Recolección y filtrado de logs/registros mediante Fluent-bit con Docker

REALIZADO POR:

CELIA GARCIA MARQUEZ
IES GONZALO NAZARENO 2CFGS
ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED

INDICE

Portada....1
INTRODUCCIÓN....3
¿Que es Fluent-bit?....4
¿Por qué usar Fluent-bit?....4
¿Dónde nace Fluent-bit?....4
Requisitos del Sistema....5
Características. Diferencias y similitudes entre Fluent-bit y Fluentd....5
Conceptos clave de Fluent-bit....6
Características notables de Fluent-bit....7
Instalación de Fluent-bit....9
Configuración de Fluent-bit....10
Visualizar configuración de Fluent-bit....11
Métricas del sistema con Fluent-bit....12
PARTE PRÁCTICA....13
ESCENARIO EN LOCAL....13
ESCENARIO EN CLOUD....19
Conclusión....26
Bibliografía....26

INTRODUCCIÓN

El objetivo principal que vamos abordar en este proyecto es la **recolección y filtrado de registros** de los servicios que se estén ejecutando en múltiples contenedores en un entorno con Docker.

Para un **administrador de sistemas** es esencial conocer los **registros** de los servicios que está administrando, esto le permite tener un control sobre los posibles errores que se producen en una aplicación o servicio, además de encontrar vulnerabilidades en cuanto a seguridad nos referimos.

Cuando se trata de gestionar o supervisar registros generados por servicios a un nivel local o en remoto es una tarea no muy difícil de abordar ya que, a día de hoy se conocen muchas herramientas de monitoreo y gestión de logs, pero cuando trabajamos con microservicios alojados en contenedores esta tarea se vuelve mucho más compleja por lo que en este proyecto hablaremos de una herramienta que aporta solución a este problema.

La herramienta que vamos a tratar en cuestión se llama **Fluent-bit**. Fluent-bit es capaz de procesar, filtrar y reenviar los logs a múltiples destinos. Es de **código abierto** y es una opción muy adecuada para uso de contenedores tanto en **Docker** como **Kubernetes**. En este caso trabajaremos con Docker y Docker-Compose.

Trabajaremos en dos entornos:

- **De forma local:** Donde Fluent-bit utilizará un intermediario llamado Loki para comunicarse y visualizar los logs en Grafana.
- **En cloud:** De forma remota usando el servicio de nube proporcionado por AWS, donde Fluent-bit se comunica y visualiza los registros mediante el plugin CloudWacht de AWS.

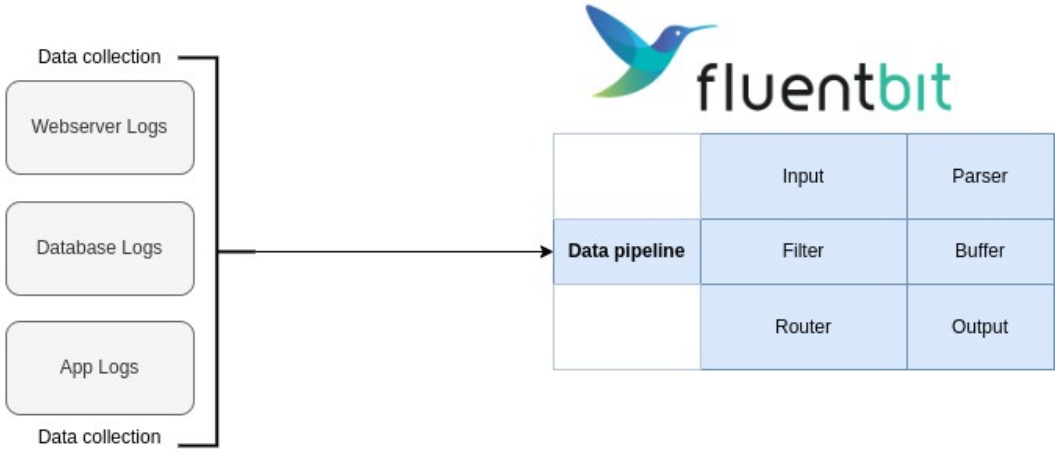
Ambos entornos necesitan configuraciones diferentes ya que estamos hablando de dos escenarios distintos, pero el objetivo siempre es el mismo: **Recolectar, filtrar y visualizar registros**. En ambos tenemos tres elementos en común:

- Un **servidor web**, en este caso Nginx
- Una **base de datos**, en este caso MySQL
- **Drupal** + **Php-Fpm**

Estos elementos serán la clave para generar logs y de ahí sacar partido a nuestras herramientas para procesarlos. Se han elegido estos tres elementos ya que es muy usual a día de hoy tener este escenario a nivel de empresa.

¿Que es Fluent-bit?

Fluent-bit es un procesador y reenviador de registros de código abierto que le permite recopilar datos, ya sean métricas o registros de diferentes fuentes. También es capaz de añadir filtros para una mejor gestión de los logs y enviarlos a múltiples destinos.



¿Por qué usar Fluent-bit?

Fluent-bit se propone como solución a unificar la gran cantidad de **datos** o **registros** que obtenemos de diferentes fuentes con **buen rendimiento** y a un **coste de recursos mínimo**, ya que, cada día recopilamos más datos (registros, métricas, eventos…) y cada vez es más complejo de gestionar toda la **información**. Esta herramienta es capaz de gestionar registros de **diferentes fuentes** de información, tratar **distintos formatos**, y reenviar esos datos a **destinos múltiples.** Como hemos indicado anteriormente, todo ello teniendo en cuenta un alto rendimiento y consumiendo muy pocos recursos de CPU y memoria.

¿Dónde nace Fluent-bit?

Fluent-bit nace en el año 2014, bajo el proyecto de **Fluentd** desarrollado por la empresa de software **Treasure Data.** Surgió con la necesidad de tener un procesador de registros más ligero para entornos con Linux embebido (servidores, teléfonos móviles, robots, microprocesadores…) y gateways o pasarelas, el proyecto formaba parte de Fluentd por lo que se llamo Fluent-bit.

Fluent-bit se convierte en una de las soluciones preferidas para resolver la gestión de registros en entornos de nube.

Requisitos del Sistema

Como hemos mencionado anteriormente, Fluent-bit genera un bajo consumo de CPU y memoria, es compatible con la mayoría de las arquitecturas. Y no necesita dependencias.

Características. Diferencias y similitudes entre Fluent-bit y Fluentd

Ambos son capaces de **recolectar**, **filtrar** y **reenviar** registros de fuentes diferentes a múltiples destinos con **formatos distinto**s. Unifican la capa de registro haciendo más fácil la gestión de los mismos. **Fluentd** es el proyecto padre que nace a finales de 2011 y **Fluent-bit** surge como subproyecto de éste en 2014. Aclarando esto pasamos a ver las similitudes:

- Cuentan con licencia **Apache v2.0**
- **Open Source.**
- Son proyectos alojados por la CNCF **(Cloud Native Computing Foundation)**
- Se utilizan a nivel de **producción**.
- **Ampliamente adoptado por la industria** (AWS, Microsoft, Google Cloud…)
- Creados por **Treasure Data.**
- Gran comunidad.

En la siguiente tabla podemos ver algunas similitudes y diferencias aclaratorias:

	FLUENTD	**FLUENT-BIT**
Alcance	Contenedores/ Servidores	Linux integrado/ Contenedores / Servidores
Lenguaje	**Escrito en C y Ruby**	**Escrito en C**
Rendimiento	Alto rendimiento	Alto rendimiento
Dependencias	**Al estar construido con Ruby necesita más dependencias**	**No necesita dependencias**
Complementos	**+1000**	**~=70**

¿Qué opción es más acertada?

Así como conclusión la diferencia más sobresaliente es que **Fluentd** cuenta con muchos **más complementos** así como funcionalidades pero conlleva un **aumento de recursos** debido a las **dependencias de Ruby**, mientras que, **Fluent-bit** soporta **menos complementos** pero no necesita dependencias y **el uso de recursos es mínimo**.

El uso de cada herramienta depende del propósito que se vaya a ejecutar. En mi caso para las tareas de este proyecto usaré **fluent-bit** ya que no necesitaré una gran cantidad de complementos y el consumo de los recursos es más bajo. Además lo usaré en un entorno de pruebas con **Docker**, en concreto con **docker-compose**. Pero si quisiéramos trabajar con **Kubernetes** la opción más adecuada sería utilizar Fluentd para aprovechar los complementos que dispone al máximo, incluso se podrían usar ambos ya que **se complementan**. De forma que en un entorno con Kubernetes, Fluent-bit actúa como transportista ligero que recoge datos de los diferentes nodos del clúster y los envía a fluentd para su agregación, procesamiento y enrutamiento a cualquier destino.

Si quieres saber algo más de Fluentd con Kubernetes te invito a leer esta entrada de blog:
Logging: Fluentd con Kubernetes

Conceptos clave de Fluent-bit

Voy a explicar brevemente algunos conceptos clave de Fluent-bit, así vamos a ver con más profundidad sus capacidades. Se tratan de conceptos muy básicos en relación a registros nos referimos.

