

14 DE JUNIO DE 2023



DOCKER

BALANCEO DE CARGAS Y MONITOREO DE LOGS EN NGINX

MANUEL LUNA MADRID

CICLO SUPERIOR DE ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN REDES
IES Medina Azahara



Esta obra está sujeta a una licencia de
Reconocimiento – No Comercial – Sin Obra Derivada
[3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

FICHA DEL PROYECTO FINAL

Título del trabajo:	Docker: Servidor Nginx con balanceo de cargas y control de registros
Nombre del autor:	Manuel Luna Madrid
Fecha de entrega (mm/aaaa):	06/2023
Área del Trabajo Final:	Seguridad, Sistemas Operativos y Redes
Ciclo Grado Superior:	Administración de Sistemas Informáticos en Red
Resumen del Trabajo:	
<p>El proyecto consiste en la implementación de un servidor Nginx como proxy inverso y balanceador de cargas, con dos servidores Apache y un sistema de monitoreo de registros en Docker. Esta solución proporciona una infraestructura escalable y eficiente para el despliegue de aplicaciones.</p> <p>El uso de Docker permite la creación de contenedores independientes y livianos que encapsulan los servidores Nginx y Apache, lo que facilita su implementación y gestión. El servidor Nginx actúa como un proxy inverso, redirigiendo las solicitudes a los servidores Apache y equilibrando la carga de trabajo de manera efectiva.</p> <p>Además, se implementa un sistema de monitoreo de registros en el servidor Nginx, lo que proporciona visibilidad sobre el tráfico, los tiempos de respuesta y posibles problemas. Esto facilita la detección temprana de problemas y la optimización continua del sistema.</p> <p>Se deben considerar las debilidades potenciales, como la complejidad en la gestión y los posibles problemas de rendimiento si no se realiza una configuración adecuada. Asimismo, las amenazas pueden surgir en términos de seguridad y competencia en el mercado.</p> <p>Entre las fortalezas del proyecto se encuentran la flexibilidad y escalabilidad proporcionadas por Docker, la optimización de recursos y la mejora de la disponibilidad a través del balanceo de cargas. Las oportunidades incluyen la escalabilidad del negocio y la posibilidad de explorar innovaciones tecnológicas.</p>	

Índice

1 - Introducción	1
2 - Análisis de contexto.....	1
3 - Justificación	2
4 - Objetivos.....	3
5 - Marco de referencia	3
6 - Enfoque y método seguido	3
7 - Costes del proyecto	4
Costes de implementación.....	4
8 - Temporización	5
Diagrama de Gantt	6
9 - Docker.....	7
Ventajas.....	7
Limitaciones	8
Elementos de Docker	8
Motor de Docker	8
Imágenes, contenedores y volúmenes	9
Imagen.....	9
Contenedor	9
Volumen	9
Docker Hub.....	9
Red Docker	10
Dockerfile	10
Docker Compose	11
Instalación de Docker en Windows.....	11
Instalación de Docker en Ubuntu.....	12
Comandos importantes.....	12
Portainer.io.....	13
10 - Ejemplo practico.....	17
Nginx.....	18
Apache.....	18
Loki	19
Grafana.....	20
Archivos de configuración	22
Nginx.conf	22
Apache html	23
Loki-config.yml	24
Plugin de loki	27
Docker-compose.yml	27
Instalación	29
Descargar las imágenes.....	29
Ejecutar Docker-compose.yml	29
Visualizar el servidor	30

Acceder a Grafana	31
Conexión de Grafana y Loki	33
Comprobación de logs	34
11 - Seguimiento del proyecto	35
Mantenimiento	35
Incidencias	35
12 - Conclusión	36
Debilidades	36
Amenazas	36
Fortalezas	37
Oportunidades	37
Posibles mejoras	37
Escalabilidad	38
13 - Bibliografía	39

Lista de imágenes

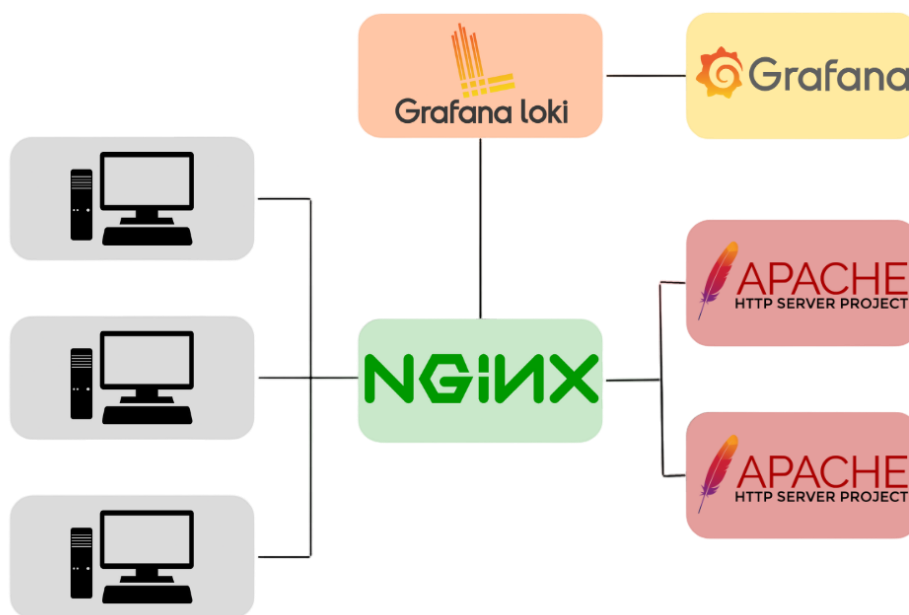
Ilustración 1 – Esquema servidor	1
Ilustración 2 – Diagrama de Gantt.....	6
Ilustración 3 – Docker vs Máquinas Virtuales	7
Ilustración 4 – Portainer.io Inicio	14
Ilustración 5 – Portainer.io Dashboard.....	14
Ilustración 6 – Portainer.io Detalles de contenedor	15
Ilustración 7 – Portainer.io Lista de imágenes	15
Ilustración 8 – Imagen Nginx Docker Hub	18
Ilustración 9 – Imagen Apache Docker Hub	19
Ilustración 10 – Imagen Loki Docker Hub.....	20
Ilustración 11 – Imagen Grafana Docker Hub	21
Ilustración 12 – Comando Docker-compose pull	29
Ilustración 13 – Comando Docker-compose up -d	29
Ilustración 14 – Comando docker ps -a	30
Ilustración 15 – Servidor 1	30
Ilustración 16 – Servidor 2	31
Ilustración 17 – Grafana log in.....	31
Ilustración 18 – Grafana Inicio.....	32
Ilustración 19 – Elección de fuente de datos	33
Ilustración 20 – Conexión a Loki	33
Ilustración 21 – Conectar el contenedor de nginx	34
Ilustración 22 – Comprobación de logs	34

1 - Introducción

En este proyecto, exploraremos una tecnología ampliamente utilizada en el mundo del desarrollo y despliegue de aplicaciones: Docker. Analizaremos las ventajas y los inconvenientes de Docker en comparación con las máquinas virtuales tradicionales, y cómo puede ayudarnos a optimizar el despliegue de nuestras aplicaciones.

Además, llevaremos a cabo un ejemplo práctico en el que utilizaremos Docker para implementar un servidor Nginx como proxy inverso y balanceador de cargas, redirigiendo las solicitudes a dos servidores Apache. También implementaremos un sistema de monitoreo de registros en el servidor Nginx, lo que nos permitirá tener una visión más clara de las solicitudes que se realizan y los tiempos de respuesta.

Ilustración 1 – Esquema servidor



2 - Análisis de contexto

En la actualidad, muchas empresas se enfrentan a desafíos relacionados con la gestión de grandes volúmenes de datos y procesos, los cuales pueden ser lentos y engorrosos. Los procesos de desarrollo y prueba también pueden llevar mucho tiempo, lo que ralentiza el progreso y limita la capacidad de la empresa. Además, estos procesos pueden resultar costosos en términos de recursos, ya que se requieren múltiples máquinas virtuales o servidores para ejecutarlos, lo que implica un alto consumo de energía.

Otro problema común es la dificultad de implementar aplicaciones desarrolladas en un entorno en otro ambiente, debido a las diferencias en configuraciones, bibliotecas y dependencias. Esto puede ocasionar errores y retrasos en la implementación.

Además, el proceso de implementación de aplicaciones puede ser largo y complicado, especialmente cuando existen múltiples dependencias y configuraciones específicas. Esto puede resultar en errores y una mayor carga de trabajo para los equipos de operaciones.

Las aplicaciones que no están diseñadas para escalar fácilmente también enfrentan problemas de rendimiento cuando se enfrentan a una mayor carga de usuarios o transacciones, lo que se traduce en tiempos de respuesta lentos y una experiencia de usuario deficiente.

Otro desafío radica en que los miembros de un equipo de desarrollo pueden trabajar en diferentes configuraciones de entorno, lo que genera problemas de compatibilidad y dificulta el intercambio de código y la colaboración eficiente.

Para abordar estos problemas, este proyecto se centrará en la utilización de Docker como una solución escalable, rápida y eficiente para la virtualización de procesos empresariales y de software. El objetivo es crear un entorno de contenedorización que permita la rápida creación de servidores seguros y controlables, así como una gestión eficiente del tráfico.

