- **Alerta:** “Alerta de CPU alta en luisgarcia-VirtualBox: Uso superior al 80%”
- **Detalle:**
    - Servidor afectado: luisgarcia-VirtualBox
    - Uso de CPU registrado: 99.32%

Acción tomada: El equipo técnico recibió la alerta e inició una investigación sobre la causa del alto uso de CPU. Tras tomar las medidas necesarias, como redistribuir la carga de trabajo, el uso de CPU volvió a niveles normales. Posteriormente, se envió una notificación automática indicando la recuperación del sistema.

- **Caso de Uso 2: Alerta de Memoria > 90%**

Grafana

Alertas › Memoria>90%Ejemplo

✅ 1 resolved instances

Resolved

Memoria>90%Ejemplo

View alert

Summary

🧯 Alerta de memoria alta en luisgarcia-VirtualBox: Uso superior al 90%

Description

🧯 **Estado de la Alerta** 🧯

Detalles del servidor afectado:
- 🖥️ Servidor: luisgarcia-VirtualBox
- ⚠️ Uso de memoria: 89.69850498080235%

Por favor, revisa el estado del servidor y libera memoria si es necesario.

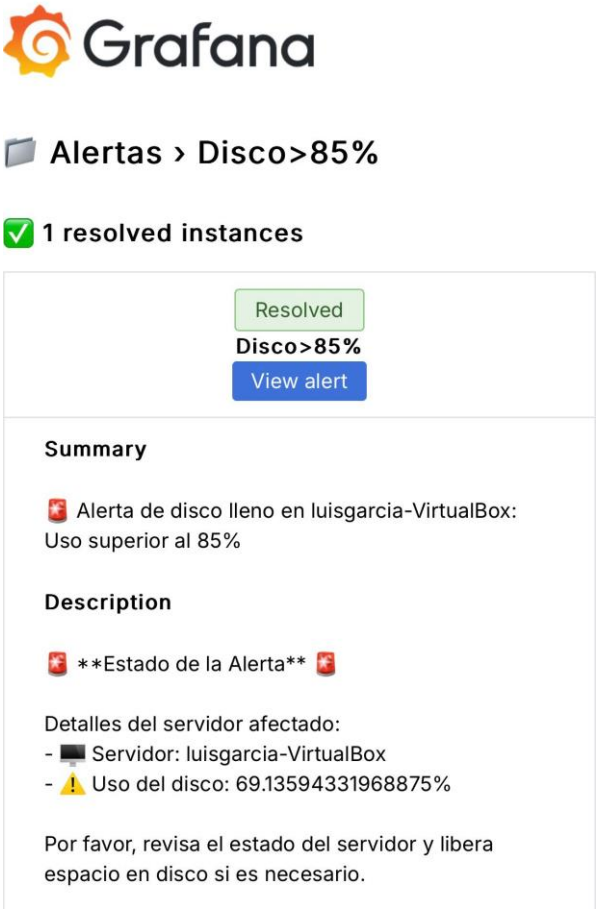
Escenario: Se configuró una alerta para que se activara cuando el uso de memoria de un servidor superara el 90%. Al alcanzar este umbral, se generó y envió una alerta personalizada.

Mensaje recibido:

- **Alerta:** “Alerta de memoria alta en luisgarcia-VirtualBox: Uso superior al 90%”
- **Detalle:**
    - Servidor afectado: luisgarcia-VirtualBox
    - Uso de memoria registrado: 89.69%

Acción tomada: El equipo recibió la notificación y procedió a liberar memoria en el servidor afectado. Tras realizar estas acciones, el sistema envió una nueva alerta confirmando la recuperación de la memoria a niveles seguros.

- **Caso de Uso 3: Alerta de Disco > 85%**



Escenario: Esta alerta se diseñó para activarse cuando el espacio disponible en disco se redujera a menos del 15%, es decir, un uso superior al 85% del almacenamiento total.

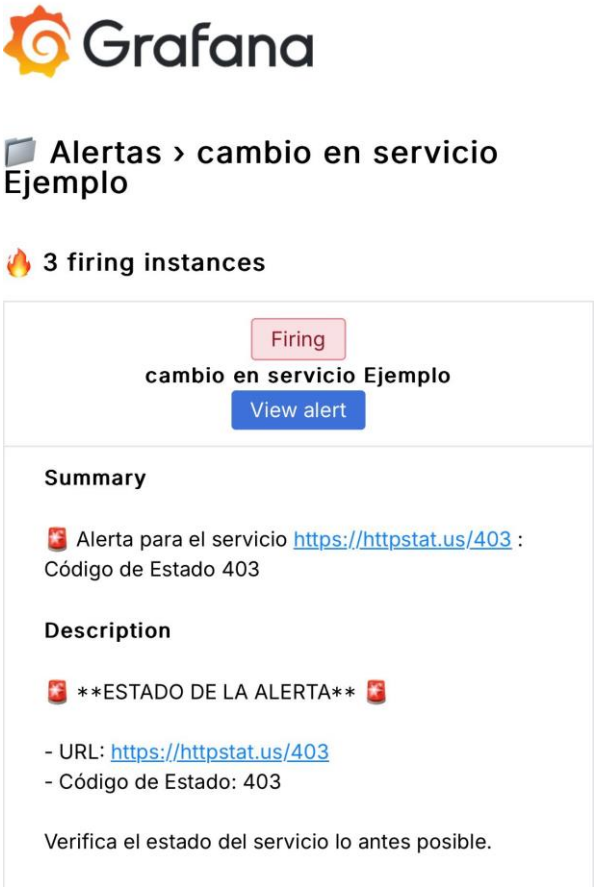
Mensaje recibido:

- **Alerta:** “Alerta de disco lleno en luisgarcia-VirtualBox: Uso superior al 85%”

- **Detalle:**
    - Servidor afectado: luisgarcia-VirtualBox
    - Uso de disco registrado: 69.13%

Acción tomada: Se procedió a liberar espacio en disco eliminando archivos temporales y datos obsoletos. Una vez que el uso de disco volvió a niveles normales, el sistema envió una notificación confirmando la resolución del incidente.

- **Caso de Uso 4: Alerta de Cambio en Servicio Web (HTTP 403)**



Escenario: Se configuró una alerta para supervisar el estado de un servicio web. Durante las pruebas, el sistema detectó que el servicio devolvía un código de estado HTTP 403 (Acceso prohibido), lo que indicaba un problema de acceso al recurso.

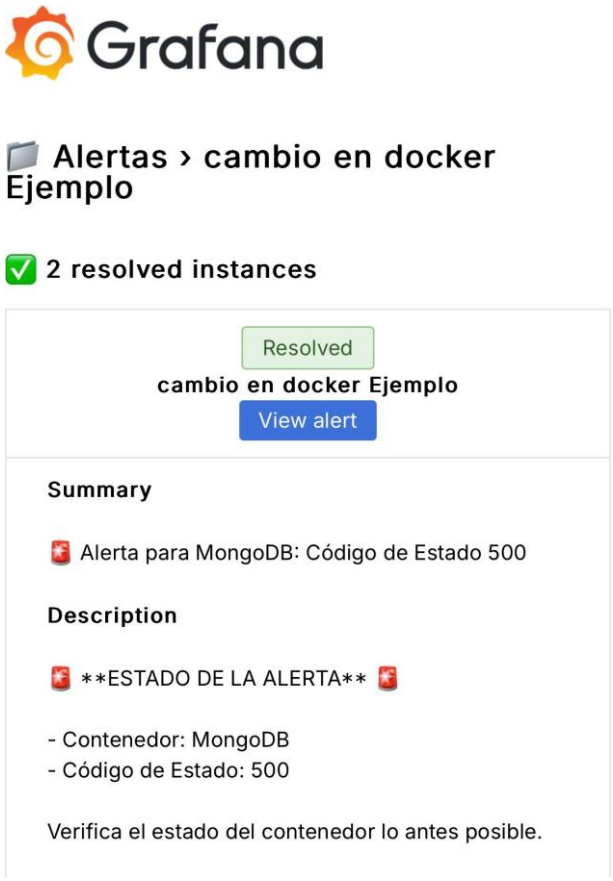
Mensaje recibido:

- **Alerta:** "Alerta para el servicio https://httpstat.us/403: Código de Estado 403"
- **Detalle:**

- URL afectada: https://httpstat.us/403
- Código de estado: 403

Acción tomada: El equipo revisó la configuración del servicio y confirmó que el problema se debía a restricciones de acceso. Después de realizar los ajustes necesarios, el servicio volvió a responder correctamente y se envió una notificación de recuperación.

- **Caso de Uso 5: Alerta de Cambio en Contenedor Docker (Código 500)**



Escenario: Durante las pruebas, se configuró una alerta para monitorizar los contenedores Docker. El sistema detectó que un contenedor de MongoDB devolvía un código de estado 500 (Error interno del servidor), lo que indicaba un fallo en el contenedor.

Mensaje recibido:

- **Alerta:** "Alerta para MongoDB: Código de Estado 500"
- **Detalle:**
  - Contenedor afectado: MongoDB
  - Código de estado: 500

Acción tomada: El equipo técnico verificó el estado del contenedor y detectó un problema en la configuración interna del mismo. Tras reiniciar el contenedor y ajustar los parámetros necesarios, el sistema recuperó su funcionalidad y se envió una notificación de recuperación.

- **Resumen de los Casos de Uso**

Las alertas configuradas en Grafana demostraron ser una herramienta eficaz para garantizar la estabilidad y el rendimiento del sistema de monitorización. Los casos de uso descritos muestran cómo las alertas permitieron al equipo técnico:

- Detectar y resolver problemas de alto consumo de recursos (CPU, memoria y disco).
- Supervisar la disponibilidad y el correcto funcionamiento de servicios web y contenedores Docker.
- Recibir notificaciones automáticas de recuperación, asegurando un seguimiento continuo de los incidentes.