Event or Record

Evento o **Registro**. Se trata de cada dato que recolecta, ya sea registro o métrica.
Cada línea se trata de un evento independiente, cada evento se compone principalmente de dos partes, el tiempo (TIMESTAMP) y la descripción (MESSAGE).

[TIMESTAMP, MESSAGE]

Filtering

Filtrado. Este concepto nos va ser muy útil ya que se trata de filtrar los eventos. Podemos hacer modificaciones en el contenido de los eventos, de forma que podemos **agregar información**, por ejemplo una ip asociada a ese registro, **seleccionar sólo una parte** del registro que nos interese o incluso **cribar los registros** según un patrón de búsqueda.

Tag

Cada evento que recopila Fluent-bit le asigna una **etiqueta**. Esta etiqueta es una cadena interna que el **enrutador** utiliza en una etapa posterior para decidir qué fase de filtro o salida debe atravesar. Estas etiquetas se indican en la configuración, si no se indica ninguna, fluent-bit por defecto le asigna el nombre de la instancia del complemento de entrada desde donde se genera el registro.

Timestamp

Marca de tiempo. Representa la hora en el que se creó el evento. Presente en la mayoría de los mensajes de registro.

Match

Relacionar o Coincidir, Fluent-bit es capaz de enviar los eventos a múltiples destinos, y esto lo hace a través de una fase de **enrutamiento,** del que hablaremos más adelante. Una coincidencia representa una regla para seleccionar aquellos registros que cumplen esa regla.

Structured Message

Mensaje estructurado. Fluent-bit convierte los mensajes en mensajes de clave-valor, siendo estos mensajes estructurados. Para ello utiliza MessagePack, que es un formato de datos de serialización binario, sería así como una versión binaria de JSON.

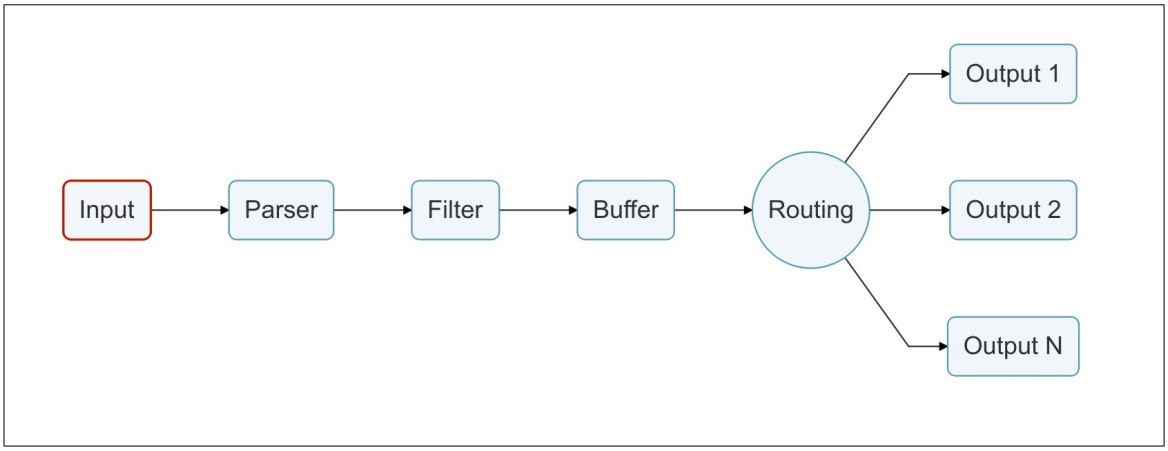
Características notables de Fluent-bit

Almacenamiento en Búffer. Redundante a fallos.

Se refiere a la capacidad de almacenar los registros en un lugar mientras a la misma vez se procesan y se entregan. Fluent-bit utiliza la **memoria del sistema** como un lugar primario y temporal para almacenar los registros antes de su entrega y es donde se procesan. De forma secundaria utiliza si lo necesita por cualquier fallo el **sistema de archivos** para almacenar los registros. Con esta **solución híbrida** da la seguridad que se necesita para un alto rendimiento.

Canalización de datos

Vamos a explicar de forma breve el proceso por el cuál fluent-bit recopila los datos, los procesa y los reenvía.



Input

Entrada de datos. Para la entrada de datos fluent utiliza varios mecanismos diferentes.
Cuando se recopila un dato, se crea una instancia interna. Cada instancia tiene su propia configuración. Las claves de configuración se denominan propiedades. Dependiendo del método de entrada que vayamos a recopilar, fluent aporta una documentación específica para cada método.

[Inputs](https://docs.fluentbit.io/manual/pipeline/inputs)

Parser

Se trata del analizador, es el que se encarga de estructurar los registros. Convierte el texto plano en un formato de clave-valor.
Según el método de entrada utilizará diferente parser, pero el más utlizado es el JSON.

[Parsers](https://docs.fluentbit.io/manual/pipeline/parsers)

Filter

El filtrado se implementa a través de unos complementos. Cada complemento provee de su propia documentación.

[Filters](https://docs.fluentbit.io/manual/pipeline/filters)

Buffer

Cuando entra en la fase de buffer, ya no se puede alterar el registro, no admite más filtros. Aquí ya está en estado de almacenamiento como hemos indicado anteriormente en '*Almacenamiento de búffer*', se almacenan en forma binaria en memoria.

Router

Se trata del enrutamiento, fluent-bit permite enrutar sus datos a través de filtros y pasarlo a uno o varios destinos. En la fase de enrutamiento tenemos dos elementos: tag y match, explicados anteriormente.

Los datos vienen con alguna etiqueta y para definir donde va estar la salida se debe especificar una regla o coincidencia (match).

Básicamente se trata de leer las etiquetas de entrada y las reglas de coincidencia para generar la salida. Si estos cumplen la coincidencia se dirige a su destino, si no se eliminan.

Se pueden usar comodines como '*' similar a una expresión regular para que sea más flexible la búsqueda.

Output

Interfaz de salida, permite definir los destinos a los que van dirigidos los datos. Pueden ser tanto como destinos de forma local en sistema de archivos o sistemas remotos como en cloud.

Las salidas también funcionan por complementos y cada uno tiene también su propia configuración. En este proyecto trataremos por ejemplo con las salidas para Amazon CloudWatch https://docs.fluentbit.io/manual/pipeline/outputs/cloudwatch

Instalación de Fluent-bit

Si consultamos la documentación oficial de fluent-bit, podemos comprobar que hay varios caminos para ello:

- Desplegando contenedores (Kubernetes, Docker y AWS)
- De forma integrada en Linux (Ubuntu, Debian, CentOS …)
- Para sistemas operativos con Windows (Server 2019 y Win10)
- Compilando desde el código fuente

(A tener en cuenta al instalar en Linux de forma local)
Si lo queremos instalar de forma local en Linux, por ejemplo con la distribución Ubuntu 20.04 LTS, además de importar la clave GPG y añadir la línea de repositorio de donde vamos a descargar el paquete, debemos tener en cuenta que el binario que vamos a descargar difiere ligeramente del paquete original 'fluent-bit', este se llama 'td-agent-bit'.

Td-agent-bit es fluent-bit pero para distribuciones específicas de Linux. Sin embargo se trata de las mismas funcionalidades, solo que ante su ejecución hay que tener en cuenta su nombre y el path en el que se encuentra.

Una vez aclarado esto, la opción que se ha elegido para este proyecto es el despliegue de contenedores Docker que brindan el servicio de fluent-bit bajo imágenes registradas y validadas en la web oficial de Docker Hub

Configuración de Fluent-bit

Fluent-bit utiliza un fichero de configuración principal llamado de manera habitual '`fluent-bit.conf`'. Este fichero se encuentra en diferentes rutas dependiendo de la forma y el sistema operativo en el que se haya instalado. En el caso de docker lo modificaremos mediante un volumen tipo bind mount.