3 - Justificación

En los últimos años, la creciente demanda de soluciones tecnológicas rápidas, escalables y flexibles ha generado un gran interés en la tecnología de contenedores. Docker es una de las plataformas de contenedores más populares y ampliamente utilizadas en el mundo de la tecnología. Con Docker, los desarrolladores pueden crear, probar e implementar aplicaciones de forma rápida y sencilla en diferentes entornos de sistema operativo.

Uno de los factores más importantes desde el punto de vista empresarial para optar por una u otra tecnología es el retorno de la inversión de cada una de ellas. Cuanto más pueda minimizar una solución los costes a la par que aumenta los beneficios, mejor solución es desde esta perspectiva, y más aún en empresas grandes que necesitan generar una facturación estable a largo plazo. Dado que Docker tiene unos requisitos de infraestructura tan reducidos, las empresas son capaces de ahorrar en costes de mantenimiento y de servidores.

Además, la implementación de un servidor Nginx como proxy inverso y balanceador de cargas junto con el uso de Docker y un sistema de monitoreo de registros proporciona una infraestructura escalable y eficiente para el despliegue de aplicaciones. Al seguir estos pasos y adaptarlos a las necesidades específicas de la empresa, se puede lograr una implementación exitosa que mejore la disponibilidad, el rendimiento y la seguridad de las aplicaciones.

Por último, este proyecto también puede abrir nuevas oportunidades de crecimiento para la empresa. La infraestructura escalable y eficiente permite una mayor capacidad de respuesta a medida que la demanda aumenta de manera rápida y con un menor coste, lo que puede atraer a nuevos clientes y fortalecer la posición competitiva en el mercado.

4 - Objetivos

- Diseñar una arquitectura de contenedores Docker que permita el lanzamiento de un servidor proxy inverso con balanceo de carga y, el control y monitoreo de los registros a los contenedores que usaremos.
- Obtener información sobre Docker, para que utilizarlo y como.
- Optimizar la seguridad, la rapidez y la carga de un servidor a través de un proxy inverso con balanceo de carga
- Monitoreo de registros en el servidor proxy inverso.

5 - Marco de referencia

Estos son algunas de las áreas de conocimiento que tienen relación con el proyecto:

- Sistemas Operativos: conocimientos sobre sistemas operativos en general, como Linux, que es el sistema operativo más utilizado en el entorno Docker y el que usaremos en este proyecto como base.
- Redes: conceptos relacionados con redes, protocolos de red y configuración de redes en entornos de contenedores.
- Seguridad informática: conocimientos en seguridad informática para configurar y proteger los contenedores y la infraestructura Docker.
- Administración de sistemas: conocimientos en administración de sistemas para configurar y administrar servidores y sistemas operativos para la implementación de Docker.
- Implantación de aplicaciones web: Para la creación del html que utilizaremos en los servidores de Apache

6 - Enfoque y método seguido

En este proyecto buscaremos diseñar una arquitectura de contenedores Docker que permita el lanzamiento del servidor de balanceo de una forma sencilla, rápida y eficiente en comparación a las máquinas virtuales y otras opciones parecidas.

Esto lo realizaremos mediante el uso de imágenes que posteriormente pasaremos a un contenedor con los servicios, todo esto en un archivo de docker-compose. Esto nos ayudara a proporcionar una mayor flexibilidad, escalabilidad y eficiencia a la infraestructura de aplicaciones, haciendo un menor gasto de recursos y de tiempo.

En primer lugar, hablaremos de Docker, así como sus ventajas e inconvenientes, sus elementos y como administrarlo de forma grafica gracias al uso de portainer.io.

Después de esto, para el ejemplo práctico que haremos del servidor con balanceo de carga y control de registros explicaremos los diferentes contenedores que utilizaremos, sus archivos de configuración y por último la instalación.

7 - Costes del proyecto

Para los costes del proyecto, solo nos debemos de fijar en el dispositivo que vayamos a usar como host para este proyecto y una conexión a internet. Por ello, tendremos que buscar unos requisitos mínimos y cuáles serían los óptimos para poder realizarlo de la mejor forma posible.

En el caso de los requisitos mínimos, tendremos los siguientes:

- Un sistema operativo compatible con Docker (como Linux, Windows, o macOS)
- Una CPU de 64 bits
- Al menos 2 GB de RAM
- Al menos 20 GB de espacio en disco disponible
- Un procesador compatible con virtualización (para Docker en Windows y macOS)

Por otro lado, los recomendados son los siguientes:

- Un sistema operativo de servidor como Ubuntu o CentOS
- Al menos 4 GB de RAM, pero 8 GB mejorara considerablemente (dependiendo del número y la complejidad de los contenedores que se ejecuten)
- Al menos 50 GB de espacio en disco disponible (para almacenar imágenes y contenedores)
- Una CPU de varios núcleos (para permitir la ejecución de múltiples contenedores)
- Una tarjeta de red compatible con Docker para permitir la comunicación entre contenedores

Sin embargo, esto son solo aproximaciones, ya que algunas aplicaciones y servicios son más complejos que otros y pueden requerir más recursos para ejecutarse en contenedores Docker. También, la cantidad de contenedores que se ejecutan simultáneamente en un sistema Docker, la carga de trabajo de los contenedores y el tipo de almacenamiento puede afectar en una variación de esos requisitos.

Costes de implementación

Para poder implementarlo crearemos un ordenador desde el inicio con los precios aproximados de los componentes en el mercado. También tendremos en cuenta la mano de obra de un empleado que lo supervise y tomaremos un sueldo base de 1500 euros al mes en la zona de Córdoba.

El coste del equipo individual sería el siguiente:

Descripción		Coste
1	Sistema Operativo Linux	0 €
2	Memoria RAM 8 GB	18,50 €
3	Disco SSD 500 GB	50 €
4	Procesador AMD Ryzen 5 4500	87,50 €
5	Tarjeta de red	30 €
6	Caja	35€
7	Teclado y Ratón	20 €
8	Pantalla	90€
Total		331€

8 - Temporización

Investigación y análisis (20 Mar – 03 Abr):

- Investigar y comprender los conceptos de Docker, servidores Nginx y Apache.
- Analizar los requisitos del proyecto y determinar los objetivos específicos.

Configuración del entorno (04 Abr – 12 Abr):

- Instalar Docker en el sistema operativo adecuado.
- Configurar y preparar los servidores Apache y Nginx para su implementación en Docker.

Creación de imágenes de Docker (13 Abr – 17 Abr):

- Crear una imagen de Docker para el servidor Nginx con las configuraciones necesarias.
- Crear imágenes de Docker para los servidores Apache con las aplicaciones y configuraciones requeridas.

Creación de contenedores y configuración del servidor proxy inverso y el balanceo de cargas (18 Abr – 26 Abr):

- Crear contenedores de Docker para el servidor Nginx, los dos servidores Apache y el sistema de monitoreo de registros.
- Configurar el servidor Nginx como proxy inverso para redirigir las solicitudes a los servidores Apache y balanceo de cargas.

Implementación del sistema de monitoreo de registros (27 Abr – 10 May):

- Configurar el sistema de monitoreo de registros en el servidor Nginx.

- Establecer la recopilación y visualización de registros para realizar un seguimiento.

Pruebas y optimización (11 May – 24 May):

- Realizar pruebas exhaustivas del sistema implementado.
- Identificar posibles cuellos de botella, problemas de rendimiento o configuraciones incorrectas y optimizar en consecuencia.

Documentación y entrega (25 May – 9 Jun):

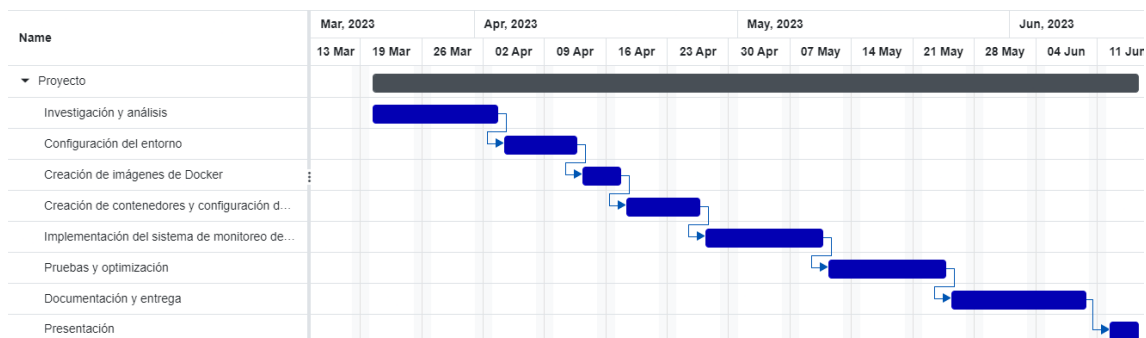
- Documentar todos los pasos de configuración y configuraciones realizadas.
- Preparar una guía de implementación y configuración para futuras referencias.
- Entregar el proyecto completado junto con la documentación al cliente o equipo correspondiente.

Presentación (10 Jun – 14 Jun):

- Creación de la presentación.

Diagrama de Gantt

Ilustración 2 – Diagrama de Gantt



9 - Docker

Docker es una plataforma de software que permite crear, desplegar y ejecutar aplicaciones en contenedores. Los contenedores son entornos aislados y portátiles que pueden ejecutarse en cualquier sistema operativo, lo que permite que las aplicaciones se ejecuten de manera consistente en diferentes entornos.