Gracias a estas alertas, el equipo pudo actuar de manera proactiva y minimizar el impacto de los incidentes en el sistema, mejorando así la continuidad operativa y la experiencia del usuario final.

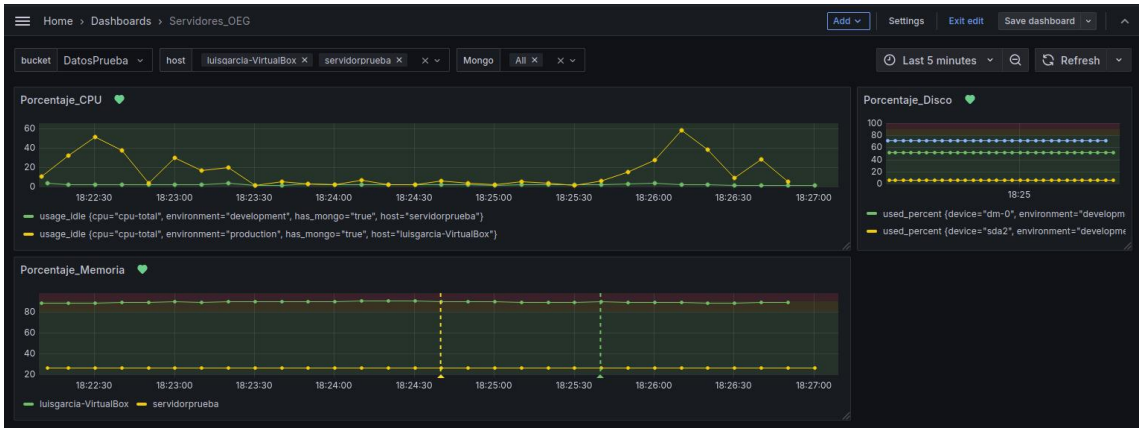
Conclusión

La configuración de alertas en Grafana ha sido uno de los pilares fundamentales de este proyecto de monitorización. Gracias a ellas, se ha logrado crear un sistema robusto y proactivo que garantiza la estabilidad y el rendimiento de la infraestructura monitorizada. La posibilidad de definir alertas personalizadas, junto con la flexibilidad en los canales de notificación, permite adaptar el sistema a las necesidades específicas de cualquier entorno, asegurando una supervisión eficaz y una rápida respuesta ante posibles incidencias.

2.5 Explicacion de los Dashboards

2.5.1 Dashboard de Servidores

2.5.1.1 Descripcion General del Dashboard



El **Dashboard de Servidores** está diseñado para proporcionar una visión detallada y en tiempo real del estado y rendimiento de los servidores monitorizados. Este dashboard es fundamental para el equipo de operaciones, ya que permite identificar rápidamente cualquier anomalía en el uso de recursos críticos como la **CPU**, la **memoria** y el **disco**.

La visualización se compone de tres paneles principales, cada uno de los cuales representa una métrica clave de los servidores, lo que facilita el análisis y la comparación de los diferentes nodos de la infraestructura. Cada panel está configurado para mostrar las métricas en formato de **serie temporal**, lo que permite seleccionar diferentes rangos de tiempo y visualizar la evolución histórica de cada recurso.

Además, este dashboard incluye **variables de filtro** que ofrecen flexibilidad y personalización en la visualización de datos. Las variables permiten filtrar por:

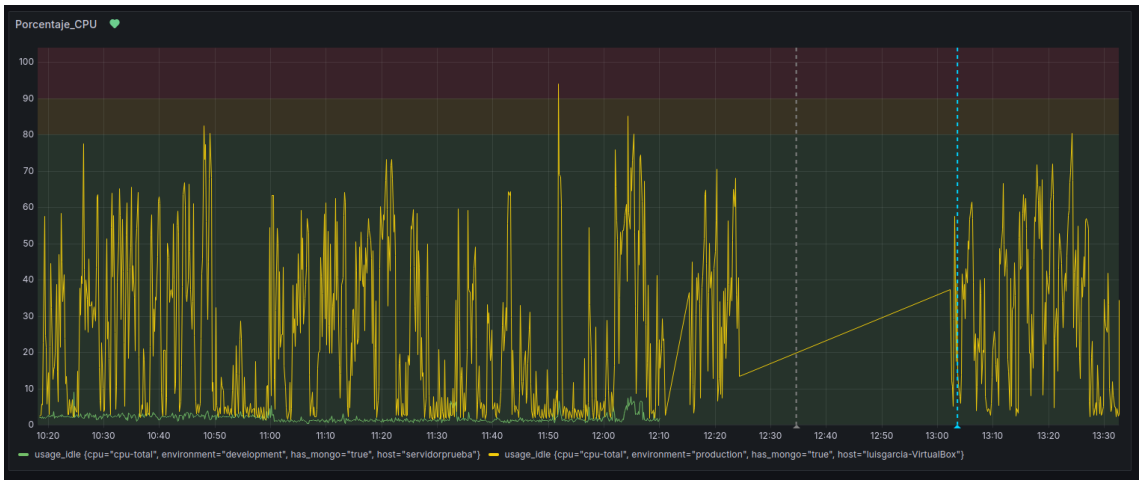
- **Host**: Selección de uno o varios servidores específicos.
- **Mongo**: Filtrar los servidores que tienen instalado MongoDB.
- **Bucket**: Seleccionar el bucket desde el que se extraen los datos, evitando modificar manualmente el dashboard cuando se cambia de bucket.

El dashboard también está preparado para soportar alertas configuradas en Grafana. Estas alertas se activan automáticamente cuando una métrica crítica supera un umbral definido, permitiendo al equipo técnico reaccionar de manera proactiva ante posibles problemas.

Este enfoque integral garantiza una monitorización eficiente de los servidores y facilita la detección temprana de incidentes, contribuyendo así a mejorar la continuidad operativa y el rendimiento de los servicios alojados.

Paneles y Descripción

2.5.1.2 Panel de CPU:

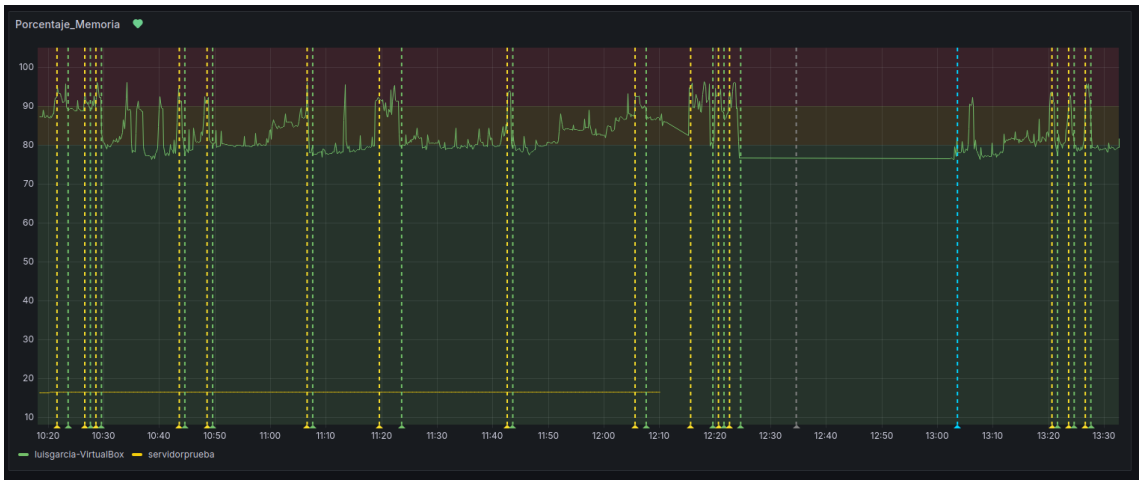


Este panel muestra el uso de CPU de cada servidor en un rango de tiempo definido.
La consulta utilizada para este panel es:

```
from(bucket: "${bucket}")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["host"] =~ /${host:regex}/)
  |> filter(fn: (r) => r["has_mongo"] =~ /${Mongo:regex}/)
  |> filter(fn: (r) => r["cpu"] == "cpu-total")
  |> filter(fn: (r) => r["_field"] == "usage_idle")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> map(fn: (r) => ({ r with _value: 100.0 - r._value }))
  |> yield(name: "mean")
```

- La consulta calcula el porcentaje de uso de CPU, restando el valor de "idle" al 100%.
- Este panel es una **serie temporal**, lo que permite seleccionar cualquier espacio de tiempo y visualizar cómo ha variado el uso de CPU durante el periodo seleccionado.
- Elegir una **serie temporal** permite también la comparación de los diferentes servidores en el mismo rango de tiempo.
- Al estar representado en **porcentajes**, se obtiene una idea visual rápida del uso de CPU. Además, el fondo del panel cambia de color según el nivel de uso: verde para menos del 80%, amarillo entre 80% y 90%, y rojo cuando supera el 90%.
- Además, si se configuran alertas vinculadas al panel, es posible identificar fácilmente los momentos en los que se han activado o recuperado dichas alertas.