La estructura del fichero está basado en formato clave-valor y espaciado mediante sangrías. Cada elemento está distinguido por **secciones.** Estas secciones pueden ser:

- **Service**. Define las propiedades del servicio de fluent-bit
- **Input**. Define la fuente de entrada (complemento)y parámetros según el complemento a utlizar.
- **Filter**. Define el nombre del filtro (complemento) y las reglas para el filtro.
- **Output**. Define la fuente de salida (complemento) y las reglas de salida.
- **Include file**. Es un método para incluir otros elementos en el servicio sin necesidad de que se encuentre en el principal y así ahorrar espacio. Normalmente en esos ficheros que se incluyen suelen ser los parsers.conf y plugins.conf

```
[SERVICE]
  #Tiempo de descarga de registros
    Flush           5
  #Ejecutar fluent-bit en segundo plano
    Daemon          off
  #Nivel de verbosidad de registro
    Log_Level       debug

[FILTER]
  #Nombre del complemento de filtro
    Name  stdout
  #Reglas para el filtro
    Match *

[INPUT]
  #Nombre del complemento de entrada
    Name cpu
  #Nombre de la etiqueta de entrada
    Tag  cpu.local

[OUTPUT]
  #Nombre del complemento de salida
    Name  stdout
  #Patrón que cumplen los datos de salida
    Match *
#Incluir ficheros
#@INCLUDE input_*.conf
@INCLUDE parsers.conf
```

Cabe destacar que también podemos usar **variables** en este fichero así:

- Exportando la variable en la terminal y usando `${MY_VARIABLE}`
- Con el comando @SET definiendo la variable en el fichero previamente `@SET my_input=cpu

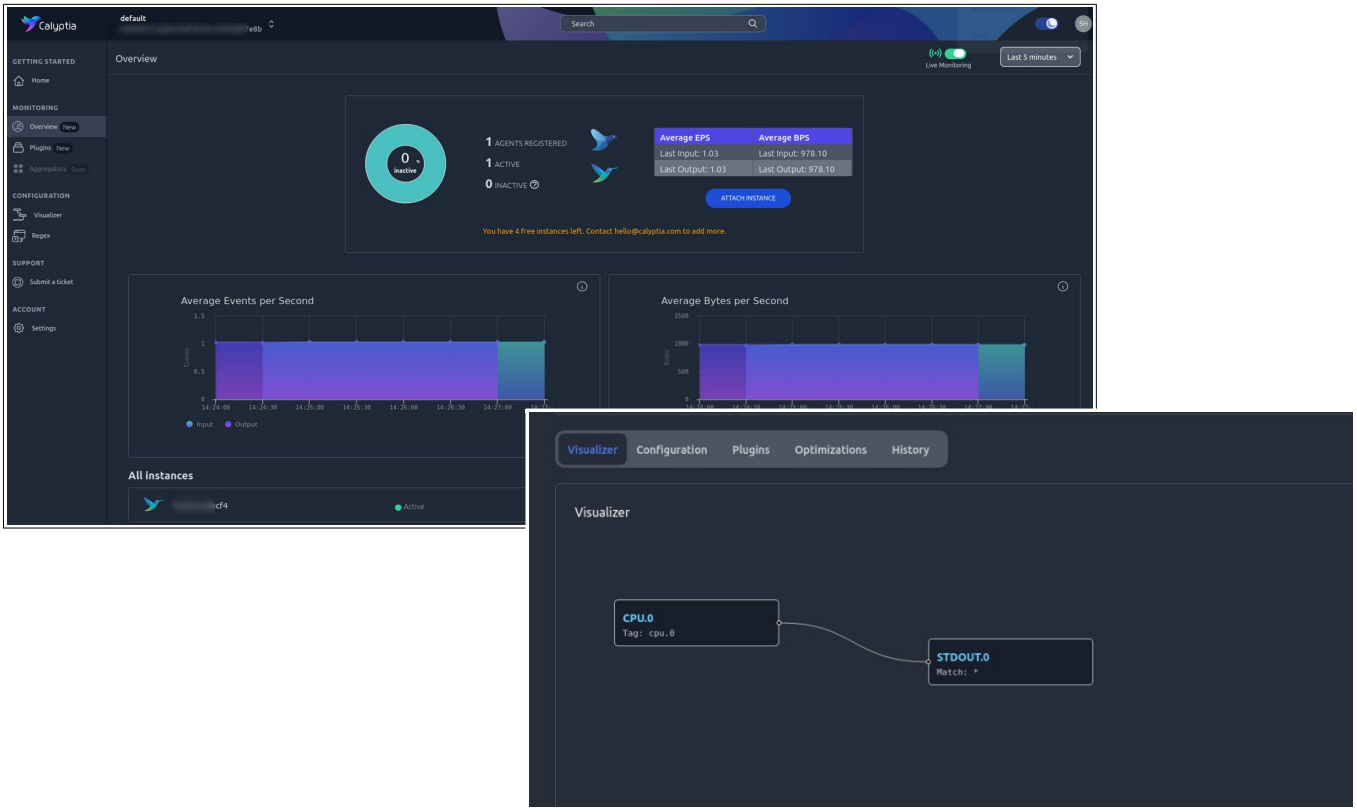
Visualizar configuración de Fluent-bit

Podemos visualizar la configuración de nuestro servicio de **Fluent-bit** de forma gráfica gracias a una plataforma creada también bajo el proyecto de Fluentd. Esta plataforma permite implementar, monitorear y administrar las canalizaciones de datos de aplicaciones de escala nativa de la nube.
Lo más interesante aquí es que podemos conectar nuestro servicio de Fluent-bit con calyptia cloud mediante un **agente** y **visualizar en vivo las canalizaciones.**

Simplemente se trata de registrarnos en Calyptia Cloud, Añadir un Agente y generar una Api Key para el fichero de configuración de nuestro servicio de Fluent-bit

```
[CUSTOM]
    name                calyptia
    api_key             apikeygenerada
    store_path          /tmp/storage
```

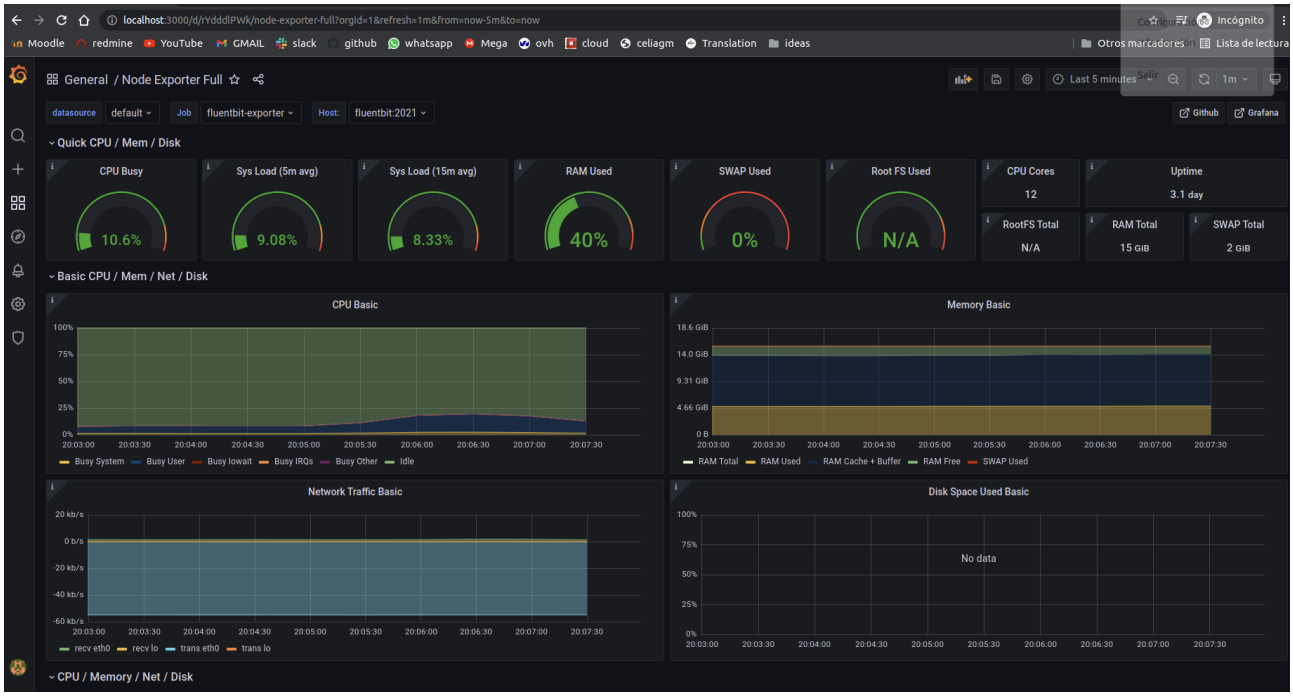
Como podemos ver en la siguiente imagen, en **Calyptia Cloud** tenemos registrado ya un agente que provee el servicio de Fluent-bit. Si accedemos al agente activo y a Visualizar, podemos ver el mapa de canalizaciones de datos.



Métricas del sistema con Fluent-bit

En este proyecto vamos a tratar de forma específica la gestión de los logs, pero es muy interesante mencionar la potencia que tiene también a la hora de recogida de **métricas**.

En el [repositorio oficial de Fluent-bit](), si accedemos al directorio de '*docker_compose*' podemos ver los ficheros necesarios para desplegar un escenario de prueba en el que conecta Fluent-bit con prometheus (prometheus exporter) y Grafana, que además incorpora ya un dashboard para Grafana en el que se pueden visualizar las métricas del sistema como vemos en la siguiente imagen.



PARTE PRÁCTICA

ESCENARIO EN LOCAL

El primer entorno en el que vamos a trabajar va ser un entorno en local. Como hemos comentado anteriormente vamos a usar Docker, y Docker-compose para levantar la infraestructura.

