Sistema de Gestión de Salud y Bienestar con Monitorización Avanzada

MIEMBROS:

-Sergio Gonzalo Pajarero
-Diego Carmona Ruiz
-Borja Arenas Conde-Bandrés

Índice

---

2. Descripción Detallada del Diseño de la Solución

El Sistema de Gestión de Salud y Bienestar con Monitorización Avanzada se basa en una arquitectura de microservicios, con una combinación de tecnologías modernas para la gestión de productos, monitorización avanzada de la infraestructura y logging centralizado. La plataforma está diseñada para ayudar a gestionar productos relacionados con la salud y el bienestar, mientras se proporcionan herramientas de observabilidad para monitorear el rendimiento del sistema.

Componentes principales del sistema:

- **Frontend (Interfaz de usuario):** React con soporte de internacionalización, TailwindCSS para estilos y Shadcn/ui para componentes.
- **Backend:** Node.js con Express, utilizando PostgreSQL con Drizzle ORM para la base de datos. Las interacciones se gestionan mediante una API RESTful.
- **Monitorización:** Stack ELK (Elasticsearch, Logstash, Kibana) y Prometheus & Grafana para la recolección y visualización de métricas.
- **Infraestructura:** Docker y Docker Compose para contenerización de los servicios, asegurando un despliegue fácil y eficiente tanto en local como en entornos de producción como AWS.

Diagrama de Arquitectura

- **Microservicios Backend:** Gestión de productos, API RESTful.
- **Frontend:** Gestión de productos, interfaz de usuario intuitiva con operaciones CRUD.
- **Monitorización:** Prometheus para la recolección de métricas y Grafana para la visualización de métricas y alertas. ELK Stack para el procesamiento y visualización de logs.
- **Contenedores Docker:** Todos los servicios están contenerizados utilizando Docker, proporcionando una infraestructura escalable y flexible.

3. Justificación de las Decisiones Técnicas Tomadas

Elección de Tecnologías

1. **Frontend:**
    - **React:** Se eligió React por su flexibilidad, facilidad de integración con bibliotecas y amplia comunidad de soporte.
    - **TailwindCSS:** Para un desarrollo rápido de la interfaz de usuario con un diseño moderno, flexible y altamente personalizable.
    - **Shadcn/ui:** Para utilizar componentes UI listos para usar que cumplen con los requisitos de accesibilidad y facilidad de integración.
    - **React Query:** Para la gestión eficiente del estado de la aplicación y el manejo de datos asincrónicos.
2. **Backend:**
    - **Node.js y Express:** Son populares por su rendimiento y escalabilidad, ideales para aplicaciones que requieren rapidez en las operaciones de E/S como este sistema de gestión.
    - **PostgreSQL con Drizzle ORM:** PostgreSQL se eligió por su robustez y soporte para operaciones complejas. Drizzle ORM proporciona una capa de abstracción que facilita la interacción con la base de datos sin perder eficiencia.
3. **Monitorización:**
    - **Prometheus y Grafana:** Estas herramientas fueron elegidas por su popularidad en la monitorización de aplicaciones y sistemas, permitiendo recopilar métricas detalladas del rendimiento y visualizarlas de forma efectiva.
    - **ELK Stack (Elasticsearch, Logstash, Kibana):** Este stack es muy eficaz para el procesamiento de logs, permitiendo su centralización y posterior análisis en tiempo real.
4. **Infraestructura:**
    - **Docker y Docker Compose:** La contenerización garantiza que los servicios sean fácilmente escalables y portables. Docker Compose permite gestionar múltiples contenedores de manera sencilla.