2.5.1.3 Panel de Memoria:

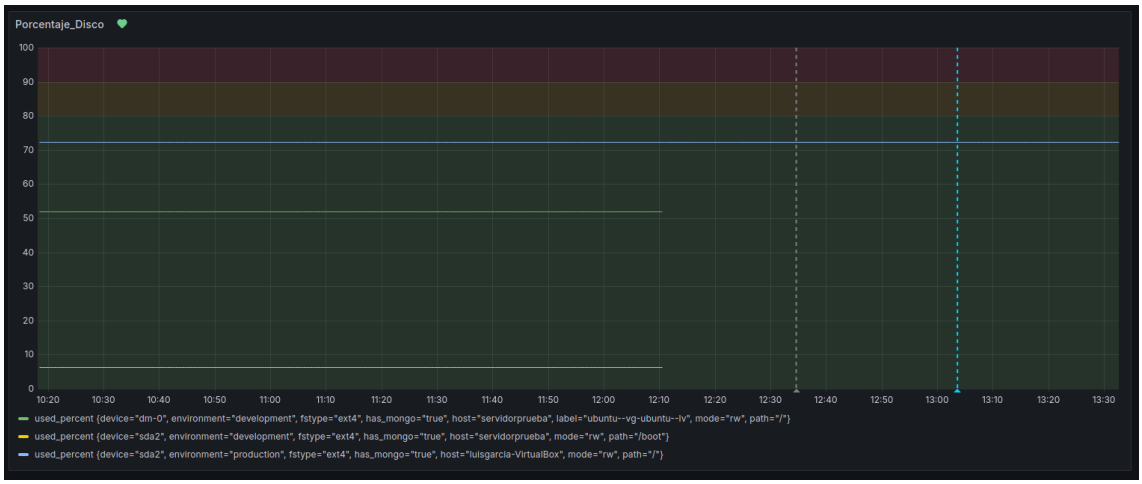


Este panel muestra el porcentaje de memoria utilizada en cada servidor.
La consulta utilizada para este panel es:

```
from(bucket: "${bucket}")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "mem")
  |> filter(fn: (r) => r["_field"] == "free" or r["_field"] == "used")
  |> filter(fn: (r) => r["host"] =~ /${host:regex}/)
  |> filter(fn: (r) => r["has_mongo"] =~ /${Mongo:regex}/)
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty:
false)
  |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn:
"_value")
  |> map(fn: (r) => ({
        _time: r._time,
        _value: float(v: r["used"]) / (float(v: r["free"]) + float(v:
r["used"])) * 100.0,
        host: r.host
      }))
  |> yield(name: "mean")
```

- La consulta calcula el porcentaje de memoria usada dividiendo la memoria usada entre el total disponible (usada + libre).
- Este panel también es una **serie temporal**, lo que facilita el análisis histórico y la detección de patrones de uso.
- Elegir una **serie temporal** permite también la comparación de los diferentes servidores en el mismo rango de tiempo.
- Al estar representado en **porcentajes**, se obtiene una idea visual rápida del uso de memoria, con el mismo sistema de colores: verde para menos del 80%, amarillo entre 80% y 90%, y rojo cuando supera el 90%.

2.5.1.4 Panel de Disco:



Este panel muestra el porcentaje de espacio de disco utilizado en cada servidor.
La consulta utilizada para este panel es:

```
from(bucket: "${bucket}")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "disk")
  |> filter(fn: (r) => r["_field"] == "used_percent" or r["_field"] ==
"free_percent")
  |> filter(fn: (r) => r["host"] =~ /${host:regex}/)
  |> filter(fn: (r) => r["has_mongo"] =~ /${Mongo:regex}/)
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty:
false)
  |> yield(name: "mean")
```

- La consulta selecciona los campos de porcentaje de disco usado y libre, y muestra el promedio de uso en el rango de tiempo seleccionado.
- Al igual que los paneles anteriores, es una **serie temporal**, lo que permite una visualización clara y sencilla del uso de disco a lo largo del tiempo.
- Elegir una **serie temporal** permite también la comparación de los diferentes servidores en el mismo rango de tiempo.
- Al estar representado en **porcentajes**, se obtiene una idea visual rápida del uso de disco, con el mismo sistema de colores: verde para menos del 80%, amarillo entre 80% y 90%, y rojo cuando supera el 90%.

2.5.1.5 Variables de Filtro

Este dashboard incluye tres variables clave de filtro que permiten al usuario personalizar y ajustar la visualización de métricas según sus necesidades

específicas. Estas variables se configuran en Grafana y permiten una interacción dinámica con los datos almacenados en InfluxDB.

Host: Permite seleccionar uno o varios servidores específicos.
Mongo: Permite filtrar los servidores según si tienen MongoDB instalado o no, gracias a la configuración de global_tags en el archivo telegraf.conf.
Bucket: Permite seleccionar el bucket desde el cual se extraerán los datos, evitando la necesidad de modificar el dashboard manualmente cuando se cambian los buckets.

2.5.1.6 Ventajas del Dashboard

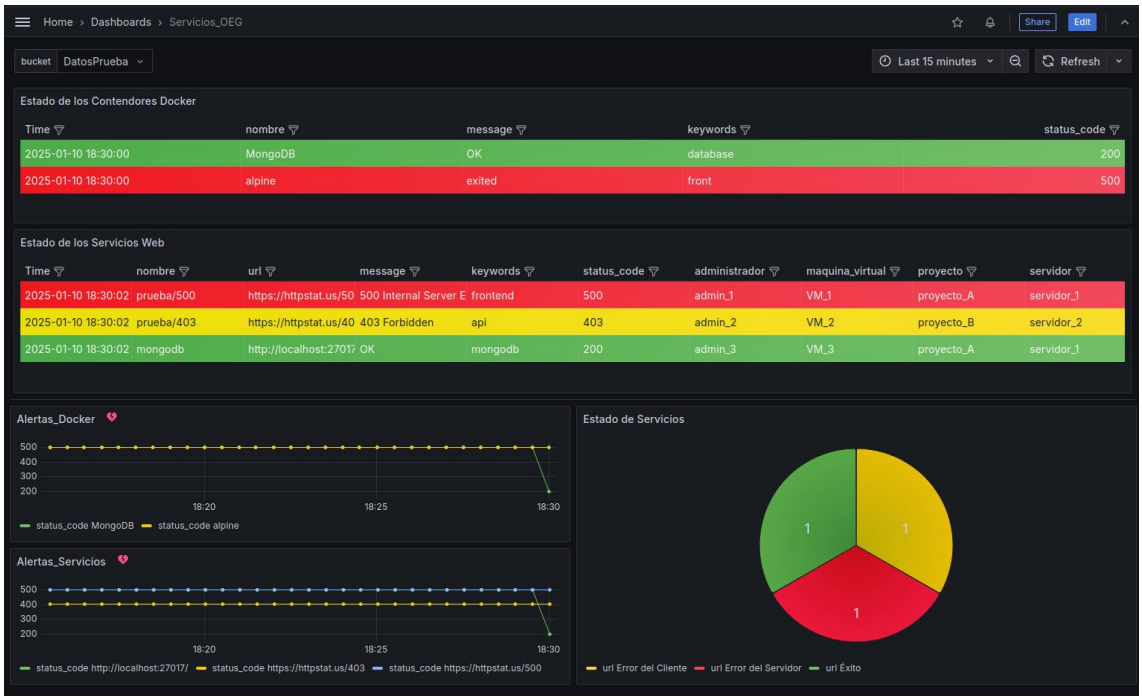
El **Dashboard de Servidores** ofrece múltiples ventajas tanto en términos de visualización como de análisis de datos, lo que lo convierte en una herramienta esencial para la monitorización continua de la infraestructura.

1. **Visualización en serie temporal**
   Al utilizar paneles de serie temporal, el usuario puede seleccionar cualquier espacio de tiempo y visualizar la evolución de las métricas en función de periodos históricos. Esto facilita el análisis de tendencias, la detección de anomalías y el estudio del comportamiento de los servidores en periodos específicos.
2. **Configuración flexible y personalizable**
   Gracias a las **variables de filtro dinámicas**, el dashboard es altamente configurable y adaptable a cualquier entorno. Esto permite que el usuario seleccione servidores, buckets o servicios específicos sin necesidad de editar manualmente el dashboard.
3. **Compatibilidad con alertas avanzadas**
   El dashboard permite vincular alertas configuradas en Grafana, lo que garantiza que el usuario pueda visualizar directamente en el dashboard los momentos en los que se han activado o recuperado las alertas. Esto mejora la capacidad de respuesta ante incidencias críticas.
4. **Indicadores visuales**
   La representación de las métricas mediante porcentajes, combinada con un sistema de colores en el fondo del panel, proporciona una visualización intuitiva. Los colores cambian automáticamente según los niveles de uso:
   - **Verde**: Menos del 80% de uso.
   - **Amarillo**: Entre 80% y 90% de uso.
   - **Rojo**: Más del 90% de uso.
5. **Detección proactiva de problemas**
   Gracias a la integración de alertas, el equipo de operaciones puede detectar y actuar de manera proactiva ante problemas potenciales antes de que afecten a los servicios de producción.
6. **Optimización del rendimiento**
   Al monitorizar métricas clave como CPU, memoria y disco, el equipo técnico puede tomar decisiones fundamentadas para optimizar el rendimiento de los servidores y prevenir situaciones de sobrecarga.

7. **Escalabilidad del sistema de monitorización**
   El dashboard está diseñado para ser escalable, lo que significa que, al añadir nuevos servidores o servicios, estos se reflejarán automáticamente en el dashboard sin necesidad de realizar configuraciones adicionales. Esto es posible gracias al uso de variables dinámicas y la arquitectura centralizada de InfluxDB.

2.5.2 Dashboard de Servicios

2.5.2.1 Descripcion General del Dashboard



El **Dashboard de Servicios** está orientado a monitorizar el estado y el rendimiento de los servicios web y aplicaciones desplegados en los servidores. Proporciona una vista detallada de parámetros críticos como la disponibilidad de los servicios, los tiempos de respuesta, y el estado de los contenedores Docker que alojan dichos servicios. Este dashboard permite a los equipos de operaciones y desarrollo detectar rápidamente problemas de rendimiento o interrupciones en los servicios esenciales.