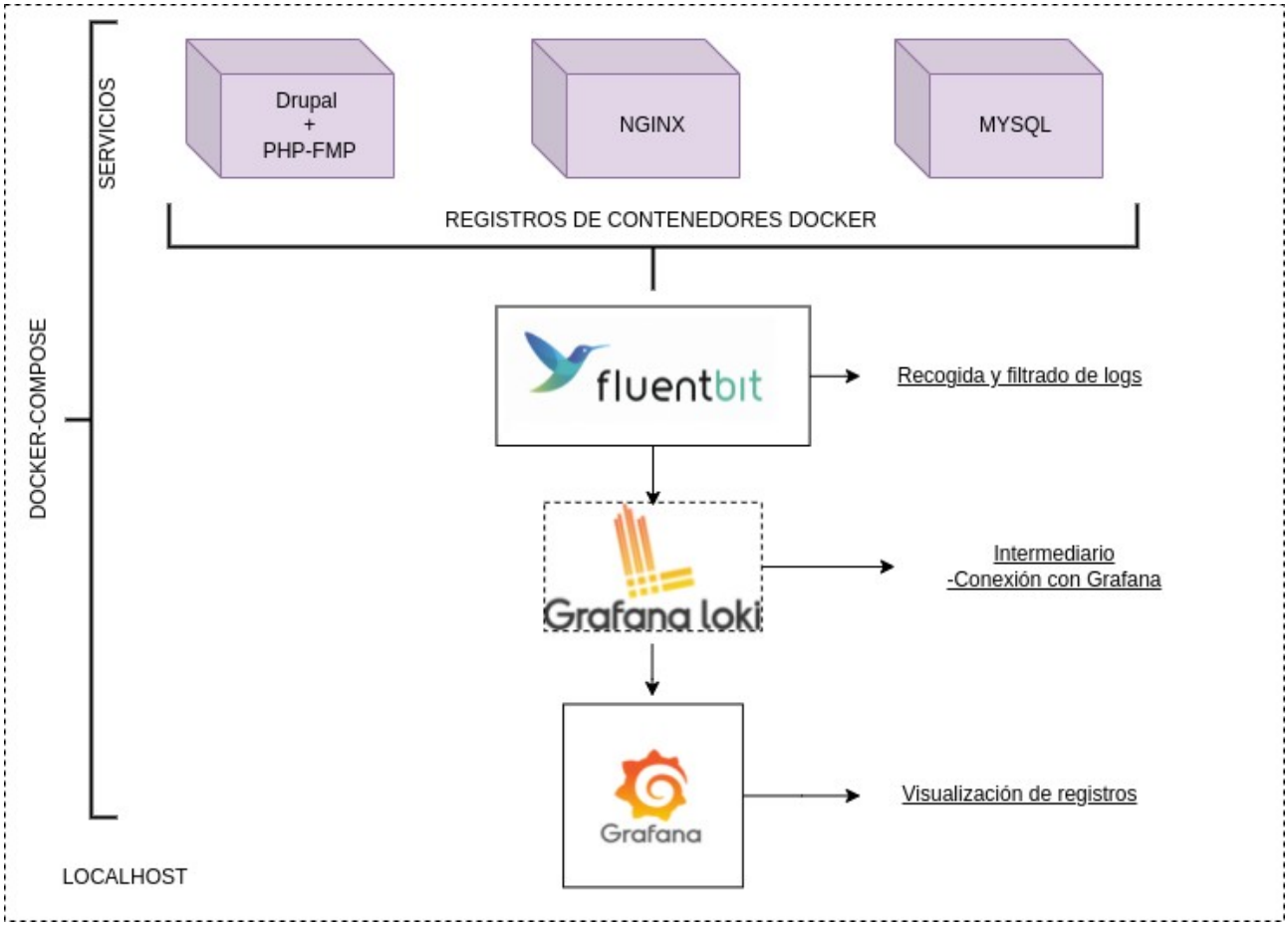
Puedes optar por acceder directamente al repositorio para una guía mas simple y breve de la tarea

Objetivo

El objetivo principal es recolectar los registros que generan los contenedores (servidor web, aplicación drupal y base de datos), dotar de un filtro a esos registros y reenviarlos mediante fluent-bit a un intermediario (plugin de Grafana) llamado Loki, que permite la visualización de los registros en Grafana, y visualizarlos mediante un Dashboard en el mismo Grafana.

Gráfico

En la siguiente imagen podemos observar el objetivo de forma gráfica:



Requisitos

- Sistema Operativo base: Ubuntu 20.04 Focal
- Paquetería actualizada
- Docker y Docker-compose instalados
- Git instalado

Clonar repositorio

Para trabajar con este proyecto vamos a clonar el siguiente repositorio ya preparado para desplegar nuestra infraestructura.

```
git clone https://github.com/CeliaGMqrz/fluent-bit_loki_grafana__docker-drupal
```

Crear el directorio de datos

Necesitamos crear una serie de directorios que serán los que almacenan los ficheros de la base de datos y drupal.

```
mkdir -p storage/files storage/mysql
```

Construir las imágenes

Necesitamos construir las imágenes para nuestro entorno, para ello ejecutamos un pequeño script simple que ejecuta esta acción por cada servicio.

```
bash build_images.sh
```

Cargar las variables de entorno

Es muy habitual que en el fichero **docker-compose.yaml**, como en este caso se incluyan **variables** para tener más flexibilidad a la hora de ejecutar despliegues. Entonces vamos a cargar las variables directamente en la terminal, copiando y pegando desde el fichero `.env`

Estas credenciales sólo son muestras de prueba. Se recomienda usar contraseñas seguras.

Docker-compose. Descripción de elementos.

Para levantar el entorno, vamos a utilizar **docker-compose**, que despliega una infraestructura a partir de un fichero llamado '**docker-compose.yaml**', en el cuál están definidos los contenedores que se van a ejecutar junto al servicio que le corresponde. Los servicios en cuestión son:

Servicios que generan registros (logs):

- Servidor Web: Nginx:1.19.6
- Base de datos: MySQL:5.7.5
- App Drupal+PHP-FPM:8.9.13

Pila de gestión de registros:

- Fluent-bit
- Loki
- Grafana

El fichero docker-compose que se encuentra en el repositorio está comentado con la descripción aclaratoria de cada uno de los elementos.

Configuración necesaria para el entorno

Esta configuración ya está dispuesta en el repositorio, estas explicaciones son simplemente informativas.

Fluent-bit

Para configurar correctamente el servicio de Fluent-bit contamos con dos partes. Cada enlace lleva al fichero indicado en el repositorio.

- Fichero ***'docker-compose.yaml'*** especificando las características del contenedor que va prestar el servicio.

```
#Fluent-bit, recolector y reenviador de registros.
fluentd:
  container_name: fluentd
  # Imagen específica de fluent-bit para Loki+Grafana
  image: grafana/fluent-bit-plugin-loki
  ports:
    - 24224:24224
    - 24224:24224/udp
    - 2020:2020
  environment:
    #Url necesaria para conectar con Loki
    LOKI_URL: http://loki:3100/loki/api/v1/push
  volumes:
    # Fichero de configuración principal de fluent-bit
    - ./fluent-bit/fluent-bit.conf:/fluent-bit/etc/fluent-bit.conf
    # Fichero de configuración para parsers
    - ./fluent-bit/parsers.conf:/fluent-bit/etc/parsers.conf
    # Directorio para ficheros proporcionados por calyptia cloud
    - ./fluent-bit/calyptia:/tmp/storage
  networks:
    - milocal
```

Se ha usado una imagen '*dockerizada*' de Fluent-bit especialmente para Loki. Los puertos expuestos son:

- 24224 – Utilizado por Fluent-bit para la recolección de logs de los servicios
- 2020 – Utilizado por Fluent-bit para ver las métricas vía HTTP

Para conectar los servicios (mysql, nginx y php) a cada contenedor definido en el fichero yaml se le añade la siguiente configuración, que indica a docker dónde enviar los logs de cada contenedor. Siendo Fluentd el driver a utilizar que proporciona Fluent-bit, e indicando la dirección y el puerto por el que Fluent-bit va recolectar los registros de cada servicio.

```
#Conectar el servicio a fluent-bit para recolectar logs
 logging:
   driver: fluentd
   options:
     fluentd-async-connect: "true"
     fluentd-address: localhost:24224
     tag: bd.logs
```

Complementos utilizados de Entrada/Salida

- **Elementos de entrada. Input.**

El complemento a utilizar para la entrada de datos en este caso sera '**forward**'. Forward es el protocolo utlizado por Fluent para **enrutar mensajes entre pares,** es decir, escucha los mensajes de reenvío. Simplemente se indica la **dirección** y el **puerto** de escucha para la recolección de logs. En este caso hemos indicado '0.0.0.0:24224' ya que es la dirección por la que estamos enviando los registros de los microservicios al contenedor de Fluent-Bit que los recopila.