4. Resultados de las Pruebas en Local y en AWS

Pruebas en Local

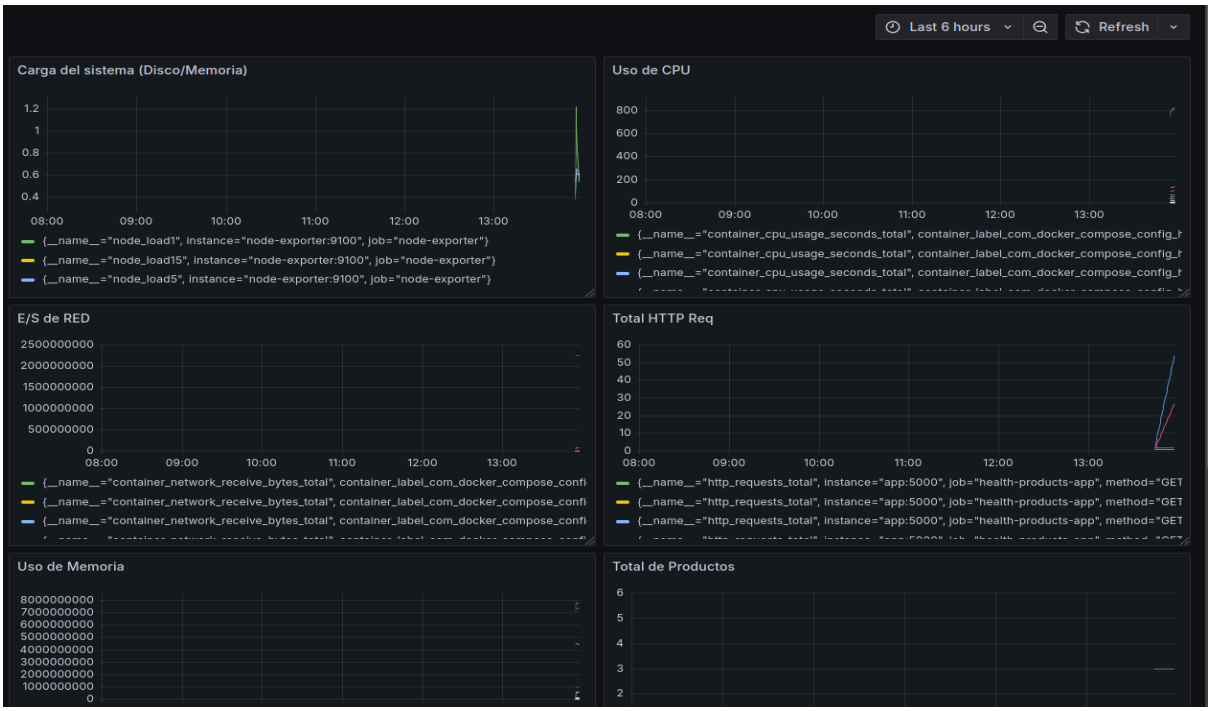
- El sistema se desplegó exitosamente en un entorno local utilizando Docker Compose. Los servicios de la aplicación, bases de datos, y herramientas de monitorización (Prometheus y Grafana) fueron accesibles a través de los puertos configurados en el archivo “docker-compose.yml”.
- Las métricas de los contenedores fueron recolectadas correctamente por Prometheus y visualizadas en Grafana, mientras que los logs fueron procesados por Logstash, almacenados en Elasticsearch y visualizados en Kibana.

Pruebas en AWS

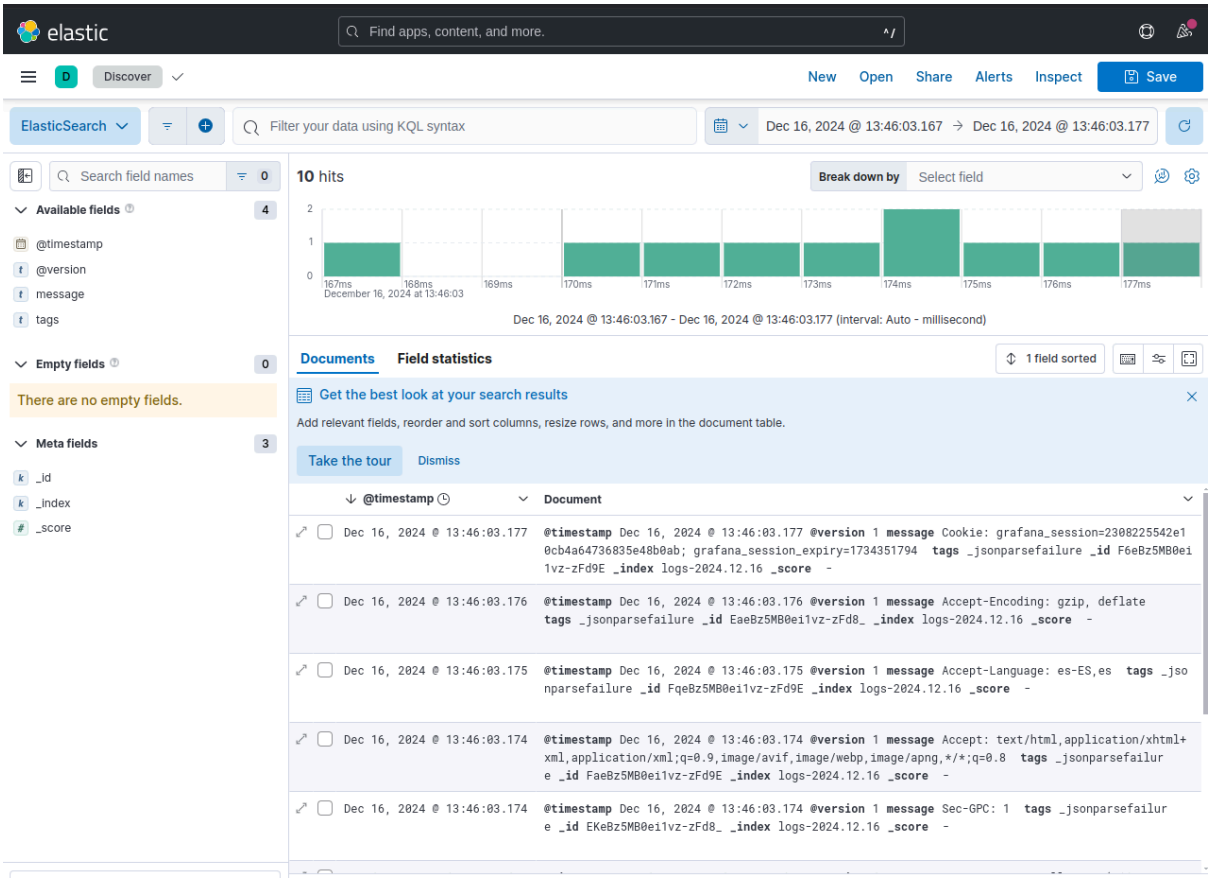
- El sistema fue desplegado en una instancia EC2 de AWS utilizando contenedores Docker. La infraestructura en la nube fue configurada con redes aisladas para los servicios de monitorización y logging.
- Prometheus y Grafana mostraron métricas precisas del rendimiento del sistema en AWS, con alertas configurables para notificar cualquier incidencia.
- La integración de los servicios de logging con ELK funcionó de manera eficiente, permitiendo una visualización clara de los logs en Kibana.