Este dashboard ha sido diseñado pensando en la flexibilidad y escalabilidad, permitiendo añadir nuevos servicios sin necesidad de modificar manualmente el dashboard. Esto se logra gracias a la configuración basada en los archivos JSON (servicios.json y dockers.json), los cuales Telegraf lee y procesa automáticamente.

Al igual que el **Dashboard de Servidores**, este dashboard incluye paneles configurados como series temporales y variables de filtro dinámicas que facilitan el análisis comparativo y personalizado.

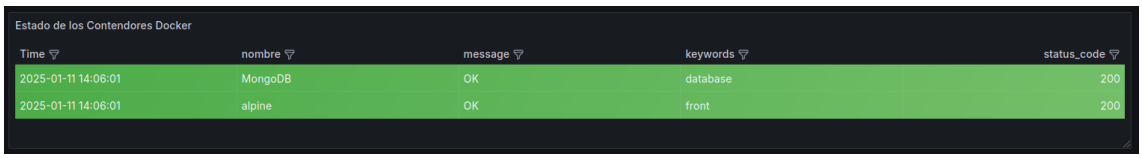
Los principales paneles de este dashboard son:

- **Estado de los servicios web:** Muestra si un servicio está activo o inactivo.
- **Tiempo de respuesta de los servicios:** Visualiza el tiempo de respuesta promedio y máximo de cada servicio monitorizado.
- **Estado de los contenedores Docker:** Indica si los contenedores están corriendo, pausados o detenidos, junto con el consumo de recursos como CPU y memoria.

Este enfoque permite al equipo técnico identificar, analizar y resolver problemas antes de que afecten a los usuarios finales, garantizando así la disponibilidad continua de los servicios.

Paneles y Descripción

2.5.2.2 Estado de los Contenedores Docker (Tabla):



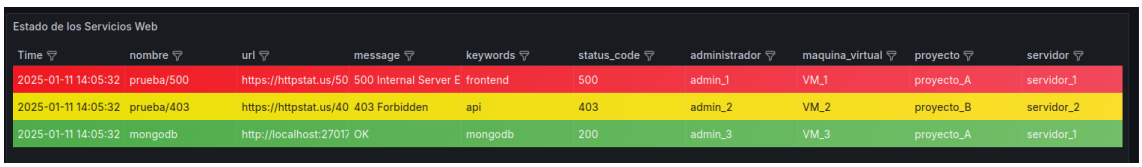
Este panel muestra el estado actual de los contenedores Docker.
Se utiliza un código de color para representar el estado:

- **Verde**: Contenedor activo.
- **Rojo**: Contenedor caído.

La consulta utilizada para este panel es:

```
from(bucket: "${bucket}")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "docker_gen")
  |> filter(fn: (r) => r["_field"] == "message" or r["_field"] ==
"status_code" or r["_field"] == "keywords")
  |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn:
"_value")
  |> group()
  |> sort(columns: ["_time"], desc: true)
  |> unique(column: "nombre")
  |> keep(columns: ["_time", "nombre", "keywords", "status_code",
"message"])
```

2.5.2.3 Estado de los Servicios Web (Tabla):



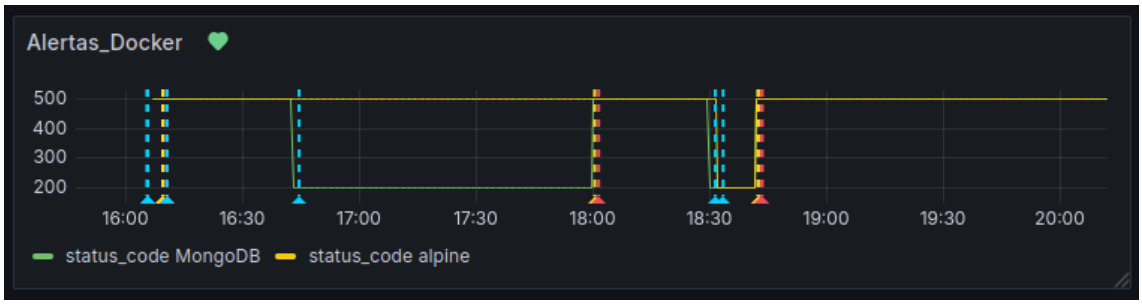
Este panel muestra el estado actual de los servicios web.
Se utiliza un código de color para representar el estado según el código de respuesta HTTP:

- **Verde**: Respuesta correcta (2xx).
- **Amarillo**: Error del cliente (4xx).
- **Rojo**: Error del servidor (5xx).

La consulta utilizada para este panel es:

```
from(bucket: "${bucket}")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "servicio_gen")
  |> filter(fn: (r) => r["_field"] == "message" or r["_field"] ==
"status_code")
  |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn:
"_value")
  |> group()
  |> sort(columns: ["_time"], desc: true)
  |> unique(column: "url")
  |> keep(columns: ["_time", "url", "nombre", "keywords",
"status_code", "message", "servidor", "proyecto", "maquina_virtual",
"administrador"])
```

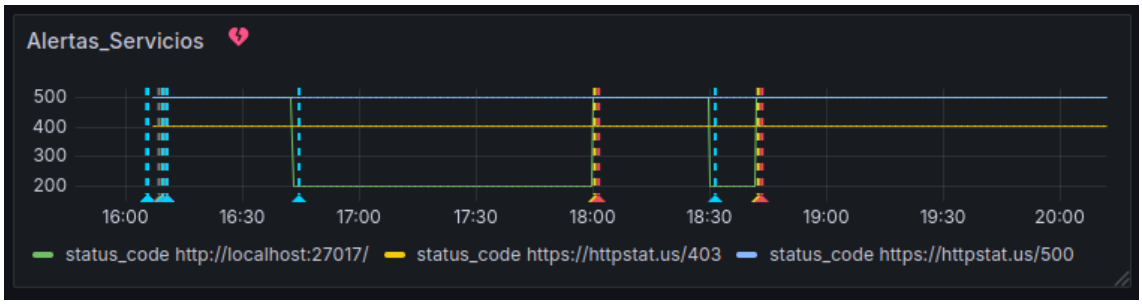
2.5.2.4 Alertas de Docker:



Este panel muestra la evolución de los códigos de estado de los contenedores Docker a lo largo del tiempo.
La consulta utilizada es:

```
from(bucket: "DatosPrueba")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "docker_gen")
  |> filter(fn: (r) => r["_field"] == "status_code")
  |> pivot(rowKey: ["_time"], columnKey: ["_field"], valueColumn:
"_value")
  |> keep(columns: ["_time", "nombre", "status_code"])
```

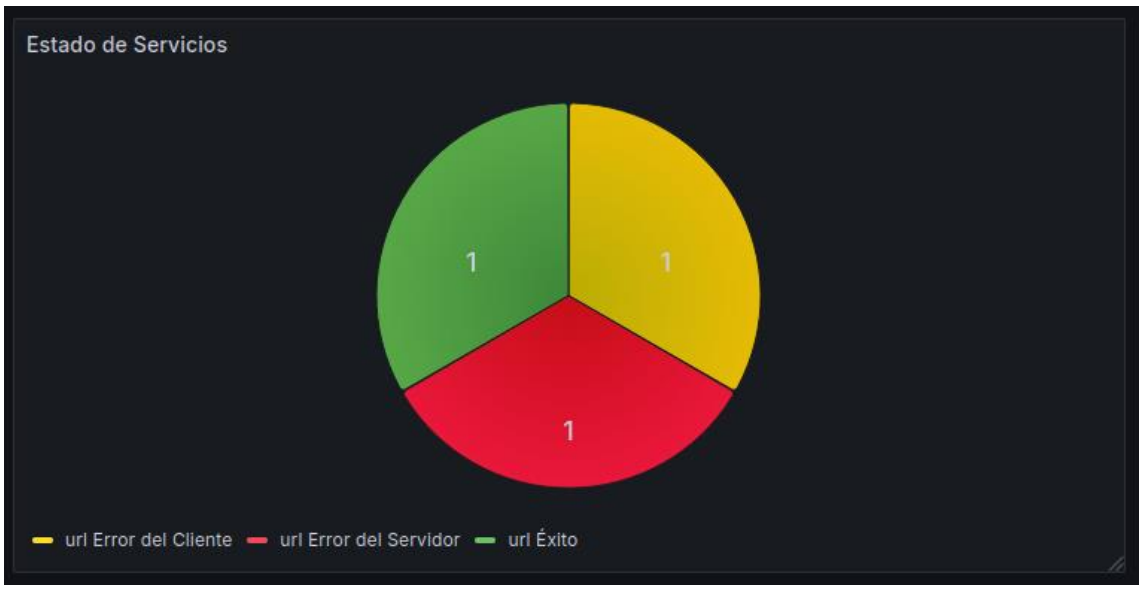
2.5.2.5 Alertas de Servicios:



Este panel muestra la evolución de los códigos de estado de los servicios web a lo largo del tiempo.
La consulta utilizada es:

```
from(bucket: "DatosPrueba")
    |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
    |> filter(fn: (r) => r["_measurement"] == "servicio_gen")
    |> filter(fn: (r) => r["_field"] == "status_code")
    |> pivot(rowKey: ["_time"], columnKey: ["_field"], valueColumn:
"_value")
    |> keep(columns: ["_time", "url", "status_code"])
```

2.5.2.6 Panel de Estado de Servicios (Queso)



Este panel muestra una representación gráfica del estado general de los servicios mediante un gráfico tipo "queso".