- **Elementos de salida. Output.**

El complemento de salida a utilizar es '**grafana-loki**' . **Loki** es un sistema de agregación de registros multinquilino de alta disponibilidad, inspirado por **Prometheus**. No indexa el contenido de los registros, si no un conjunto de **etiquetas** para cada flujo de registros.

Este complemento permite recibir los registros recopilados por Fluent-bit y gestionarlo a base de etiquetas. Se le especifica la **Url** de Loki desde donde extrae los registros ya etiquetados por él mediante la Api. Añadimos la **etiqueta** que especifica de dónde va a extraer los logs, en este caso de Fluent-bit. Además **filtramos** los registros por nombre de contenedor indicando mediante el parámetro '**LabelKeys**' '**container_name**'. Especificamos el formato (**JSON**) y el nivel de verbosidad en este caso '**info**'.

Desplegar el entorno

Una vez explicada la configuración pasamos a desplegar el entorno con docker-compose.

Ejecutamos

```
docker-compose up -d
```

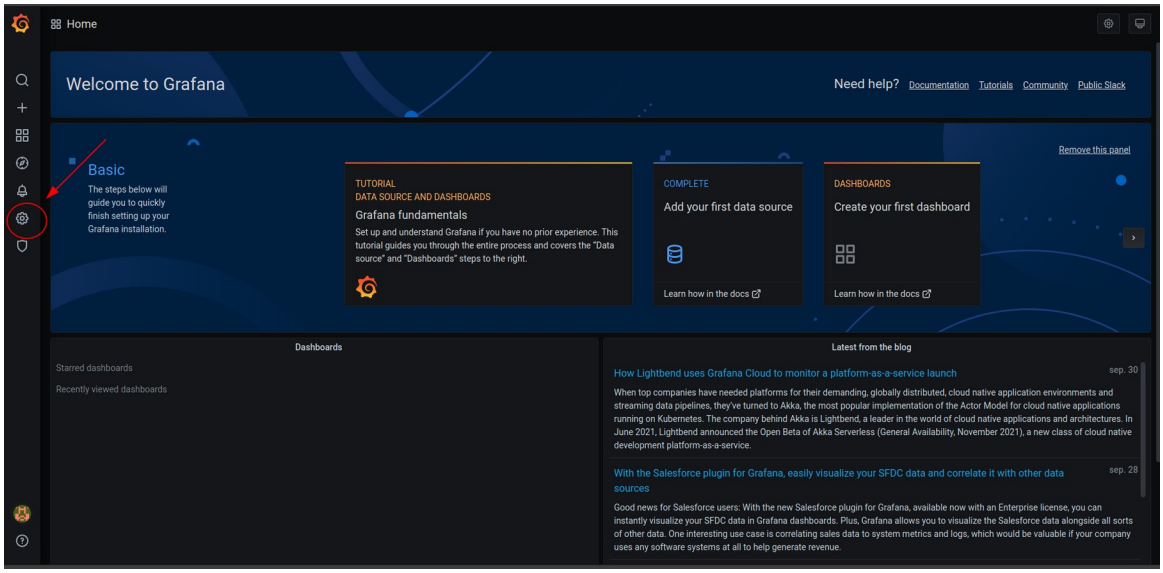
Y comprobamos que los contenedores están en funcionamiento

```
cgarcia@cgarcia-GL63-8SD:~/github/fluent-bit_loki_grafana__docker-drupal$ docker-compose ps
  Name                Command               State
 Ports
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
drupal_c    docker-php-entrypoint php- ...   Up      9000/tcp

fluentd     /fluent-bit/bin/fluent-bit ...   Up      0.0.0.0:2020->2020/tcp,:::2020->2020/tcp, 0.0.0.0:24224-
>24224/tcp,:::24224->24224/tcp, 0.0.0.0:24224->24224/udp,:::24224->24224/udp
grafana     /run.sh                          Up      0.0.0.0:3000->3000/tcp,:::3000->3000/tcp

loki        /usr/bin/loki -config.file ...   Up      0.0.0.0:3100->3100/tcp,:::3100->3100/tcp

mysql_c     docker-entrypoint.sh mysqld      Up      3306/tcp, 33060/tcp

nginx_c     /bin/bash /assets/bin/dock ...   Up      0.0.0.0:443->443/tcp,:::443->443/tcp,
0.0.0.0:80->80/tcp,:::80->80/tcp
```

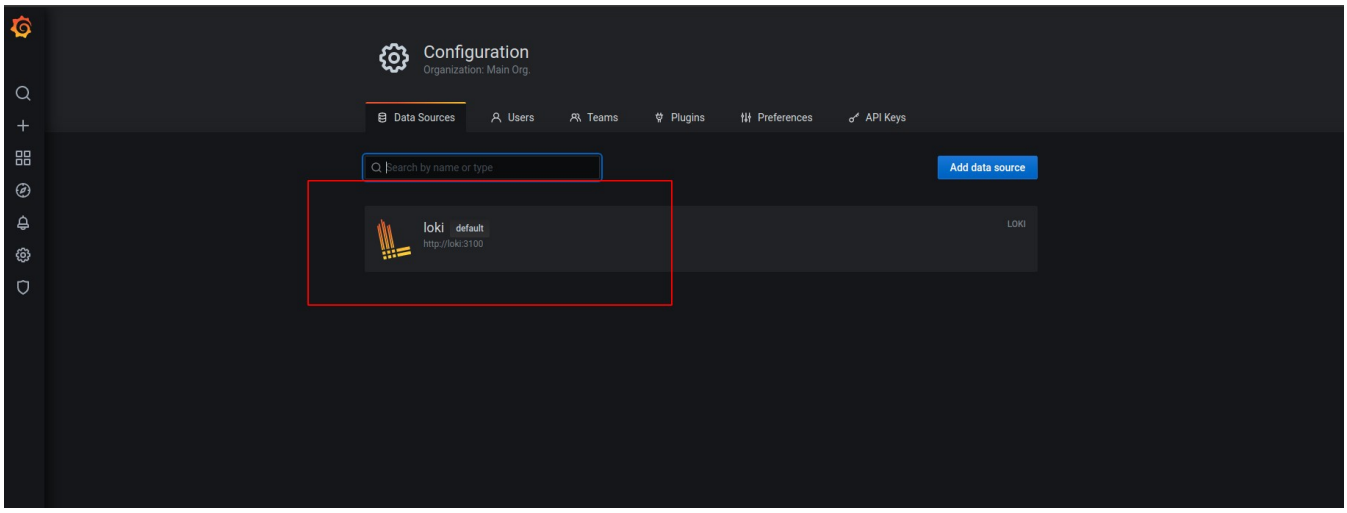
Visualizar los registros con Grafana

Entramos en el panel de Grafana → **http://localhost:3000**

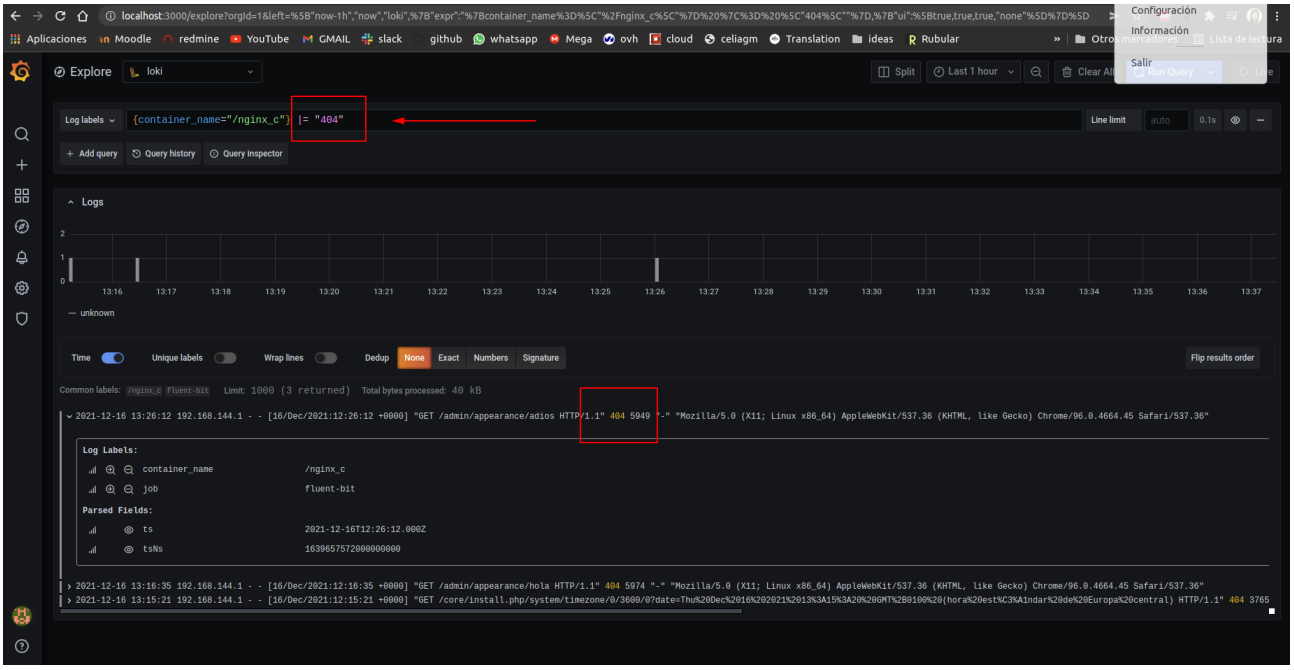
Indicamos las credenciales y podemos ver la página de bienvenida de Grafana. Accedemos a la C**onfiguración** >> **Data Sources**



Como podemos comprobar se ha añadido el **datasource** de loki, ya que lo hemos especificado gracias al fichero '***datasource.yml***' montado en volumen.