Docker utiliza la tecnología de virtualización a nivel de sistema operativo para crear contenedores, lo que significa que los contenedores comparten el kernel del sistema operativo del host en el que se ejecutan. Esto los hace más eficientes que las máquinas virtuales tradicionales, ya que no hay necesidad de replicar todo el sistema operativo en cada contenedor.

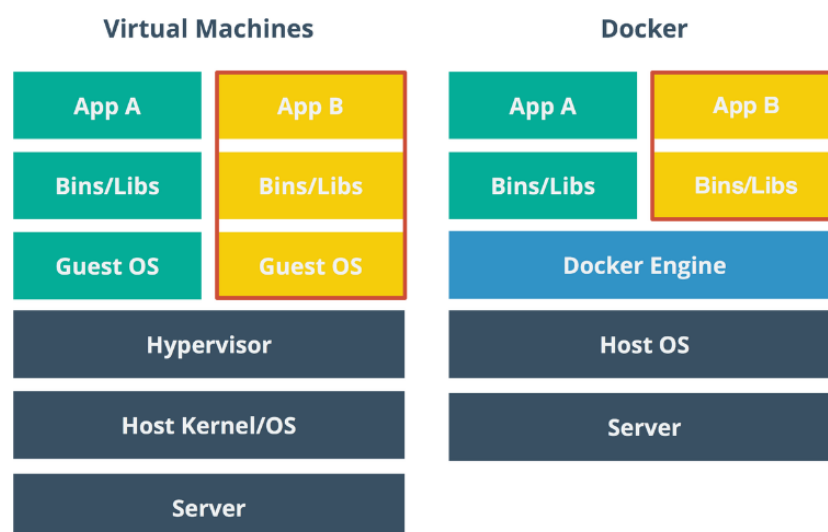
Además de permitir la creación de contenedores, Docker proporciona herramientas para administrar y orquestar contenedores en un entorno de producción, lo que facilita la creación y el despliegue de aplicaciones en escala. Docker se ha convertido en una tecnología muy popular en el mundo del desarrollo de software y la infraestructura de aplicaciones.

Ventajas

Lejos de ser un sistema operativo como tal, esta plataforma de código abierto hace uso de las funciones de aislamiento de recursos del kernel para poder dar lugar a contenedores independientes, dentro de los cuales se ejecutará una única aplicación con sus respectivas dependencias, pero funcionando siempre con un único kernel, el de la máquina real, en lugar de virtualizar uno por cada contenedor o máquina virtual.

Mientras cada aplicación en máquinas virtuales necesita contar con un sistema operativo completo virtualizado, Docker aprovecha el kernel de la máquina real y, con su motor, tan solo carga en la memoria las librerías y dependencias necesarias para ejecutar la aplicación, despreciando todo lo demás y ocupando alrededor de un 80% menos.

Ilustración 3 – Docker vs Máquinas Virtuales



Otra ventaja clara es la portabilidad, ya que todos los contenedores son portables y nos permiten de forma sencilla llevarlos a cualquier equipo con Docker sin necesidad de tener que volver a configurar nada.

Y, por último, la autosuficiencia, ya que Docker se encarga de todo y solo será necesario que la aplicación funcione. Lo único de lo que tendrás que encargarte será de las librerías, archivos y configuraciones necesarias para poder realizar su función.

Limitaciones

A pesar de que Docker ofrece muchos beneficios, también tiene algunas limitaciones que es importante considerar:

- **Complejidad:** Docker puede ser complejo y requerir cierto nivel de conocimiento técnico para configurar y administrar correctamente los contenedores.
- **Recursos limitados:** Aunque Docker está diseñado para ser liviano y eficiente, los contenedores aún requieren recursos para ejecutarse. Si se ejecutan demasiados contenedores en una sola máquina o si los recursos asignados a los contenedores son insuficientes, esto puede afectar el rendimiento de los contenedores y de la máquina host.
- **Seguridad:** Los contenedores Docker comparten el kernel del sistema operativo de la máquina host, lo que puede aumentar el riesgo de vulnerabilidades de seguridad.
- **Dependencias de la aplicación:** Si una aplicación depende de bibliotecas o componentes específicos que no están disponibles en el contenedor base de Docker, se necesitará una imagen personalizada.
- **Persistencia de datos:** Docker está diseñado para ser efímero, lo que significa que los contenedores pueden ser destruidos y recreados fácilmente.

Elementos de Docker

Docker está formado por tres componentes principales que son el motor de Docker, las imágenes y por último el Docker hub.

Motor de Docker

Es la parte central de todo el sistema Docker. El motor de Docker o Docker Engine es una aplicación que sigue arquitectura cliente-servidor. Está instalado en la máquina host y hay tres componentes en Docker Engine:

- **Servidor:** Es el demonio de la ventana acoplable llamado Dockerd. Puede crear y administrar imágenes de la ventana acoplable. Contenedores, redes, etc.
- **API de descanso:** Se utiliza para indicar al demonio de la ventana acoplable qué hacer.

- Interfaz de línea de comandos (CLI): Es un cliente que se usa para ingresar comandos de docker.

Imágenes, contenedores y volúmenes

En Docker, las imágenes, volúmenes y contenedores son componentes clave que permiten crear, distribuir y ejecutar aplicaciones en contenedores.

Imagen

Una imagen de Docker es un paquete que contiene todo lo necesario para ejecutar una aplicación. Una imagen está compuesta por múltiples capas que se combinan para formar la imagen final. Cada capa contiene los cambios realizados en la capa anterior, lo que permite que las imágenes sean ligeras y reutilizables. Las imágenes se pueden crear a partir de un archivo Dockerfile o descargarse desde un registro de Docker, como Docker Hub, y a través de estas imágenes podremos crear los contenedores que deseemos.

Contenedor

Un contenedor de Docker es una instancia en ejecución de una imagen. Un contenedor está aislado del host y de otros contenedores, pero puede comunicarse con ellos utilizando las redes de Docker.

Volumen

Un volumen de Docker es un mecanismo de persistencia de datos que permite a los contenedores acceder y almacenar datos de forma independiente del ciclo de vida del contenedor. Los volúmenes se pueden utilizar para compartir datos entre contenedores o para almacenar datos en un entorno de producción.

Docker Hub

Docker Hub es un registro de imágenes de Docker en línea, que permite a los usuarios almacenar, distribuir y gestionar las imágenes de Docker de manera centralizada. Es propiedad de Docker Inc. y ofrece un lugar para compartir y buscar imágenes de Docker, lo que lo convierte en una herramienta útil para desarrolladores, equipos de DevOps y empresas.

En Docker Hub, los usuarios pueden subir sus propias imágenes de Docker o descargar imágenes públicas compartidas por otros usuarios. También se pueden crear repositorios privados para almacenar imágenes privadas, que solo pueden ser accesibles por los usuarios autorizados. Además, Docker Hub cuenta con una amplia biblioteca de imágenes de Docker oficiales, incluyendo sistemas operativos, lenguajes de programación, bases de datos, herramientas de desarrollo y otros componentes esenciales.

Los usuarios de Docker Hub también pueden aprovechar sus características de automatización, como la integración y la entrega continuas (CI/CD), para construir y distribuir automáticamente imágenes de Docker en diferentes entornos de forma rápida y fácil. Además, Docker Hub ofrece integración con otras herramientas de gestión de contenedores, como Kubernetes, lo que permite una gestión de contenedores más completa y eficiente.

Red Docker

Docker utiliza un modelo de red que permite a los contenedores comunicarse entre sí y con el mundo exterior. El modelo de red de Docker se basa en la creación de redes virtuales que conectan los contenedores entre sí y con el host.

Cuando se crea una red en Docker, se le asigna un nombre y se especifica el tipo de driver que se utilizará para implementar la red. Cada contenedor puede ser conectado a una o varias redes, lo que permite que los contenedores se comuniquen entre sí y con el host. Cuando se conecta un contenedor a una red, se le asigna una dirección IP dentro de la red. Esto permite que los contenedores se comuniquen entre sí utilizando protocolos de red estándar como TCP/IP.

Y en el caso de que te quieras conectar con algún dispositivo de la misma red que la máquina física tendrás que poner la dirección IP del host de Docker y el puerto que le hayas asignado previamente al contenedor que quieres utilizar.

Dockerfile

Dockerfile es un archivo de texto que contiene una serie de instrucciones que Docker utiliza para construir automáticamente una imagen de Docker. Estas instrucciones especifican los pasos necesarios para crear una imagen a partir de un archivo base, instalar aplicaciones o bibliotecas adicionales, configurar el entorno y copiar archivos necesarios.

Para crear un Dockerfile en primer lugar tendrás que abrir un editor de textos y crear un archivo llamado Dockerfile.

Tras esto, tendrás que elegir una imagen la cual usaras como base que puedes elegir de Docker Hub y para asignarla tendrás que utilizar la instrucción "FROM" seguida del nombre de la imagen y la versión correspondiente. (Si no se le asigna ninguna versión, elegirá la más reciente por defecto).