La consulta utilizada es:

```
from(bucket: "${bucket}")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "servicio_gen")
  |> filter(fn: (r) => r["_field"] == "status_code")
  |> group(columns: ["url"])
  |> sort(columns: ["_time"], desc: true)
  |> unique(column: "url")
  |> map(fn: (r) => ({
       r with
       _value: if r._value < 300 then "Éxito"
               else if r._value < 400 then "Redirección"
               else if r._value < 500 then "Error del Cliente"
               else "Error del Servidor"
     }))
  |> group(columns: ["_value"])
  |> count(column: "url")
  |> yield(name: "status_distribution")
```

2.5.2.7 Ventajas del Dashboard

1. **Detección rápida de problemas en los servicios**
   Gracias a la monitorización en tiempo real del estado de los servicios y contenedores Docker, el equipo de operaciones puede identificar rápidamente cuándo un servicio deja de estar disponible o cuándo un contenedor presenta un comportamiento anómalo.
2. **Análisis detallado del rendimiento**
   Los paneles de tiempo de respuesta permiten al equipo analizar cómo varía el rendimiento de los servicios a lo largo del tiempo, facilitando la detección de picos de latencia o problemas de red.
3. **Automatización en la incorporación de nuevos servicios**
   La configuración dinámica a través de los archivos JSON permite que el sistema detecte automáticamente nuevos servicios y contenedores Docker sin necesidad de modificar el dashboard manualmente. Esto reduce significativamente el tiempo y el esfuerzo requeridos para mantener la monitorización actualizada.
4. **Compatibilidad con alertas personalizadas**
   El dashboard permite configurar alertas avanzadas en Grafana para parámetros críticos, como el tiempo de respuesta o la disponibilidad de un servicio. Estas alertas se pueden vincular al dashboard, mostrando visualmente los momentos en los que se activan o recuperan.
5. **Visualización intuitiva del estado de los servicios**
   El uso de indicadores visuales, como colores y gráficos de estado, facilita una comprensión rápida del estado general de los servicios.
   - **Verde:** El servicio está funcionando correctamente.

- **Amarillo:** El servicio presenta problemas moderados, como un tiempo de respuesta elevado.
- **Rojo:** El servicio está caído o presenta problemas graves.

6. **Optimización del despliegue de aplicaciones**
   Al monitorizar el estado de los contenedores Docker y su consumo de recursos, el equipo técnico puede identificar contenedores que consumen más recursos de lo esperado o que podrían estar causando problemas de rendimiento en el servidor.
7. **Escalabilidad garantizada**
   Este dashboard está preparado para escalar junto con la infraestructura. Cada vez que se despliega un nuevo servicio o contenedor, este se añade automáticamente al sistema de monitorización, lo que permite mantener un control total sin necesidad de configuraciones adicionales.
8. **Reducción de tiempos de inactividad**
   La detección proactiva de problemas y la posibilidad de configurar alertas automáticas contribuyen a reducir los tiempos de inactividad de los servicios, mejorando así la continuidad operativa y la experiencia del usuario final.

2.5.3 Dashboard Tecnico de Docker

2.5.3.1 Descripcion General del Dashboard

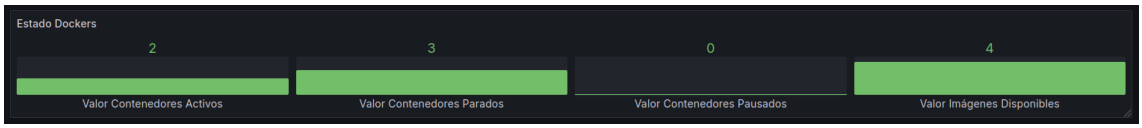
El **Dashboard de Métricas Docker** ha sido diseñado específicamente para monitorizar contenedores Docker desplegados en los servidores sin necesidad de instalar Python. A diferencia del **Dashboard de Servicios**, donde un servidor central con Python gestiona las consultas a los servicios, este dashboard se apoya únicamente en el plugin nativo de Docker proporcionado por Telegraf, lo que simplifica su implementación y aumenta la portabilidad del sistema. Este dashboard ofrece información detallada y en tiempo real sobre el estado de los contenedores Docker, incluyendo su consumo de CPU, memoria, disco y red. Gracias a su arquitectura, los paneles se actualizan automáticamente a medida

que se despliegan nuevos contenedores, sin necesidad de modificar la configuración del dashboard manualmente.
Algunos de los paneles clave incluyen:
Estado general de contenedores: Proporciona un resumen visual del número de contenedores activos, pausados y detenidos, así como el número total de imágenes disponibles en el servidor.
Consumo de recursos: Paneles dedicados a monitorizar el uso de CPU, memoria, disco y red de los contenedores, presentados en formato de serie temporal para facilitar el análisis de tendencias.
Este enfoque permite una monitorización eficiente de los contenedores en entornos dinámicos, asegurando que el equipo técnico disponga de la información necesaria para gestionar y optimizar el rendimiento de los contenedores en producción.
Limitaciones del Dashboard:

- No tiene la posibilidad de crear una tabla que indique qué contenedores están activos y cuáles no.
- Para saber qué contenedores están activos, se debe revisar visualmente los paneles de series temporales y comprobar si hay métricas disponibles para cada contenedor.

Paneles y Descripción

2.5.3.2 Estado Dockers (Barras):



Este panel muestra información general sobre el estado de los contenedores Docker, incluyendo:

- **Contenedores Activos**.
- **Contenedores Parados**.
- **Contenedores Pausados**.
- **Imágenes Disponibles**.

La consulta utilizada para este panel es:

```
union(
  tables: [
    from(bucket: "${bucket}")
      |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
      |> filter(fn: (r) => r["_measurement"] == "docker")
      |> filter(fn: (r) => r["_field"] == "n_containers_running")
      |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty:
false)
      |> map(fn: (r) => ({ _value: r._value, metric: "Contenedores
Activos" })),
    from(bucket: "${bucket}")
```

```
      |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
      |> filter(fn: (r) => r["_measurement"] == "docker")
      |> filter(fn: (r) => r["_field"] == "n_containers_stopped")
      |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty:
false)
      |> map(fn: (r) => ({ _value: r._value, metric: "Contenedores
Parados" })),
    from(bucket: "${bucket}")
      |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
      |> filter(fn: (r) => r["_measurement"] == "docker")
      |> filter(fn: (r) => r["_field"] == "n_containers_paused")
      |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty:
false)
      |> map(fn: (r) => ({ _value: r._value, metric: "Contenedores
Pausados" })),
    from(bucket: "${bucket}")
      |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
      |> filter(fn: (r) => r["_measurement"] == "docker")
      |> filter(fn: (r) => r["_field"] == "n_images")
      |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty:
false)
      |> map(fn: (r) => ({ _value: r._value, metric: "Imágenes
Disponibles" }))
  ]
)
  |> rename(columns: { _value: "Valor", metric: "Métrica" })
  |> group(columns: ["Métrica"])
```

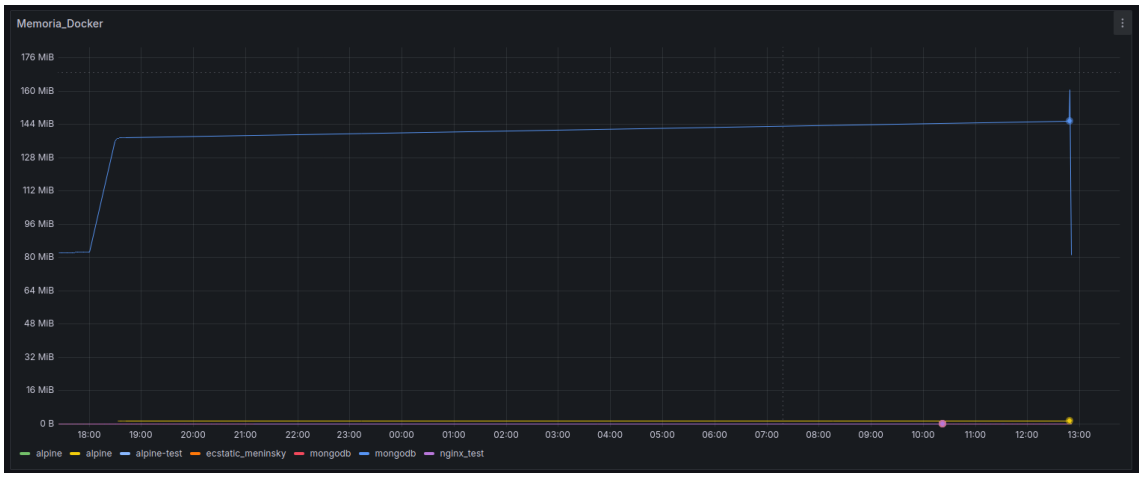
2.5.3.3 CPU Docker (Serie Temporal):

Este panel muestra el uso de CPU de los contenedores Docker a lo largo del tiempo.