Ahora Podemos ir a **Explore/Explorar** y comprobamos que visualizamos todos los registros filtrado por el nombre de cada contenedor. Además, podemos hacer consultas más específicas añadiendo filtros según el [lenguaje **LogQL**] de Loki(Grafana). Como en el siguiente ejemplo filtrando por página no encontrada con error (**404**).



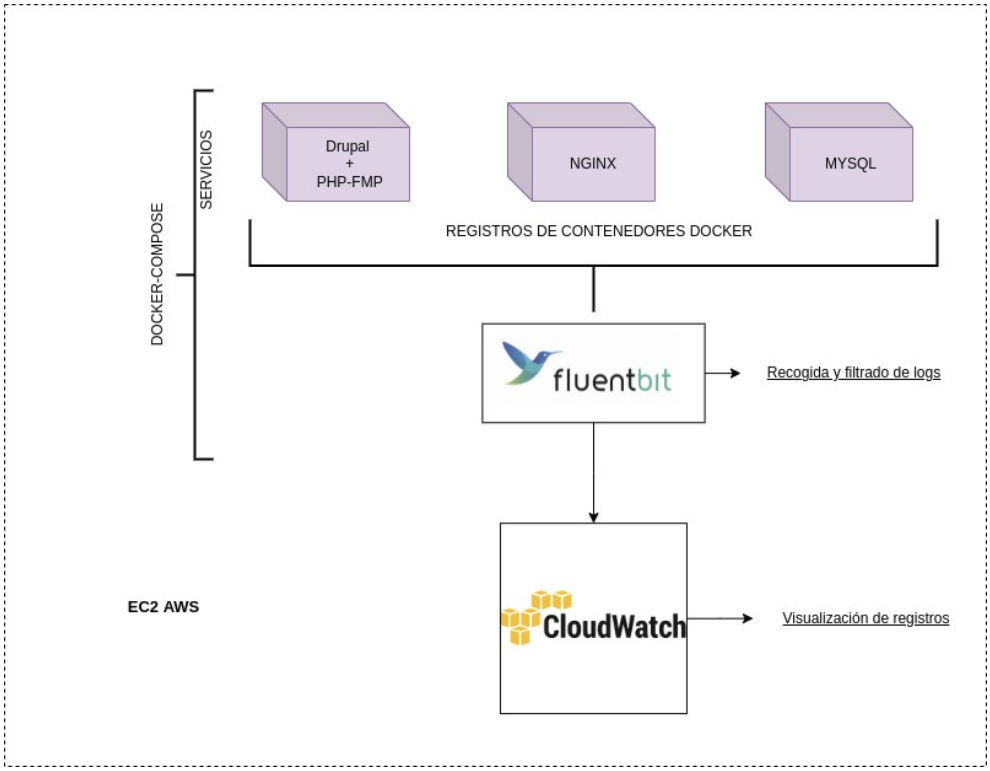
ESCENARIO EN CLOUD

El siguiente entorno en el que vamos a trabajar va ser un entorno en **remoto**. Como hemos comentado anteriormente vamos a usar Docker, y Docker-compose para levantar la infraestructura. Podríamos aplicar *Terraform* o *Ansible* para la automatización del despliegue en AWS pero se escapa del objetivo de este proyecto por lo que vamos a tratar de centrar la atención en la gestión de logs.

Objetivo

El objetivo principal es recolectar los registros que generan los contenedores (**servidor web, aplicación drupal y base de datos**), dotar de un filtro a esos registros y reenviarlos mediante Fluent-Bit a una máquina EC2 con un servicio de monitorización de **Amazon** llamado **CloudWatch**.

Gráfico
En la siguiente imagen podemos observar el objetivo de forma gráfica:



Requisitos

- Registrar una cuenta de AWS
- Una máquina Ec2 con Ubuntu/Debian (Paquetería actualizada, git, docker, docker-compose instalados)

Máquina Ec2

Accedemos a nuestra cuenta de AWS, si no tenemos lanzada la instancia con Ec2 la lanzamos, nos aseguramos que utilizamos una imagen con Ubuntu 20.04.

Actualizar paquetería y comprobar versiones.

Actualizamos el la paquetería instalamos los paquetes necesarios si no los tenemos instalados y comprobamos las versiones.

```
#Docker comopose https://docs.docker.com/compose/install/

ubuntu@ip-172-31-15-87:~$ docker-compose version
docker-compose version 1.29.2, build 5becea4c
docker-py version: 5.0.0
CPython version: 3.7.10
OpenSSL version: OpenSSL 1.1.0l  10 Sep 2019


#Docker Engine https://docs.docker.com/engine/install/ubuntu/

ubuntu@ip-172-31-15-87:~$ docker version
Client: Docker Engine - Community
 Version:           20.10.12
 API version:       1.41
 Go version:        go1.16.12
 Git commit:        e91ed57
 Built:             Mon Dec 13 11:45:33 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true
```

Clonar repositorio

Creamos un directorio de trabajo y clonamos el repositorio

```
git clone https://github.com/CeliaGMqrz/ec2_fluent-bit_entorno_drupal.git
```

Crear el directorio de datos

Necesitamos crear una serie de directorios que serán los que almacenan los ficheros de la base de datos y drupal.

```
mkdir -p storage/files storage/mysql
```

Construir las imágenes

Necesitamos construir las imágenes para nuestro entorno, para ello ejecutamos un pequeño script simple que ejecuta esta acción por cada servicio.

```
bash build_images.sh
```

Cargar las variables de entorno

Es muy habitual que en el fichero **docker-compose.yaml**, como en este caso se incluyan **variables** para tener más flexibilidad a la hora de ejecutar despliegues. Entonces vamos a cargar las variables directamente en la terminal, copiando y pegando desde el fichero `.env`

Estas credenciales sólo son muestras de prueba. Se recomienda usar contraseñas seguras.

Docker-compose. Descripción de elementos.

Para levantar el entorno, vamos a utilizar **docker-compose**, que despliega una infraestructura a partir de un fichero llamado '**docker-compose.yaml**', en el cuál están definidos los contenedores que se van a ejecutar junto al servicio que le corresponde. Los servicios en cuestión son:

Servicios que generan registros (logs):

- Servidor Web: Nginx:1.19.6
- Base de datos: MySQL:5.7.5
- App Drupal+PHP-FPM:8.9.13

Pila de gestión de registros:

- Fluent-bit
- CloudWatch

Configuración necesaria para el entorno

Fluent-bit

En el fichero docker-compose.yaml está expuesto de la siguiente forma:

```
  fluentd:
    container_name: fluentd
    image: amazon/aws-for-fluent-bit
    ports:
      - 24224:24224
      - 24224:24224/udp
    volumes:
      - ./fluent-bit/fluent-bit.conf:/fluent-bit/etc/fluent-bit.conf
    networks:
      - milocal
```

Al igual que el escenario en local, Fluent-bit tiene la misma funcionalidad, sin embargo en este caso vamos a usar una imagen específica de aws para conectar fluent-bit con AWS.