Luego, para añadir las configuraciones que deseamos, copiar archivos, instalar dependencias o establecer variables de entorno haremos uso de las instrucciones COPY, que nos permitirá copiar archivos que se encuentren en la máquina física, RUN, para ejecutar comandos durante la construcción de la imagen de Docker, y ENV, que establece las variables de entorno.

Podrás usar también las instrucciones EXPOSE, que te permitirá exponer un puerto al exterior, y CMD, para especificar el comando que quieres que se ejecute cuando se inicie el contenedor.

Posteriormente para crear la imagen deberás realizar el siguiente comando:


```
Docker build -t "nombre" .
```

Con esto tendrías la imagen creada y lista para ser usada o subirla a Docker Hub y que el resto de usuarios la pueda usar.

Docker Compose

Docker Compose es una herramienta de Docker que permite definir y ejecutar aplicaciones multi-contenedor. Con Docker Compose, se pueden definir servicios que se ejecutan en contenedores, así como la forma en que se conectan y se comunican entre sí.

En lugar de tener que definir y ejecutar cada contenedor individualmente, Docker Compose permite definir toda la aplicación en un archivo YAML, lo que hace que sea mucho más fácil de manejar y escalar.

Docker Compose se utiliza típicamente para aplicaciones que tienen varios componentes, como una aplicación web con un servidor de base de datos y un servidor web, o una aplicación que utiliza múltiples microservicios.

Para utilizar Docker Compose, se debe crear un archivo YAML que defina los servicios y sus configuraciones. A continuación, se puede utilizar el comando `docker-compose up` para ejecutar la aplicación definida en el archivo YAML.

El archivo YAML de Docker Compose puede definir diferentes aspectos de la aplicación, como los servicios, las redes y los volúmenes. Cada servicio se define por separado en el archivo YAML y se especifica qué imagen de Docker utilizar, qué puertos exponer y cómo conectarse a otros servicios.

Además, Docker Compose permite definir variables de entorno, volúmenes compartidos y redes personalizadas, lo que hace que sea fácil configurar y gestionar la infraestructura de la aplicación.

Instalación de Docker en Windows

Para la instalación de Docker en Windows, en primer lugar, iremos a la página oficial de Docker para allí descargar el archivo `DockerDesktopInstaler.exe`, y lo instalamos. Sitio web oficial de Docker: <https://www.docker.com/products/docker-desktop>

Una vez que se haya descargado el archivo de instalación, haz doble clic para abrirlo y sigue las instrucciones en pantalla. Se te pedirá que aceptes los términos de servicio y que autorices la instalación de componentes adicionales, como Hyper-V.

Cuando se complete la instalación, inicia Docker Desktop. Es posible que se te solicite que permitas que la aplicación acceda a tu red.

Una vez que Docker Desktop esté en funcionamiento, podrás ver el icono de Docker en la barra de tareas de Windows. Haz clic derecho en el icono para acceder al menú y selecciona "Configuración" para personalizar tus opciones de Docker.

Para verificar que Docker se ha instalado correctamente, abre una ventana de PowerShell o de la línea de comandos y ejecuta el siguiente comando: `docker version`. Esto debería mostrar la versión actual de Docker que has instalado.

Instalación de Docker en Ubuntu

Para este proyecto, haremos las pruebas en un equipo con Ubuntu 22.04 ya que Docker está mejor adaptada para ser utilizada en sistemas operativos de Linux. En primer lugar, tendremos que actualizar los repositorios.

```
sudo apt update
```

Después de esto, instalaremos los paquetes necesarios para Docker con el siguiente comando:

```
sudo apt install ca-certificates curl gnupg lsb-release
```

Una vez instalado añadiremos la clave GPG oficial de Docker de la siguiente manera:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Con el siguiente comando configuraremos el repositorio estable:

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Con el repositorio correctamente instalado tendremos que actualizar de nuevo los repositorios y posteriormente instalaremos Docker y Docker compose.

```
sudo apt update
```

```
sudo apt install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

De esta manera, si se instala correctamente podremos comprobarlo y mirar su versión con el comando:

```
sudo docker -v
```

Comandos importantes

Para iniciar, detener o reiniciar un contenedor tienes que utilizar el comando Docker junto a la acción que quieras realizar. Además, si quieres eliminar un contenedor, este tiene que estar detenido. Los comandos son los siguientes:

```
Docker start "3 primeros caracteres del contenedor" (iniciar)
```

```
Docker stop "3 primeros caracteres del contenedor" (detener)
```

DOCKER

Docker restart “3 primeros caracteres del contenedor” (reiniciar)

Docker rm “3 primeros caracteres del contenedor” (eliminar)

Para convertir un contenedor a una imagen deberemos tener el contenedor que queremos usar apagado y posteriormente usaremos el comando “commit” .

Docker commit “nombre o id del contenedor” “nombre que le queremos asignar a la imagen”

Para poder ver todos los contenedores, volúmenes, imágenes y redes podremos usar estos comandos

docker ps -a (contenedores)

docker volume ls (volúmenes)

docker images (imágenes)

docker network ls (redes)

Por último, para poder iniciar una sesión interactiva en la shell de contenedor, utilizaremos el siguiente comando:

Docker exec -it “nombre o id del contenedor” bash

Portainer.io

Para simplificar el monitoreo y la administración de los contenedores que tengamos, así como los volúmenes e imágenes, usaremos portainer.io, una herramienta web que se ejecuta en un contenedor de Docker y se puede acceder desde cualquier navegador web en cualquier lugar. La interfaz de usuario de Portainer.io es intuitiva y fácil de usar, lo que facilita la gestión de contenedores y la administración de recursos en Docker.

En primer lugar, para instalar portainer.io tendremos que descargar la imagen que encontraremos en Docker Hub y nos explicaran como bajarla. Tendremos que elegir la versión llamada “portainer-ce” (community-edition) ya que la versión estándar esta obsoleta. Para descargar la imagen usaremos el siguiente comando:

docker pull portainer/portainer-ce

Después de eso, arrancaremos un contenedor con esa imagen que hemos bajado, que asignaremos al puerto 9000 para poder acceder desde el navegador, también le daremos un nombre, haremos que siempre que se mantenga encendido y montaremos el socket de Docker en el contenedor para que podamos interactuar con Docker. Todo esto lo haremos con el siguiente comando:

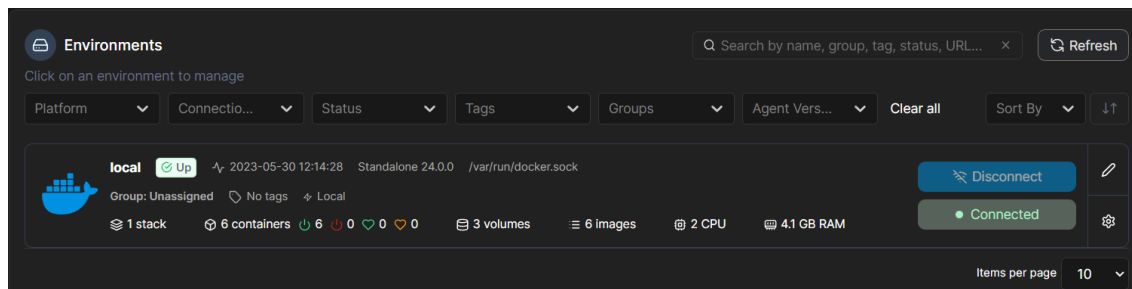
docker run -d -p 9000:9000 --name=portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer-ce

DOCKER

Una vez creado iremos al navegador, y buscaremos la ip de la maquina en la que está instalado Docker, junto el puerto 9000 que le hemos asignado. Tras esto, nos aparecerá la interfaz de portainer.io y para asignar un usuario.

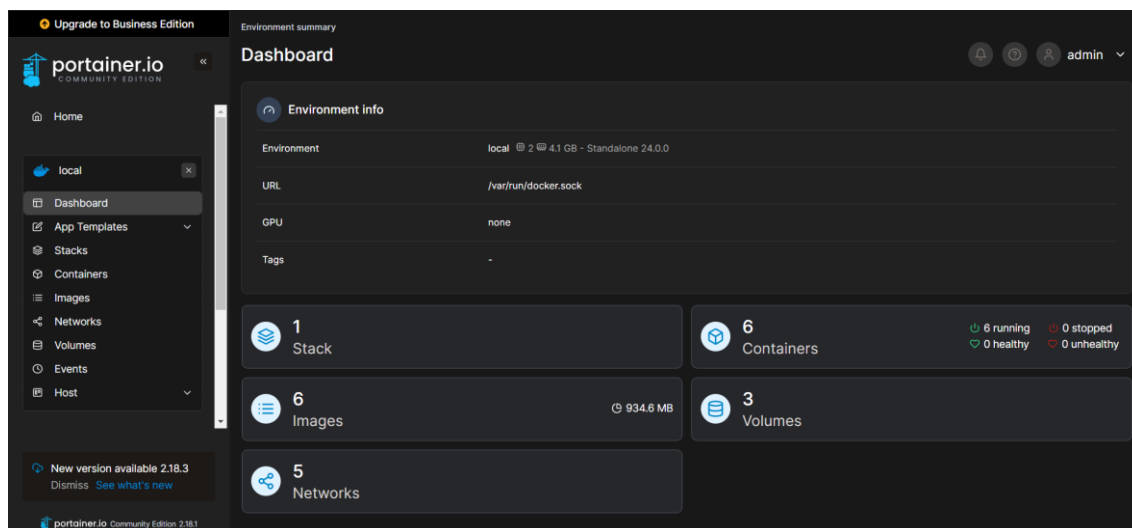
Una vez dentro nos aparecerán los entornos disponibles, que en nuestro caso será el entorno de Docker, junto a unas especificaciones de este como pueden ser el número de stacks (docker-compose), contenedores, el estado en el que están, volúmenes, imágenes, CPU y RAM utilizada.

Ilustración 4 – Portainer.io Inicio



Al entrar en el dashboard, nos aparecerán más detallados los apartados de los stacks, contenedores, volúmenes, imágenes y redes que tenemos disponibles.

Ilustración 5 – Portainer.io Dashboard



En el apartado de stacks podremos ver los conjuntos de contenedores que tenemos y cuales están en cada uno de esos conjuntos. También te mostrara cuando fue creado, de que tipo es, el creador y cuales de los contenedores están activos.

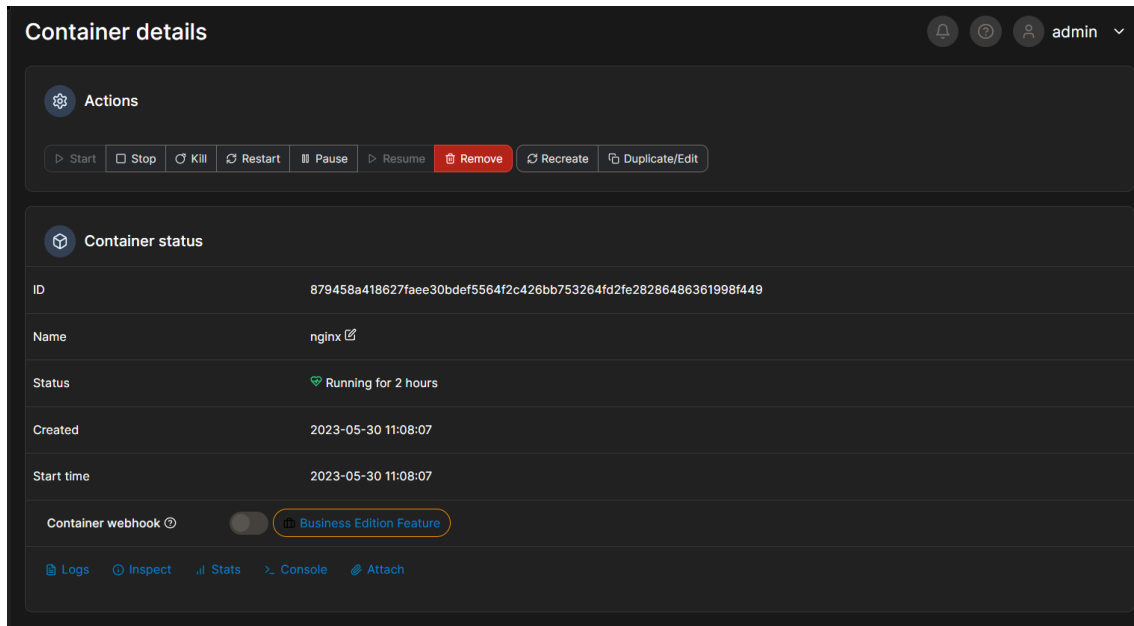
En el apartado de contenedores, nos mostrara tanto los nombres, el estado en el que se encuentra, la imagen que esta utilizando, fecha de creación, la ip correspondiente y el puerto que se le ha asignado.

Una vez dentro de cada contenedor, tendremos un apartado de acciones donde podremos iniciarlo, reiniciarlo, pararlo o eliminarlo, así como recrearlo o duplicarlo. También nos dará información y detalles del contenedor. Y por último, nos dará la

DOCKER

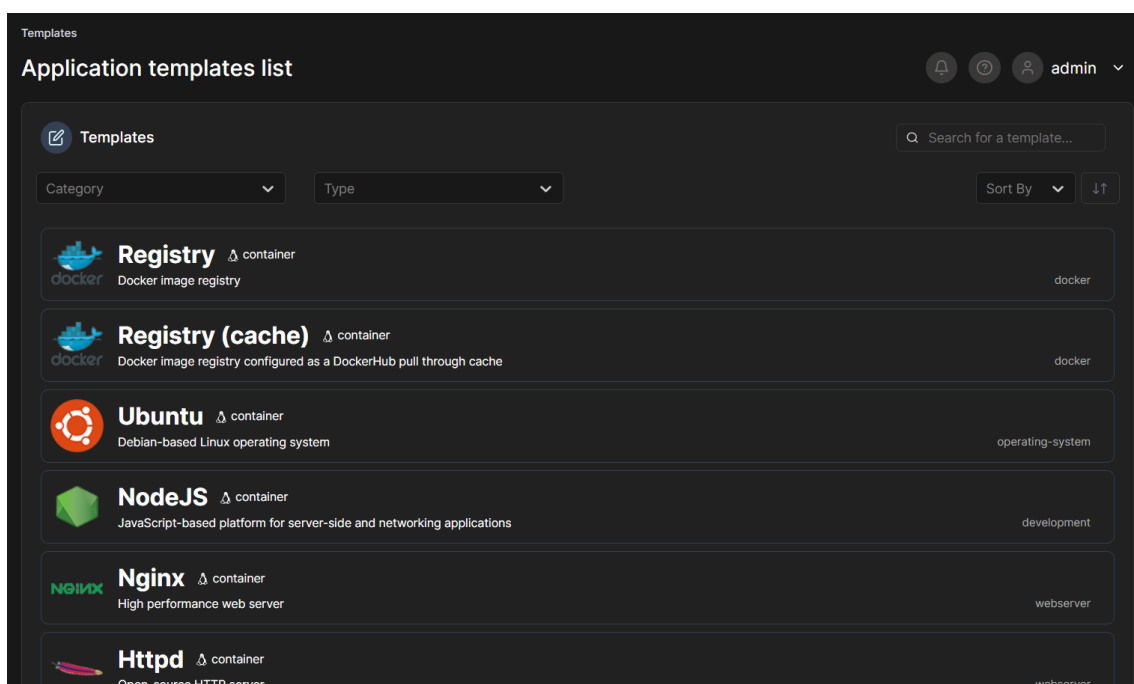
opción de mirar los logs, inspeccionar el contenedor, mirar estadísticas o entrar en la consola de este.

Ilustración 6 – Portainer.io Detalles de contenedor



En el apartado de imágenes nos encontramos las imágenes que tenemos disponibles en las que viene indicadas el tag, id, versión, tamaño, fecha de creación y si está en desuso para poder eliminarla. A su vez, hay un apartado llamado “App Templates” en el cual nos aparecerán todas las imágenes disponibles y la opción para descargarla y desplegar un contenedor.

Ilustración 7 – Portainer.io Lista de imágenes



Por último, están los volúmenes y las redes, que, en el caso de los volúmenes, indica la ruta en donde se encuentran en la maquina física, y en las redes indica la ip de la red, así como su driver.

10 - Ejemplo practico

En este apartado, buscaremos configurar un balanceo de carga utilizando Nginx y dos servidores Apache, y al mismo tiempo haremos un monitoreo y un registro de los logs al servidor Nginx con ayuda de plugin de Grafana llamado Loki y Grafana que nos permitirá la visualización de los logs. Este enfoque se llevará a cabo utilizando Docker, una plataforma que permite el despliegue de aplicaciones en contenedores.

El balanceo de carga es una técnica utilizada en infraestructuras de servidores para distribuir el tráfico de red de manera equilibrada entre varios servidores o recursos, como servidores web, aplicaciones o bases de datos. El objetivo principal del balanceo de carga es optimizar el rendimiento, la escalabilidad y la disponibilidad del sistema. Algunas ventajas clave del balanceo de carga son las siguientes:

- **Mejora del rendimiento:** Al distribuir las solicitudes de los clientes entre varios servidores, se evita la sobrecarga de un único servidor, lo que resulta en una mayor capacidad de procesamiento y una respuesta más rápida. El balanceo de carga permite aprovechar al máximo los recursos disponibles y reducir los cuellos de botella.
- **Mayor escalabilidad:** Cuando la demanda de tráfico aumenta, el balanceo de carga permite escalar horizontalmente agregando nuevos servidores al grupo. De esta manera, se puede manejar un mayor número de solicitudes sin afectar el rendimiento. La escalabilidad horizontal es más sencilla de implementar con el balanceo de carga, en comparación con depender de un único servidor más potente.
- **Alta disponibilidad:** Al distribuir el tráfico entre varios servidores, se mejora la disponibilidad del sistema. Si un servidor falla o se vuelve inaccesible, el balanceador de carga redirige automáticamente las solicitudes a otros servidores activos, evitando interrupciones en el servicio. Esto mejora la tolerancia a fallos y asegura que los usuarios puedan acceder a la aplicación o servicio de manera continua.
- **Optimización de recursos:** El balanceo de carga permite aprovechar al máximo los recursos disponibles en el entorno de servidores. En lugar de depender de un solo servidor con recursos limitados, se pueden utilizar varios servidores más pequeños pero eficientes para distribuir la carga de trabajo. Esto mejora la eficiencia y reduce los costos al evitar la infrautilización de recursos.
- **Facilidad de mantenimiento:** Al utilizar balanceo de carga, es posible realizar actualizaciones y mantenimiento en los servidores sin interrumpir el servicio. Los servidores individuales se pueden desactivar temporalmente para aplicar parches o realizar tareas de mantenimiento, mientras que el balanceador de carga redirige el tráfico a otros servidores activos. Esto garantiza una mayor disponibilidad y evita tiempos de inactividad no planificados.

En este caso, se utilizará el servidor Nginx como punto de entrada para el tráfico entrante y se distribuirá de manera equitativa a los dos servidores Apache, a los cuales

no se podrá acceder desde el exterior ya que estarán en la red interna de Docker y solo se abrirá el puerto 80 para habilitar el http en el servidor nginx.

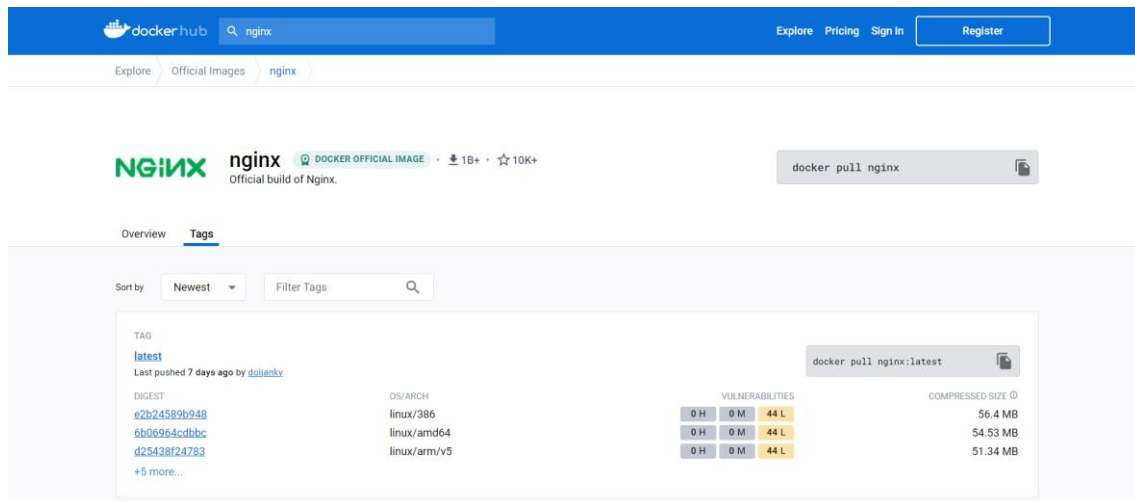
Nginx

Nginx es un servidor web y proxy inverso de alto rendimiento. Se caracteriza por su arquitectura ligera y eficiente, diseñada para manejar grandes cargas de tráfico y proporcionar un alto rendimiento incluso en sistemas con recursos limitados. Está basado en un modelo de evento impulsado por el asincrónico I/O (entrada/salida) no bloqueante, lo que significa que puede manejar múltiples solicitudes simultáneamente sin necesidad de crear hilos adicionales por solicitud, lo que consume menos memoria y recursos del sistema.

Las principales funciones que tendrá en este caso será haciendo de proxy inverso, actúa como un intermediario entre los clientes y los servidores de apache, distribuyendo las solicitudes de manera equilibrada. También pudiendo realizar funciones de caché para acelerar la entrega de contenido y mejorar el rendimiento del sitio web. Y como balanceo de carga, garantizando una carga equilibrada y una mayor disponibilidad del servicio.