La consulta utilizada es:

```
from(bucket: "${bucket}")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "docker_container_cpu")
  |> filter(fn: (r) => r["_field"] == "usage_percent")
  |> filter(fn: (r) => r["container_name"] =~ /${container:regex}/)
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty:
false)
  |> yield(name: "mean")
```

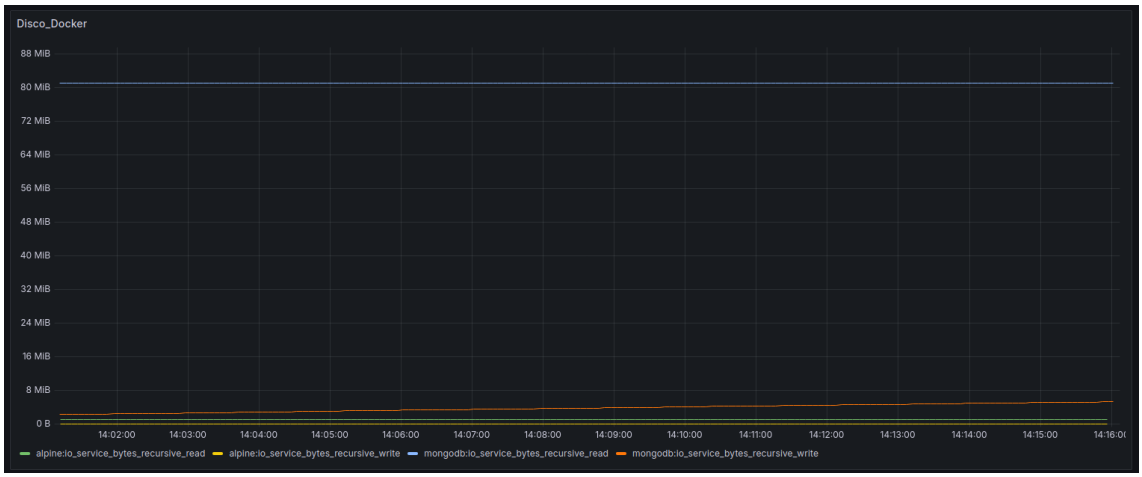
2.5.3.4 Memoria Docker (Serie Temporal):



Este panel muestra el consumo de memoria de los contenedores Docker.
La consulta utilizada es:

```
from(bucket: "${bucket}")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "docker_container_mem")
  |> filter(fn: (r) => r["_field"] == "usage")
  |> filter(fn: (r) => r["container_name"] =~ /${container:regex}/)
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty:
false)
  |> yield(name: "mean")
```

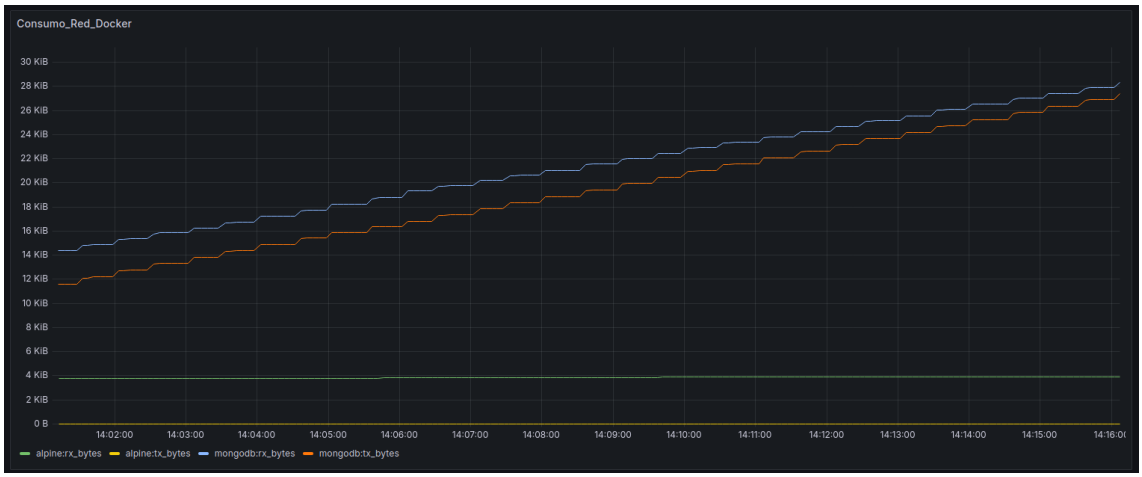
2.5.3.5 Disco Docker (Serie Temporal):



Este panel muestra el uso de disco de los contenedores Docker.
La consulta utilizada es:

```
from(bucket: "${bucket}")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "docker_container_blkio")
  |> filter(fn: (r) => r["_field"] ==
"io_service_bytes_recursive_read" or r["_field"] ==
"io_service_bytes_recursive_write")
  |> filter(fn: (r) => r["container_name"] =~ /${container:regex}/)
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty:
false)
  |> yield(name: "mean")
```

2.5.3.6 Consumo de Red Docker (Serie Temporal):



Este panel muestra el consumo de red de los contenedores Docker, tanto en tráfico de entrada (**rxbytes) como de salida (txbytes**).
La consulta utilizada es:

```
from(bucket: "${bucket}")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "docker_container_net")
  |> filter(fn: (r) => r["_field"] == "rx_bytes" or r["_field"] ==
"tx_bytes")
  |> filter(fn: (r) => r["container_name"] =~ /${container:regex}/)
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty:
false)
  |> yield(name: "mean")
```

2.5.3.7 Ventajas del Dashboard

1. **Monitorización completa sin dependencias adicionales**
   Al no requerir Python ni scripts personalizados, este dashboard puede implementarse fácilmente en cualquier entorno que utilice contenedores Docker. Esto facilita la adopción del sistema en entornos donde la instalación de dependencias externas es un problema.
2. **Resumen visual del estado de los contenedores**
   El panel de estado general permite visualizar rápidamente cuántos contenedores están activos, pausados o detenidos, así como el número de imágenes disponibles. Esta información es clave para mantener un control adecuado de los recursos desplegados.
3. **Análisis detallado del uso de recursos**
   Los paneles de CPU, memoria, disco y red proporcionan una vista detallada de cómo los contenedores consumen los recursos del servidor. Esto permite identificar contenedores que podrían estar consumiendo más recursos de lo esperado o generando cuellos de botella en la infraestructura.
4. **Visualización en serie temporal**
   Al utilizar paneles configurados como series temporales, el usuario puede seleccionar diferentes rangos de tiempo y comparar el comportamiento de los contenedores en distintos periodos. Esto facilita la detección de patrones de uso y la toma de decisiones basadas en datos históricos.
5. **Filtros avanzados y personalización**
   El dashboard incluye variables de filtro dinámicas que permiten seleccionar contenedores específicos o visualizar todos los contenedores simultáneamente. Además, es posible filtrar por servidor, lo que resulta especialmente útil en entornos con múltiples nodos monitorizados.
6. **Escalabilidad y automatización**
   Gracias al uso del plugin nativo de Docker de Telegraf y la configuración dinámica de los paneles, este dashboard es altamente escalable. A medida que se despliegan nuevos contenedores, estos se

reflejan automáticamente en el dashboard sin necesidad de configuraciones manuales adicionales.

7. **Detección proactiva de problemas**
   La integración con el sistema de alertas de Grafana permite configurar notificaciones que se activan cuando un contenedor supera ciertos umbrales críticos de uso de recursos. Esto ayuda al equipo técnico a reaccionar de manera proactiva ante posibles problemas antes de que afecten a los servicios.
8. **Optimización del rendimiento de contenedores**
   Al disponer de información detallada sobre el consumo de recursos de cada contenedor, el equipo de operaciones puede identificar y ajustar los recursos asignados a cada contenedor, optimizando el rendimiento general de la infraestructura y garantizando un uso eficiente de los servidores.

2.6 Codigo Utilizado

En este apartado se describen los principales scripts y archivos de configuración utilizados para el sistema de monitorización. Estos scripts automatizan la recolección de métricas y el monitoreo de servicios web y contenedores Docker, permitiendo la visualización de estas métricas en los dashboards correspondientes de Grafana. Además, se detallan los motivos por los cuales estos códigos son necesarios y su función en el sistema de monitorización.

2.6.1 Scripts principales

2.6.1.1 monitor_docker.py

Este script se encarga de monitorizar el estado de los contenedores Docker configurados en el archivo dockers.json. Su función principal es comprobar si los contenedores están activos, pausados o detenidos, y generar una salida que Telegraf pueda procesar mediante el plugin exec. Esta información es fundamental para alimentar el **Dashboard de Docker**, que muestra el estado de los contenedores y sus métricas.

Código en anexo.

Explicación técnica:

1. **Carga de configuración:** El script carga el archivo dockers.json, que contiene la lista de contenedores a monitorizar.
2. **Conexión al cliente Docker:** Utiliza la biblioteca docker para conectarse al socket local de Docker y obtener información sobre los contenedores.
3. **Iteración sobre contenedores:** Por cada contenedor definido en el archivo de configuración:
   - Si el contenedor está en ejecución, genera una salida con un código de estado 200.

- Si el contenedor no está en ejecución, genera una salida con un código de error 500 o 404 según corresponda.

4. **Gestión de errores:** Captura y maneja posibles errores relacionados con la conexión a Docker o la lectura del archivo de configuración.

Función del script en el sistema: Este script es esencial para el **Dashboard de Docker**, ya que proporciona información sobre el estado de los contenedores en tiempo real. Los paneles de este dashboard muestran cuántos contenedores están activos, detenidos o pausados, permitiendo al equipo técnico tener una visión clara de la infraestructura de contenedores.

2.6.1.2 monitor_servicio.py

Este script se encarga de monitorizar el estado de los servicios web configurados en el archivo servicios.json. Utiliza la biblioteca requests para realizar peticiones HTTP a las URLs de los servicios y generar una salida que Telegraf procesa mediante el plugin exec. Esta información es utilizada en el **Dashboard de Servicios**, que muestra el estado de los servicios web y sus respuestas.

Código en anexo.

Explicación técnica:

1. **Carga de configuración:** El script carga el archivo servicios.json, que contiene la lista de servicios web a monitorizar.
2. **Iteración sobre servicios:** Por cada servicio definido en el archivo de configuración, realiza una petición HTTP a la URL correspondiente.
3. **Gestor de excepciones:** Maneja errores como tiempo de espera agotado o problemas de conexión y devuelve un código de error según corresponda.

Función del script en el sistema: Este script es vital para el **Dashboard de Servicios**, ya que permite visualizar el estado de los servicios web en tiempo real. Además, permite identificar rápidamente si un servicio ha dejado de responder o presenta errores.