5. Capturas de Pantalla de la Monitorización en Producción

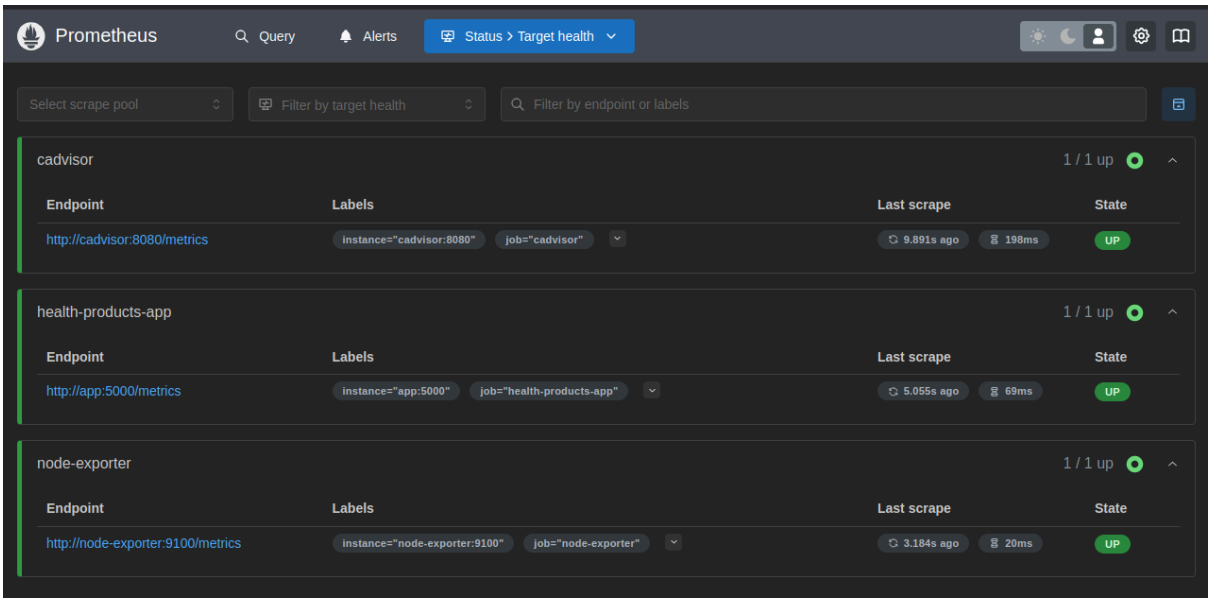
Grafana - Dashboard de Métricas de Rendimiento



Kibana - Visualización de Logs en Tiempo Real



Prometheus - Métricas del Sistema



6. Conclusiones Finales

Estas son las mayores complicaciones que nos encontramos realizando el proyecto:

1. Conexión a AWS RDS desde EC2:
    - Problema: Errores 500 al intentar crear productos
    - Solución: Estructuración correcta del archivo .env sin espacios adicionales ni comillas
2. Integración de logs con ELK Stack:
    - Problema: Errores en el envío de logs a Logstash
    - Solución: Implementación de sistema de logging vía UDP/TCP con formato JSON
3. Monitorización:
    - Problema: Necesidad de métricas en tiempo real
        - Solución: Implementación de endpoints específicos para Promet
        - heus (/metrics) y dashboards en Grafana