Para el ejemplo tendremos que elegir una versión para Nginx que buscaremos en Docker Hub. En nuestro caso, elegiremos la última, por lo que tendremos que descargar la imagen de la siguiente forma “docker pull nginx:latest”.

Ilustración 8 – Imagen Nginx Docker Hub



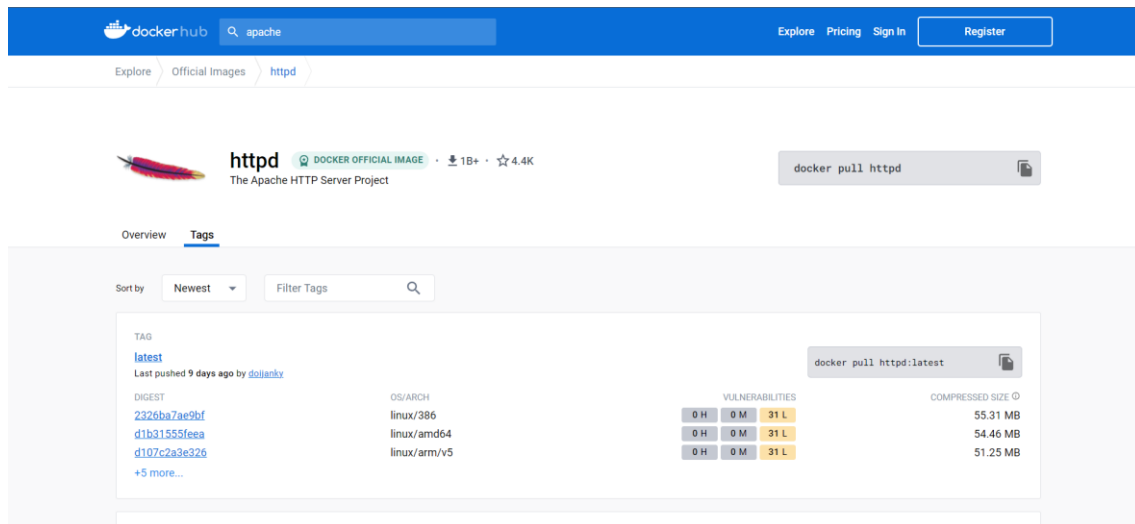
Apache

Apache es un servidor web de código abierto ampliamente utilizado en el mundo de la tecnología. Es uno de los servidores web más populares y está disponible en múltiples plataformas, incluyendo Unix, Linux, Windows y macOS.

El propósito principal de Apache en esta practica es servir contenido web y proporcionar servicios web a los usuarios que acceden a través del balanceador de carga.

Para este caso también usaremos la imagen más reciente de apache en Docker Hub que en este caso se descargara de la siguiente manera “docker pull httpd:latest”

Ilustración 9 – Imagen Apache Docker Hub



Loki

Para el monitoreo de logs usaremos Docker Compose para configurar Grafana con Loki y reenviar sus registros desde sus contenedores en ejecución a Loki.

Loki es un sistema de registro distribuido diseñado para recolectar, almacenar y consultar registros o logs de aplicaciones. Fue desarrollado por Grafana Labs como una solución eficiente y escalable para el monitoreo de registros en entornos de contenedores y microservicios.

El propósito principal de Loki es ayudar a los desarrolladores y operadores a gestionar y analizar grandes volúmenes de registros generados por aplicaciones y servicios. Algunas características y usos destacados de Loki son:

Recolección de registros: Loki permite la recolección centralizada de registros de múltiples fuentes, como aplicaciones, servicios y sistemas distribuidos. Puede recibir registros en formato de texto estructurado como JSON, lo que facilita la extracción y análisis de datos relevantes.

Almacenamiento eficiente: En lugar de almacenar registros de forma tradicional en archivos planos o bases de datos, Loki utiliza una arquitectura de almacenamiento optimizada. Utiliza un almacenamiento en columna llamado "índices de etiquetas" que comprime y almacena eficientemente los registros, lo que permite un consumo de recursos más bajo y un acceso rápido a los datos.

Consultas de registros: Loki ofrece un lenguaje de consulta basado en etiquetas para realizar búsquedas y filtrado de registros. Permite consultar registros en función de diferentes criterios, como rangos de tiempo, etiquetas específicas o palabras clave. Esto facilita el análisis y la búsqueda de información relevante en grandes volúmenes de registros.

Integración con herramientas de visualización: Loki se integra de forma nativa con herramientas populares de visualización de datos, como Grafana. Esto permite crear

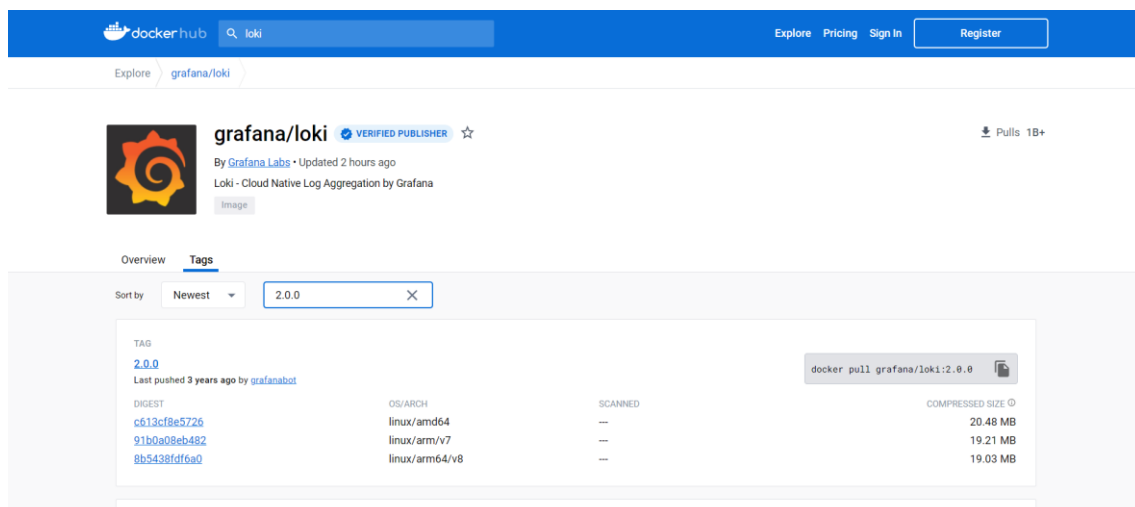
paneles y gráficos interactivos para analizar y representar visualmente los datos de registro, lo que facilita la identificación de patrones, tendencias y problemas en los sistemas.

Escalabilidad y tolerancia a fallos: Loki fue diseñado pensando en la escalabilidad y la resiliencia. Puede manejar grandes volúmenes de registros y se adapta bien a entornos distribuidos y en contenedores. Además, utiliza técnicas de replicación y respaldo para garantizar la disponibilidad y la integridad de los datos de registro.

Además, tendremos que instalar un plugin de Docker llamado "loki-docker-driver" proporcionado por Grafana Labs, el cual se integra con el sistema de registro Loki. Este plugin permite la captura y el envío de los registros de contenedores Docker directamente a Loki para su almacenamiento y análisis.

Para la imagen de loki, tendremos que ver que configuración es compatible con la versión de grafana que utilizaremos por lo que en este caso descargaremos la versión 2.0.0 para Loki, y el comando para ello sería "docker pull grafana/loki:2.0.0". Aunque no sea la última versión, funciona correctamente.

Ilustración 10 – Imagen Loki Docker Hub



Grafana

Grafana es una plataforma de código abierto utilizada para la visualización y el monitoreo de métricas y registros de sistemas. Proporciona una interfaz gráfica intuitiva y personalizable que permite crear paneles y gráficos interactivos para representar datos en tiempo real. Grafana Loki se integra con Grafana y es una opción popular para el análisis y visualización de registros junto con las métricas.

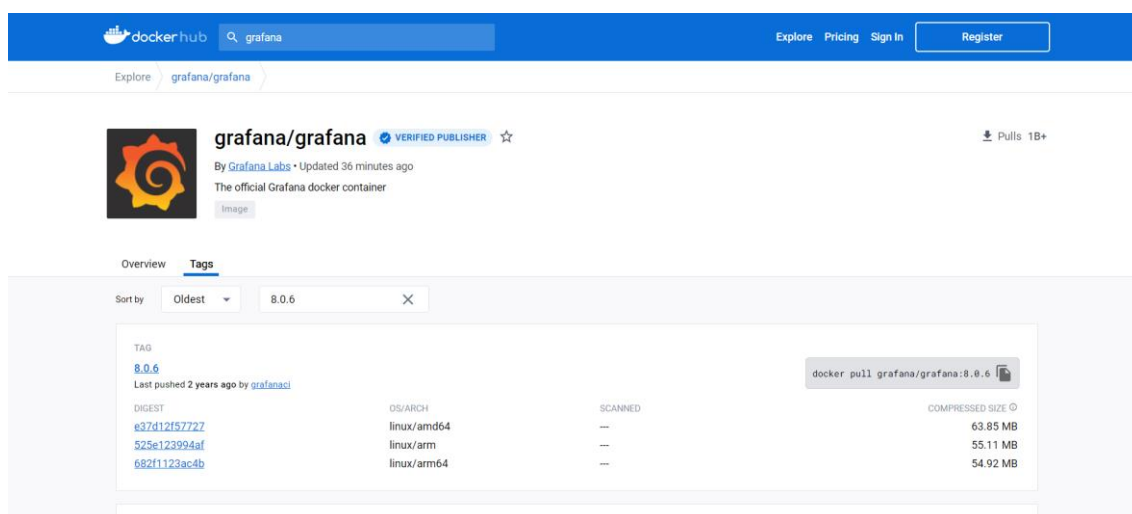
Cuando se combinan Grafana y Grafana Loki, se obtiene una solución integral para el monitoreo y la observabilidad de sistemas. Algunas de las capacidades y beneficios de utilizar Grafana con Grafana Loki son los siguientes:

- Integración fácil: Grafana y Grafana Loki están diseñados para trabajar juntos de manera transparente. Grafana se integra con Grafana Loki como una fuente de datos nativa, lo que facilita la configuración y el uso conjunto de ambos sistemas.

- Visualización de registros: Grafana proporciona paneles y gráficos personalizables para representar visualmente los registros almacenados en Grafana Loki. Esto permite analizar y buscar información en los registros de manera más efectiva, lo que facilita la identificación de problemas, tendencias y patrones.
- Consulta de registros: A través de Grafana, se pueden realizar consultas y filtros avanzados en los registros almacenados en Grafana Loki. Grafana ofrece un lenguaje de consulta y filtros basados en etiquetas que permiten extraer datos específicos de los registros para su visualización y análisis.
- Integración con métricas: Grafana también es compatible con la visualización y el monitoreo de métricas de sistemas. Esto significa que se pueden combinar los datos de registros de Grafana Loki con métricas de otras fuentes, lo que proporciona una visión más completa y contextualizada del estado y el rendimiento del sistema.
- Escalabilidad y rendimiento: Tanto Grafana como Grafana Loki están diseñados para ser escalables y eficientes en entornos de alta carga. Pueden manejar grandes volúmenes de registros y métricas, y su arquitectura permite una respuesta rápida y un rendimiento óptimo en el análisis y la visualización de datos.

Para la imagen, como tenemos que conseguir que este contenedor sea compatible con la extensión de Grafana Loki que hemos elegido, versión 2.0.0, elegiremos en este caso la versión 8.0.6 y usaremos este comando para la descarga de la imagen “`docker pull grafana/grafana:8.0.6`”.

Ilustración 11 – Imagen Grafana Docker Hub



Archivos de configuración

Ahora pasaremos a la configuración de todos los contenedores usados, sus archivos de configuración y enlazaremos el servidor de nginx con grafana loki y posteriormente con grafana para poder visualizar correctamente los logs.

En primer lugar, tendremos que configurar el archivo de nginx.conf para que nos permita dirigir de manera equilibrada las peticiones a los servidores de apache, y asignarlos junto al puerto necesario.

Después, asignaremos los archivos html para las paginas web de los servidores apache y con la ayuda de Bootstrap personalizaremos el diseño del html.

Para la recolección de los logs, usaremos un archivo de configuración de loki e instalaremos el plugin necesario.

Por último, uniremos loki junto a grafana para poder visualizarlos de forma sencilla y a su vez también poder clasificarlos.

Todo esto lo ejecutaremos a través de un archivo de docker-compose, que los unirá, asignara los puertos y unirá los archivos que hemos creado con el contenedor asignado.

Nginx.conf

Este archivo de configuración de nginx se encontrará en la ruta `"/etc/nginx/nginx.conf"` y establece un servidor Nginx con balanceo de carga hacia dos servidores Apache. Configura el número de procesos de trabajo de Nginx en automático, define la ubicación del archivo de registro de errores y el archivo PID de Nginx. Además, establece el número máximo de conexiones simultáneas que pueden manejar los trabajadores de Nginx.

Dentro del bloque HTTP, se configura un grupo de servidores llamado "apache" utilizando la directiva upstream, que incluye los servidores "apache1" y "apache2" con sus respectivas direcciones IP y puertos (80).

Dentro del bloque server, se configura Nginx para escuchar en el puerto 80. En la ubicación "/", se realiza un proxy_pass a los servidores Apache definidos en el grupo "apache".