2.6.2 Archivos JSON utilizados

2.6.2.1 dockers.json

Este archivo contiene la configuración de los contenedores Docker que deben ser monitorizados. Su estructura permite definir el nombre del contenedor, un alias descriptivo y palabras clave asociadas para su filtrado en los dashboards.

```
[
    {
```

```
    "container_name": "mongodb",
    "alias": "MongoDB",
    "keywords": "database"
  },
  {
    "container_name": "alpine",
    "alias": "alpine",
    "keywords": "front"
  }
]
```

- **container_name**: Nombre del contenedor Docker que se desea monitorear. Este nombre debe coincidir exactamente con el nombre asignado al contenedor en el sistema.
- **alias**: Un nombre alternativo o descripción del contenedor. Se utiliza para facilitar la identificación y el filtrado del contenedor en los dashboards y reportes de monitoreo.
- **keywords**: Palabras clave asociadas al contenedor. Estas palabras permiten realizar búsquedas rápidas y filtrar contenedores según su función o categoría (por ejemplo, "database" o "front").

2.6.2.2 servicios.json

Este archivo contiene la configuración de los servicios web que deben ser monitorizados. Su estructura incluye información sobre la URL del servicio, su alias, palabras clave y otros metadatos como el servidor y el proyecto al que pertenece.

```
[
    {
        "url": "https://httpstat.us/500",
        "alias": "prueba/500",
        "keywords": "frontend",
        "servidor": "servidor_1",
        "proyecto": "proyecto_A",
        "maquina_virtual": "VM_1",
        "administrador": "admin_1"
    },
    {
        "url": "https://httpstat.us/403",
        "alias": "prueba/403",
        "keywords": "api",
        "servidor": "servidor_2",
        "proyecto": "proyecto_B",
        "maquina_virtual": "VM_2",
        "administrador": "admin_2"
    },
```

```
    {
        "url": "http://localhost:27017/",
        "alias": "mongodb",
        "keywords": "mongodb",
        "servidor": "servidor_1",
        "proyecto": "proyecto_A",
        "maquina_virtual": "VM_3",
        "administrador": "admin_3"
    }
]
```

- **url**: Dirección URL del servicio web que se desea monitorear. El script utilizará esta URL para verificar la disponibilidad y el estado del servicio.
- **alias**: Un nombre alternativo o descripción del servicio. Este alias se utiliza para facilitar la identificación del servicio en los reportes y dashboards.
- **keywords**: Palabras clave asociadas al servicio. Permiten realizar búsquedas rápidas y filtrar servicios según su categoría o función (por ejemplo, "frontend" o "api").
- **servidor**: Identificador del servidor físico o virtual donde está alojado el servicio. Este campo es útil para relacionar servicios con su infraestructura subyacente.
- **proyecto**: Nombre del proyecto al que pertenece el servicio. Este campo permite organizar los servicios según los proyectos en los que participan.
- **maquina_virtual**: Nombre de la máquina virtual donde está corriendo el servicio. Es útil para identificar entornos virtualizados dentro de la infraestructura.
- **administrador**: Nombre o identificador del administrador responsable del servicio. Este campo facilita la asignación de tareas y el contacto con la persona encargada de resolver incidencias.

3 Resultados y Conclusiones

3.1 Resultados Obtenidos

El sistema desarrollado ha logrado cumplir con los objetivos planteados inicialmente, proporcionando un entorno integral de monitorización de infraestructura. Los resultados obtenidos se pueden dividir en varios aspectos clave.

El trabajo realizado también puede ser comprobado en el siguiente GitHub: https://github.com/luisGarciiaa/monitorizacion-grafana-influx-telegraf.git

3.1.1 Implementación Exitosa de la Arquitectura

Se logró integrar de manera eficiente las herramientas seleccionadas, **Telegraf**, **InfluxDB** y **Grafana**, creando un flujo de trabajo continuo que permite la recolección, almacenamiento y visualización de métricas.

- **Telegraf** fue configurado en los servidores para recolectar métricas relevantes de CPU, memoria, disco y estado de los contenedores Docker.
- **InfluxDB v2** se utilizó como base de datos temporal para almacenar las métricas recolectadas.
- **Grafana** permitió la visualización de los datos mediante dashboards personalizables, que ofrecen información detallada del estado de los servidores y servicios.

3.1.2 Dashboards Dinámicos y Personalizables

Los dashboards desarrollados permiten al usuario realizar un monitoreo visual de los distintos componentes del sistema:

- **Dashboard de Servidores**: Proporciona una visualización detallada de los recursos de los servidores, como CPU, memoria y disco, con filtros dinámicos por host y bucket.
- **Dashboard de Servicios**: Permite supervisar el estado de los servicios web mediante la configuración de URLs monitorizadas.
- **Dashboard de Docker**: Facilita el seguimiento de los contenedores Docker sin necesidad de instalar Python en cada servidor.

Cada uno de estos dashboards está diseñado para soportar alertas y notificaciones automáticas, lo que mejora significativamente la capacidad del equipo técnico para identificar y responder rápidamente a cualquier anomalía o incidente.
Actualmente, el sistema está siendo desplegado en los servidores del grupo de investigación **OEG**, con una versión inicial en la URL: https://monitor.linkeddata.es.
Además, en la **sección 2.5** se presentan imágenes ilustrativas de los dashboards configurados, lo que permite al lector obtener una visión más clara de la interfaz y las funcionalidades del sistema de monitorización.

3.1.3 Configuración de Alertas y Notificaciones

Se configuraron alertas en Grafana para monitorear el uso de recursos y el estado de los servicios críticos del sistema. Estas alertas fueron diseñadas con el objetivo de garantizar una supervisión proactiva, permitiendo detectar y notificar al equipo de operaciones en tiempo real cuando una métrica crítica supera los umbrales definidos.

Las alertas se establecieron para cubrir distintos escenarios, tales como el uso excesivo de CPU, memoria y disco, así como el estado de los servicios web y contenedores Docker. Al configurarse, cada alerta incluye un umbral predefinido que, al ser superado, desencadena una notificación automática que llega al equipo técnico a través de los canales configurados.

Canales de notificación: Se configuraron notificaciones a través de **correo electrónico**, permitiendo que el equipo reciba avisos detallados en sus bandejas de entrada.

En el punto 2.4, se muestran ejemplos concretos de los avisos de alertas enviados por correo electrónico, donde se detalla el formato de los mensajes y la información incluida, como el tipo de alerta, el servidor o servicio afectado, el umbral superado y el estado del sistema. Estas notificaciones no solo alertan al equipo cuando se detecta una condición crítica, sino que también envían un mensaje de recuperación una vez que el sistema vuelve a la normalidad, proporcionando así un seguimiento completo del estado del sistema.

La correcta configuración de estas alertas ha sido fundamental para mejorar la capacidad de respuesta ante incidencias, permitiendo al equipo técnico actuar de manera rápida y eficiente, minimizando así posibles impactos en el rendimiento del sistema y asegurando la continuidad operativa de los servicios monitorizados.

3.1.4 Automatización de la Monitorización

Se logró una automatización eficiente mediante la configuración de Telegraf como servicio en cada servidor monitorizado, permitiendo una recolección continua de métricas sin intervención manual.

3.2 Conclusiones

El desarrollo de este proyecto ha demostrado ser una solución eficaz para la monitorización de infraestructuras tecnológicas, cumpliendo con los objetivos propuestos:

1. **Cumplimiento de los objetivos**: El sistema permite la recolección, almacenamiento y visualización de métricas de manera automatizada y centralizada, ofreciendo un entorno fácil de usar y escalable.
2. **Aporte al grupo de investigación**: Este sistema contribuye a mejorar la capacidad del grupo para gestionar y supervisar su infraestructura, permitiendo una detección temprana de problemas y una toma de decisiones más informada.
3. **Escalabilidad**: La arquitectura del sistema permite agregar nuevos servidores y servicios de manera sencilla, gracias a la configuración dinámica de Telegraf y los dashboards de Grafana.
4. **Aprendizaje obtenido**: Este proyecto ha permitido al estudiante adquirir conocimientos prácticos en el uso de herramientas de monitorización, configuración de sistemas y análisis de datos.

3.3 Trabajos Futuros

Existen varias líneas de trabajo que podrían mejorar o ampliar el sistema desarrollado:

1. **Nuevas métricas y dashboards**: Incorporar métricas adicionales, como latencia de red, consumo de energía o logs de aplicaciones.
2. **Integración con nuevas herramientas**: Explorar la posibilidad de integrar el sistema con otras herramientas de BI (Business Intelligence) para un análisis más avanzado.
3. **Mejoras en la configuración de alertas**: Implementar un sistema de aprendizaje automático que ajuste dinámicamente los umbrales de alerta según el comportamiento histórico de las métricas.
4. **Documentación y tutoriales**: Crear una guía detallada y tutoriales para facilitar la adopción del sistema por parte de otros grupos de investigación o equipos de operaciones.

En resumen, el proyecto desarrollado no solo cumple con los requisitos iniciales, sino que también establece una base sólida para futuras mejoras y extensiones, contribuyendo al avance en la gestión y monitorización de infraestructuras tecnológicas.

4 Análisis de Impacto

4.1 Impacto Personal

La realización de este Trabajo Fin de Grado (TFG) ha supuesto un impacto significativo a nivel personal. El desarrollo del sistema de monitorización y visualización de datos ha permitido:

- **Ampliación de conocimientos**: Durante el desarrollo del TFG se adquirieron habilidades técnicas en herramientas como Telegraf, InfluxDB y Grafana, así como conocimientos sobre integración de datos y creación de dashboards avanzados.
- **Resolución de problemas**: Los retos encontrados en la implementación del sistema reforzaron las habilidades de análisis y resolución de problemas.
- **Crecimiento profesional**: Este proyecto consolida la capacidad de trabajar en entornos reales y ofrece una base para futuros proyectos en monitorización y análisis de datos.