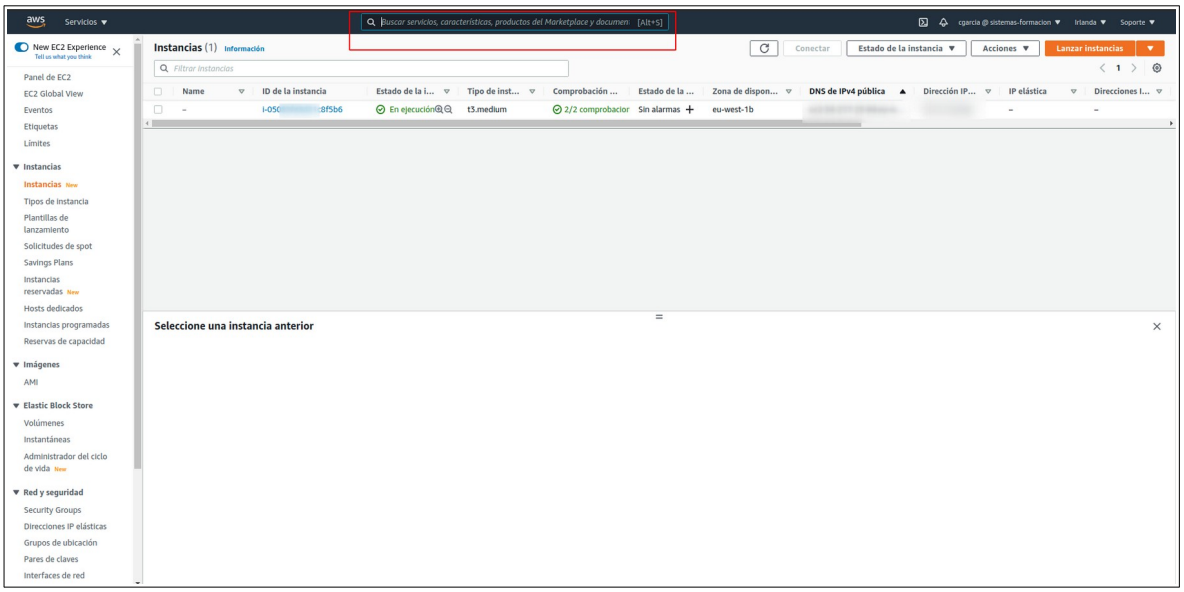
En el fichero de configuración principal de Fluent-bit el complemento de salida también será diferente.

```
. . .
. . .

[INPUT]
    Name              forward
    Listen            0.0.0.0
    Port              24224
    Buffer_Chunk_Size 1M
    Buffer_Max_Size   6M

[OUTPUT]
    Name cloudwatch_logs
    Match   *
    region eu-west-1
    log_group_name fluent-bit-cloudwatch
    log_stream_prefix fluent-bit-cloudwatch-celia-
    auto_create_group On
    workers 1
```

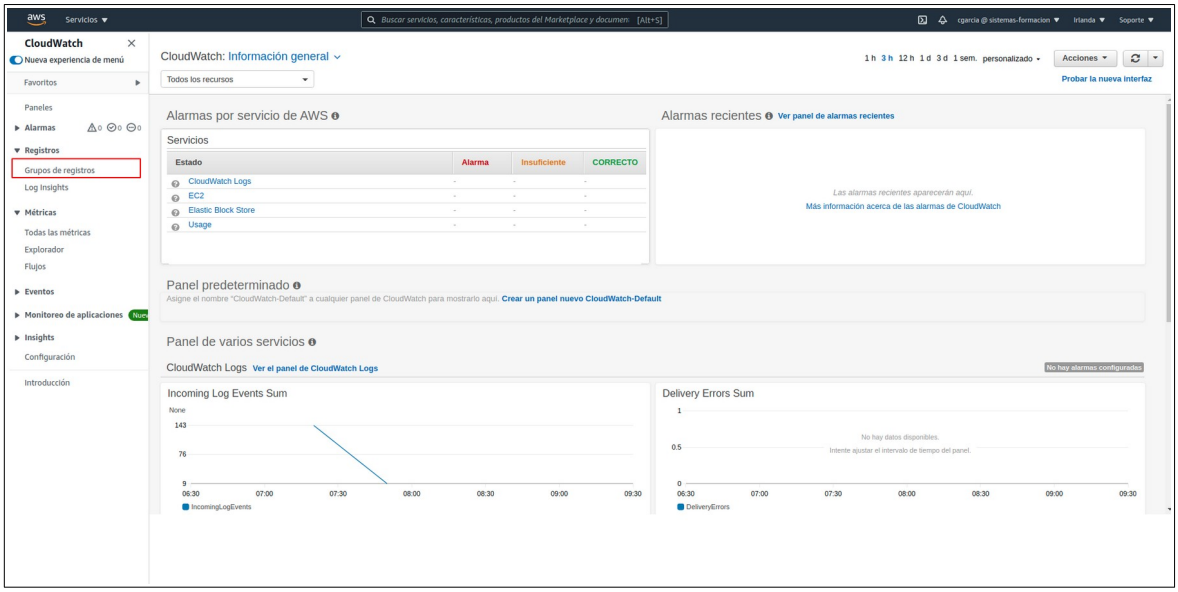
El nombre del complemento hace referencia a **CloudWatch**. Tenemos que indicar la región en la que nos encontramos. El nombre de grupo de registros y el prefijo que adoptará cada salida de registros.

Configuración CloudWatch

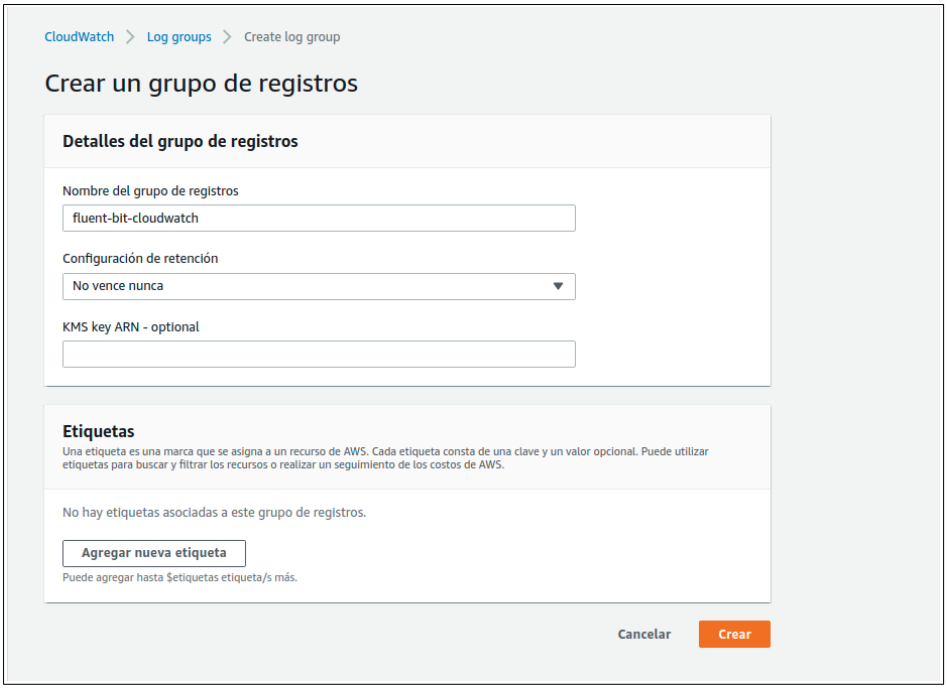
Ahora vamos a ir a la consola de AWS, y en el buscador introducimos CloudWatch



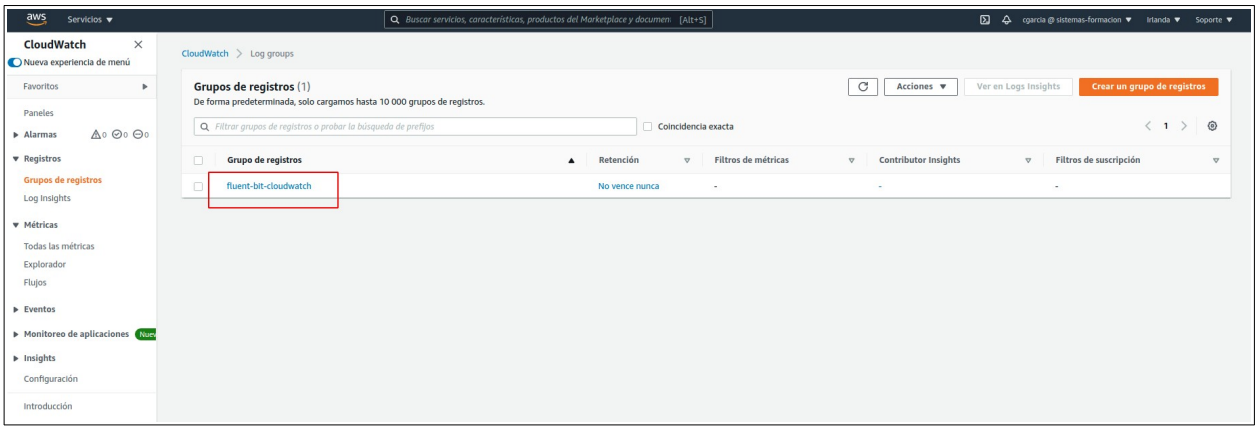
En el panel de la izquierda podemos ver **Grupos de registros** donde entramos para crear nuestro grupo de logs



Creamos el nuevo grupo con un nombre identificativo, le podemos pasar el tiempo de vencimiento. Además aquí podemos añadir etiquetas o tags de forma opcional.