```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log notice;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    upstream apache {
```

```

server apache1:80;
server apache2:80;
}
server {
    listen 80;
    server_name example.com;
    location / {
        proxy_pass http://apache;
    }
}
}

```

Apache html

Para los servidores de Apache se utilizarán un archivo html, un css obtenido gracias a Bootstrap y las imágenes guardadas en una carpeta llamada “imágenes”. Estos se guardaran en la ruta “/usr/local/apache2/htdocs/” del contenedor de apache y cada uno de ellos tendrá cambiado el numero del servidor para poder comprobar que el balanceo se realiza correctamente. El html usado es el siguiente:

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-
scale=1.0">
    <link rel="icon" type="image/png" href="/imagenes/docker.png"/>
    <link rel="stylesheet" href="/css/bootstrap.min.css">
    <title>Servidor 1</title>
</head>
<body>
    <header>
        <nav class="navbar navbar-dark bg-dark">
            <div class="container-fluid">
                <a class="navbar-brand" href="#">

```

```

    Proyecto de Docker Servidor 1

  </a>

  <span class="navbar-text aling-middle">

    Manuel Luna Madrid

  </span>

</div>

</nav>

</header>

<main class="container">

  <br>

  <h1 class="text-center">Servidor 1</h1>

  <br>

  <div class="text-center">

    >

  </div>

  <br><br>

  <h2 class="text-center">Docker</h2>

  <br>

  <p>Este es uno de los servidores que forma parte de nuestro sistema de balanceo de carga
para el sitio web de nuestra empresa. Como parte de este sistema, el servidor 1 trabaja junto
con otros ...</p>

  <p>Nuestro sistema de balanceo de carga se encarga de dirigir el tráfico de manera
efectiva, permitiendo que los usuarios accedan al sitio web sin interrupciones. Cada servidor
que forma parte...</p>

</main>

</body>

</html>

```

Loki-config.yml

Para loki, necesitaremos el archivo de configuración que es el archivo estándar que recomienda la página oficial de loki.