4.2 Impacto Empresarial

El sistema desarrollado tiene un impacto potencial positivo en el ámbito empresarial debido a:

- **Mejoras en la gestión de recursos**: La monitorización en tiempo real permite identificar rápidamente problemas en servidores, servicios y contenedores Docker, optimizando el tiempo de respuesta.
- **Reducción de costes**: Al prevenir fallos críticos mediante alertas tempranas, se minimizan las interrupciones y se reducen los gastos asociados a tiempo de inactividad.
- **Escalabilidad**: La flexibilidad del sistema permite adaptarse a diferentes infraestructuras, facilitando su adopción en diversas empresas y sectores.

4.3 Impacto Social

A nivel social, este proyecto contribuye de las siguientes maneras:

- **Facilitación del trabajo colaborativo**: Los dashboards permiten que los equipos de operaciones compartan información en tiempo real, mejorando la colaboración y toma de decisiones.
- **Accesibilidad técnica**: El uso de herramientas open source como InfluxDB y Grafana permite que el sistema sea accesible para equipos con recursos limitados, fomentando su adopción en pequeñas empresas o instituciones educativas.
- **Transparencia y confianza**: Los dashboards permiten a los usuarios de los servicios visualizar el estado y el rendimiento de la infraestructura en tiempo real, así como el historial de disponibilidad

(uptime). Esta transparencia contribuye a generar confianza en el sistema y en el equipo encargado de su mantenimiento.

4.4 Impacto Económico

El impacto económico del sistema se refleja en:

- **Eficiencia operativa**: La automatización de la monitorización reduce la carga de trabajo manual, permitiendo que el personal técnico se enfoque en tareas más estratégicas.
- **Ahorro en licencias**: Al tratarse de herramientas gratuitas y de código abierto, el sistema no incurre en costes asociados a licencias de software propietario.
- **Prevención de pérdidas**: La identificación temprana de problemas ayuda a evitar pérdidas financieras debidas a interrupciones no planificadas.

4.5 Impacto Medioambiental

El impacto medioambiental del sistema es positivo en los siguientes aspectos:

- **Eficiencia energética**: La monitorización permite identificar y optimizar el uso de recursos de hardware, reduciendo el consumo energético de los servidores.
- **Reducción de desperdicios**: Al prolongar la vida útil de los equipos mediante un mantenimiento proactivo, se disminuye la generación de residuos electrónicos.
- **ODS relacionados**: Este proyecto contribuye al Objetivo de Desarrollo Sostenible 12 (Producción y consumo responsables) al fomentar un uso eficiente de los recursos tecnológicos.

4.6 Impacto Cultural

Aunque el impacto cultural es más limitado, el sistema desarrollado:

- **Promueve la cultura de datos**: Al facilitar la visualización y análisis de datos en tiempo real, se fomenta una mentalidad basada en la información para la toma de decisiones.
- **Acceso a tecnologías avanzadas**: Este proyecto democratiza el acceso a herramientas de monitorización modernas, permitiendo que más personas y organizaciones las utilicen y comprendan.

Además, la transparencia facilitada por los dashboards fomenta una cultura de apertura, permitiendo que usuarios y administradores tengan una visión compartida del estado del sistema.

4.7 Impacto en los Objetivos de Desarrollo Sostenible (ODS)

El sistema desarrollado tiene el potencial de contribuir a varios ODS de la Agenda 2030, entre ellos:

- **ODS 9: Industria, innovación e infraestructura**: Promueve el uso de tecnologías innovadoras para mejorar la eficiencia operativa en infraestructuras tecnológicas.
- **ODS 12: Producción y consumo responsables**: Facilita la optimización de recursos energéticos y materiales.
- **ODS 13: Acción por el clima**: Ayuda a reducir el consumo energético y, por ende, la huella de carbono de las operaciones tecnológicas.

4.8 Decisiones basadas en el impacto

Durante la realización del TFG se tomaron varias decisiones que consideraron el impacto:

- **Uso de herramientas open source**: Se seleccionaron herramientas como Telegraf, InfluxDB y Grafana para garantizar accesibilidad económica y técnica.
- **Implementación de alertas proactivas**: Las alertas tempranas permiten reducir el tiempo de respuesta ante incidentes, minimizando interrupciones y su impacto asociado.
- **Optimización de recursos**: El sistema fue diseñado para consumir un mínimo de recursos y escalar según las necesidades de la infraestructura.

En resumen, este capítulo destaca los beneficios esperados del sistema desarrollado, así como las decisiones orientadas a maximizar su impacto positivo en diferentes contextos. Se subraya la contribución a los Objetivos de Desarrollo Sostenible y el compromiso con la eficiencia y la accesibilidad. Si se requieren mejoras adicionales, el sistema cuenta con una base sólida para su extensión y escalabilidad.

5 Bibliografía

[1] InfluxData, "Telegraf Documentation," [Online]. Available: https://docs.influxdata.com/telegraf . Accessed: Nov. 6, 2024.

[2] InfluxData, "InfluxDB Documentation," [Online]. Available: https://docs.influxdata.com/influxdb . Accessed: Nov. 6, 2024.

[3] Grafana Labs, "Grafana Documentation," [Online]. Available: https://grafana.com/docs/grafana/latest . Accessed: Nov. 6, 2024.

[4] Grafana Labs, "Getting Started with Grafana Alerts," [Online]. Available: https://grafana.com/docs/grafana/latest/alerting . Accessed: Nov. 6, 2024.

[5] Docker, "Get Started with Docker," [Online]. Available: https://docs.docker.com/get-started . Accessed: Nov. 6, 2024.

[6] The Linux Foundation, "Introduction to Virtualization," [Online]. Available: https://www.linuxfoundation.org/ . Accessed: Dic. 1, 2024.

[7] DigitalOcean, "How to Use Docker Compose," [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-use-docker-compose . Accessed: Dic. 1, 2024.

[8] InfluxData, "Flux Language Reference," [Online]. Available: https://docs.influxdata.com/flux/v0.x/ . Accessed: Dic. 1, 2024.

[9] Grafana Labs, "Using Variables in Grafana," [Online]. Available: https://grafana.com/docs/grafana/latest/variables/ . Accessed: Dic. 1, 2024.

6 Anexo

monitor_servicio.py:

```python
import requests
import json

# Ruta absoluta al archivo de configuración
config_file = "ruta/absoluta/servicios.json"

try:
    # Carga la configuración
    with open(config_file, "r") as f:
        services = json.load(f)

    # Itera sobre cada servicio y realiza la monitorización
    for service in services:
        url = service["url"]
        alias = service["alias"]
        keywords = service["keywords"]
        servidor = service.get("servidor", "default_server")
        proyecto = service.get("proyecto", "default_project")
        maquina_virtual = service.get("maquina_virtual", "default_vm")
        administrador = service.get("administrador", "default_admin")
        try:
            response = requests.get(url, timeout=2)
            if response.status_code == 200:
                print(f"http_response,url={url},source=url,nombre={alias},keywords={keywords},
                servidor={servidor},proyecto={proyecto},maquina_virtual={maquina_virtual},
                administrador={administrador} status_code=200,message=\"OK\"")
            else:
                error_message = response.text.replace('"', '\\"')
                print(f"http_response,url={url},source=url,nombre={alias},keywords={keywords},
                servidor={servidor},proyecto={proyecto},maquina_virtual={maquina_virtual},
                administrador={administrador} status_code={response.status_code},
                message=\"{error_message}\"")
        except requests.exceptions.Timeout:
            print(f"http_response,url={url},source=url,nombre={alias},keywords={keywords},
            servidor={servidor},proyecto={proyecto},maquina_virtual={maquina_virtual},
            administrador={administrador} status_code=408,message=\"Timeout\"")
        except requests.exceptions.RequestException as e:
            error_message = str(e).replace('"', '\\"')
            print(f"http_response,url={url},source=url,nombre={alias},keywords={keywords},
            servidor={servidor},proyecto={proyecto},maquina_virtual={maquina_virtual},
            administrador={administrador} status_code=500,message=\"{error_message}\"")
except Exception as e:
    print(f"Error al cargar el archivo de configuración: {e}")
```

monitor_docker.py:

```python
import docker
import json

# Ruta al archivo de configuración
config_file = "/ruta/absoluta/dockers.json"

try:
    # Carga la configuración
    with open(config_file, "r") as f:
        containers = json.load(f)

    # Conecta al cliente Docker
    client = docker.DockerClient(base_url='unix://var/run/docker.sock')

    # Itera sobre cada contenedor y realiza la monitorización
    for container_info in containers:
        container_name = container_info["container_name"]
        alias = container_info["alias"]
        keywords = container_info["keywords"]

        try:
            container = client.containers.get(container_name)
            container_status = container.status

            if container_status == "running":
                print(f"http_response,nombre={alias},source=docker,
                keywords={keywords} status_code=200,message=\"OK\"")
            else:
                print(f"http_response,nombre={alias},source=docker,
                keywords={keywords} status_code=500,message=\"{container_status}\"")
        except docker.errors.NotFound:
            print(f"http_response,nombre={alias},source=docker,
            keywords={keywords} status_code=404,message=\"Not Found\"")
        except docker.errors.DockerException as e:
            error_message = str(e).replace('"', '\\"')
            print(f"http_response,nombre={alias},source=docker,
            keywords={keywords} status_code=500,message=\"{error_message}\"")
except Exception as e:
    print(f"Error al cargar el archivo de configuración: {e}")
```

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jan 15 21:50:38 CET 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)