Comprobamos que el panel está creado

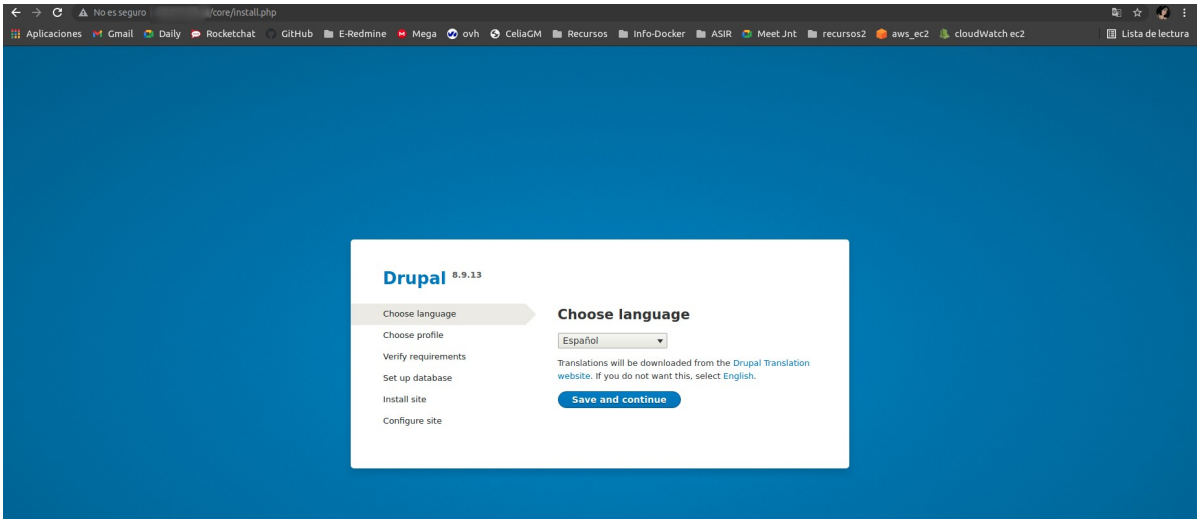


Desplegar el entornos

Tras la configuración vamos a proceder a desplegar el entorno, desde el directorio principal del repositorio y comprobamos que los contenedores están en funcionamiento y se han levantado todos

```
docker-compose up -d
docker-compose ps
```

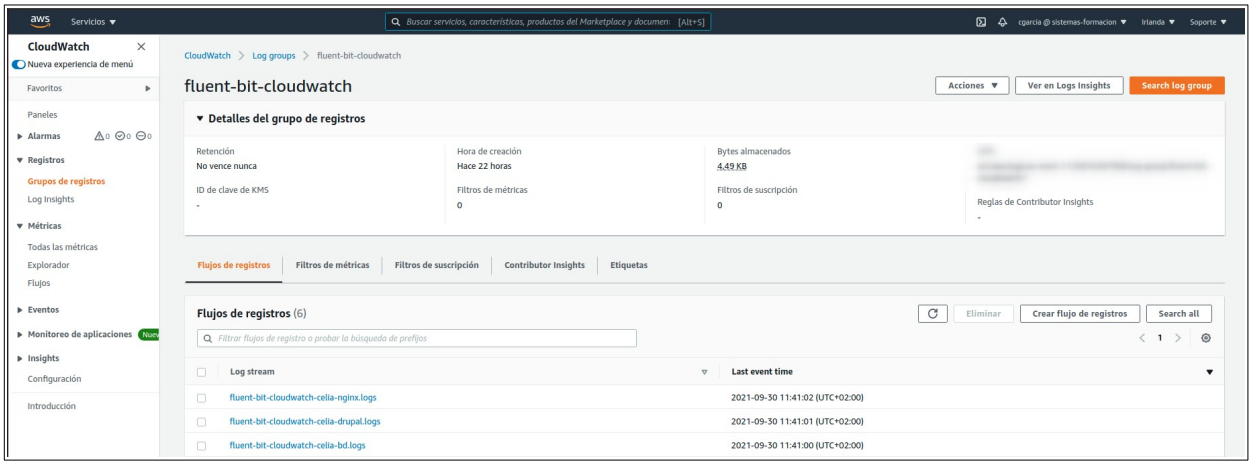
Podemos acceder a la ip pública de nuestra máquina Ec2 y comprobar que nos muestra el instalador de Drupal funcionando.



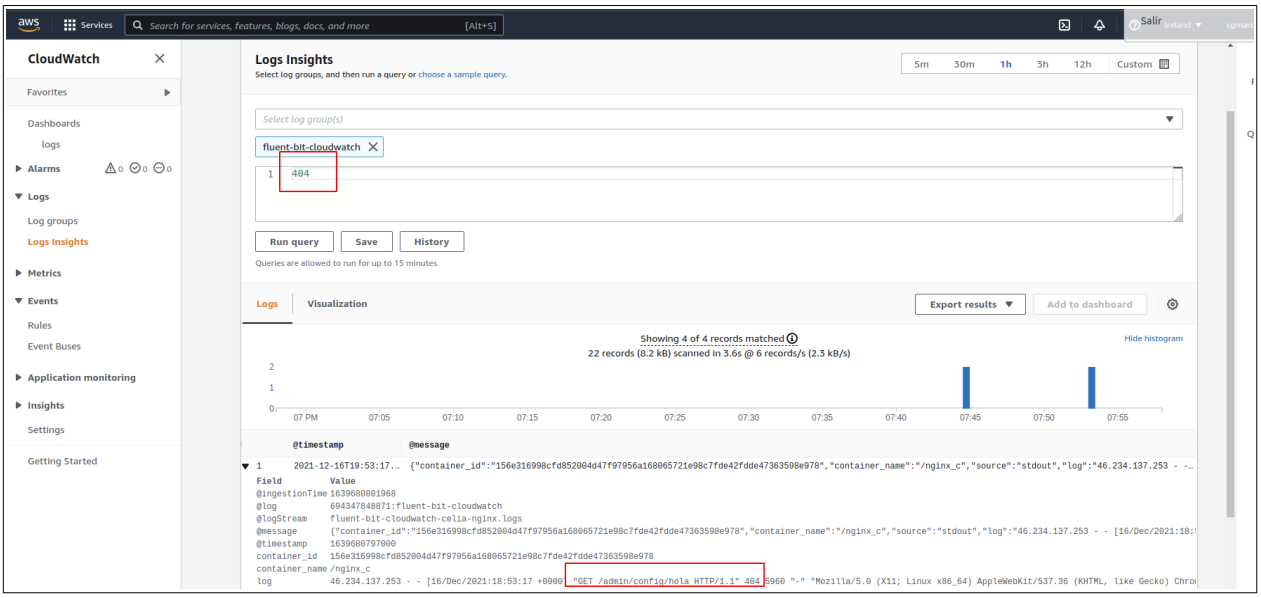
Muestra de registros en CloudWatch

Una vez realizado todo el proceso vamos a ir de nuevo al panel de AWS y buscar CloudWatch, entramos en los grupos de logs y comprobamos que se han creado los grupos de registros por cada contenedor especificado

- MySQL
- Nginx
- Drupal



Además si entramos en cada grupo podemos ver los registros en formato JSON y de igual forma podemos filtrar por cadena de texto, como por ejemplo buscar los errores 404 que aparecen en Nginx



Conclusión

Durante todo el documento hemos visto en funcionamiento Fluent-bit en diferentes entornos con el objetivo de recolectar, procesar y reenviar logs a varios destinos, incluso de filtrar esos registros por nombre de contenedor. Esta tarea Fluent-bit la hace ver menos compleja de lo que se trata, pues la gestión de registros cuando trabajamos a nivel de contenedores llega a ser extensa y difícil de manejar.

Con Fluent-Bit a un coste de mínimos recursos hemos podido comprobar que es rápido y eficiente y que además cuenta con numerosos complementos tanto de entrada, salida como de filtrado, siendo claramente una buena opción para gestionar registros cuando el entorno tiene un límite de recursos de CPU o RAM; Y por supuesto si necesitamos una gestión de registros más avanzada y contamos con más recursos podemos optar por Fluentd que cuenta con muchos más complementos y configuraciones.

Tanto Fluentd como Fluent-bit cada día están mas integrados en la comunidad y la cantidad de complementos crece. Como por ejemplo con Amazon.

Finalmente creo que Fluent-bit combinado con Fluentd puede ser una opción muy acertada para la gestión de logs en entornos con contenedores tanto como Kubernetes como con Docker.

Bibliografía

Fluent-bit docs - https://docs.fluentbit.io/manual/
Grafana-Loki docs - https://grafana.com/oss/loki/
Grafana - https://grafana.com/docs/
AWS-Fluent-bit- https://docs.aws.amazon.com/es_es/AmazonCloudWatch/latest/monitoring/Container-Insights-setup-logs-FluentBit.html
Repositorio oficial fluent-bit- https://github.com/fluent/fluent-bit
Docker logging - https://docs.docker.com/config/containers/logging/log_tags/

Otros enlaces:

https://brunopaz.dev/blog/better-docker-log-search-experience-with-loki/
https://medium.com/@elfakharany/how-to-use-the-fluent-bit-rewrite-tag-plugin-with-the-kubernetes-plugin-for-log-parsing-5934a85ce9e
http://www.inanzzz.com/index.php/post/rel5/using-fluent-bit-to-forward-docker-php-fpm-and-nginx-logs-to-elasticsearch