Este archivo establece opciones como la autenticación desactivada, el puerto de escucha HTTP en el 3100, la configuración del ciclo de vida del ingester, el esquema de

almacenamiento de registros, opciones de almacenamiento, el proceso de compactación, límites de procesamiento de muestras antiguas, configuración del almacenamiento de chunks, administrador de tablas para la retención de datos y configuración del motor de reglas y alertas.

Todo esto puede ser modificado con los periodos que uno quiera para su caso, sin embargo, cambiar estos pueden acelerar el trabajo y hacer que ocupen más o que usen más recursos. El archivo de configuración es el siguiente.

```
auth_enabled: fals
server:
  http_listen_port: 3100
ingester:
  lifecycler:
    address: 127.0.0.1
    ring:
      kvstore:
        store: inmemory
      replication_factor: 1
    final_sleep: 0s
  chunk_idle_period: 1h    # Any chunk not receiving new logs in this time will be flushed
  max_chunk_age: 1h       # All chunks will be flushed when they hit this age, default is 1h
  chunk_target_size: 1048576 # Loki will attempt to build chunks up to 1.5MB, flushing first if
                             # chunk_idle_period or max_chunk_age is reached first
  chunk_retain_period: 30s # Must be greater than index read cache TTL if using an index
                             # cache (Default index read cache TTL is 5m)
  max_transfer_retries: 0  # Chunk transfers disabled
schema_config:
  configs:
    - from: 2020-10-24
      store: boltdb-shipper
      object_store: filesystem
      schema: v11
      index:
        prefix: index_
        period: 24h
```

```
storage_config:
  boltdb_shipper:
    active_index_directory: /loki/boltdb-shipper-active
    cache_location: /loki/boltdb-shipper-cache
    cache_ttl: 24h      # Can be increased for faster performance over longer query periods,
                        # uses more disk space
    shared_store: filesystem
  filesystem:
    directory: /loki/chunks
  compactor:
    working_directory: /loki/boltdb-shipper-compactor
    shared_store: filesystem
  limits_config:
    reject_old_samples: true
    reject_old_samples_max_age: 168h
  chunk_store_config:
    max_look_back_period: 0s
  table_manager:
    retention_deletes_enabled: false
    retention_period: 0s
  ruler:
    storage:
      type: local
    local:
      directory: /loki/rules
    rule_path: /loki/rules-temp
    alertmanager_url: http://localhost:9093
  ring:
    kvstore:
      store: inmemory
  enable_api: true
```


Plugin de loki

Este es un plugin que contiene un driver de loki el cual nos ayudara a mandar los logs generados en el servidor nginx al servicio de loki. Este se especificará de forma más detallada en el archivo docker-compose.yml

```
docker plugin install grafana/loki-docker-driver:latest --alias loki --grant-all-permission
```

Docker-compose.yml

En este Docker, recopilaremos todos los archivos mencionados anteriormente y los asignaremos en el contenedor correspondiente. También, crearemos los volúmenes necesarios para Grafana y Loki, que estarán conectados con las rutas “/var/lib/grafana” y “/loki” respectivamente.

Para el servicio de Grafana se utiliza la imagen grafana/grafana:8.0.6 de Grafana, se configura para reiniciarse automáticamente a menos que se detenga explícitamente y el servicio se expone en el puerto 3000:3000 para acceder a la interfaz web de Grafana.

En el servicio de Loki utiliza la imagen grafana/loki:2.0.0 de Loki, y al igual que el servicio de Grafana, se configura para reiniciarse automáticamente a menos que se detenga explícitamente. El servicio se expone en el puerto 3100:3100 y se configura para ejecutar el comando '-config.file=/mnt/config/loki-config.yml', que especifica la ubicación del archivo de configuración de Loki.

En el servicio de nginx utilizamos la imagen más reciente y también se configura para reiniciarse automáticamente a menos que se detenga explícitamente. Además, se configura para enviar los registros a Loki utilizando el controlador de registro loki y se especifica la URL de Loki para el envío de registros. El servicio se expone en el puerto 80:80 para acceder al servidor web nginx. Además, se crea una dependencia de los servidores para crear el balanceo de carga.

Y por último, los servidores de apache utilizan la imagen httpd:latest para dos instancias de servidores web Apache separados. Se configuran volúmenes para montar los directorios ./servidor1 y ./servidor2 dentro de los contenedores, lo que permite el acceso a los archivos del servidor web.

```
version: "3.7"

volumes:
  grafana-data:
  loki-data:

services:
  grafana:
    image: grafana/grafana:8.0.6
    container_name: grafana
```

```
restart: unless-stopped
volumes:
- grafana-data:/var/lib/grafana
ports:
- 3000:3000
loki:
image: grafana/loki:2.0.0
container_name: loki
restart: unless-stopped
volumes:
- ./loki-config.yml:/mnt/config/loki-config.yml
- loki-data:/loki
ports:
- 3100:3100
command:
- '-config.file=/mnt/config/loki-config.yml'
nginx:
image: nginx
container_name: nginx
restart: unless-stopped
volumes:
- ./nginx.conf:/etc/nginx/nginx.conf
logging:
driver: loki
options:
loki-url: "http://localhost:3100/loki/api/v1/push"
ports:
- 80:80
depends_on:
- apache1
- apache2
apache1:
```

```

image: httpd:latest

volumes:
  - ./servidor1:/usr/local/apache2/htdocs/

apache2:
  image: httpd:latest

  volumes:
    - ./servidor2:/usr/local/apache2/htdocs/

```

Instalación

Una vez creados los archivos de configuración, todos ellos en las rutas especificadas dentro de la carpeta que estemos usando para docker, tendremos que ejecutar el archivo docker-compose.yml.

Descargar las imágenes

En primer lugar, comprobaremos que las imágenes están instaladas correctamente usando el siguiente comando:

```
Docker-compose pull
```

Ilustración 12 – Comando Docker-compose pull

```

docker@docker:~/docker$ docker-compose pull
Pulling grafana ... done
Pulling loki ... done
Pulling apache1 ... done
Pulling apache2 ... done
Pulling nginx ... done
docker@docker:~/docker$

```

Ejecutar Docker-compose.yml

Después de comprobar que están correctamente instaladas, ejecutaremos el archivo de docker-compose.yml con el siguiente comando:

```
Docker-compose up -d
```

Ilustración 13 – Comando Docker-compose up -d

```

docker@docker:~/docker$ docker compose up -d
[+] Building 0.0s (0/0)
[+] Running 5/5
✓ Container loki Started
✓ Container docker-apache1-1 Started
✓ Container docker-apache2-1 Started
✓ Container grafana Started
✓ Container nginx Started
docker@docker:~/docker$

```

La opción `-d` es una bandera que se utiliza para ejecutar los contenedores en segundo plano. Esto significa que los contenedores se ejecutarán como procesos en segundo plano y liberarán la terminal para que puedas seguir utilizando la línea de comandos.

También, podemos comprobar que todos están corriendo correctamente y no ha habido ningún problema al iniciarlos:

```
Docker ps -a
```

Ilustración 14 – Comando `docker ps -a`

```
docker@docker: ~/docker$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
7cc15698dff2	nginx	"/docker-entrypoint..."	About a minute ago	Up About a minute
de5297bf3514	grafana/grafana:8.0.6	"/run.sh"	About a minute ago	Up About a minute
4a1da7fc5225	httpd:latest	"httpd-foreground"	About a minute ago	Up About a minute
173631c4167f	httpd:latest	"httpd-foreground"	About a minute ago	Up About a minute
7678604622eb	grafana/loki:2.0.0	"/usr/bin/loki -conf..."	About a minute ago	Up About a minute
fec33e0d5604	portainer/portainer-ce	"/portainer"	6 weeks ago	Exited (2) 5 days ago

```
docker@docker: ~/docker$
```

Visualizar el servidor

Tras esto iremos al navegador y accederemos al servicio de nginx a través de la ip de la maquina en la que tenemos montado docker, o en el caso que estemos en ella podremos buscar como localhost.

Ilustración 15 – Servidor 1



Servidor 1



Docker

Este es uno de los servidores que forma parte de nuestro sistema de balanceo de carga para el sitio web de nuestra empresa. Como parte de este sistema, el servidor 1 trabaja junto con otros servidores para distribuir el tráfico de la web de manera equitativa, lo que ayuda a garantizar que el sitio sea rápido y esté disponible para todos los usuarios, incluso en momentos de alta demanda.

Nuestro sistema de balanceo de carga se encarga de dirigir el tráfico de manera efectiva, permitiendo que los usuarios accedan al sitio web sin interrupciones. Cada servidor que forma parte del sistema está configurado de manera idéntica, lo que significa que todos tienen la misma información y están preparados para responder a las solicitudes de los usuarios de manera uniforme.

Al ver esto comprobamos que se ha lanzado correctamente y que se puede ver que estamos en el servidor de apache 1 por lo que nos a redirigido correctamente el servidor nginx y entonces sabemos que funciona el proxy inverso. También probaremos al recargar la página, que nos tendrá que dar el acceso al servidor de apache 2, todo esos en la misma ip y en el puerto 80.

Ilustración 16 – Servidor 2



Servidor 2



Docker

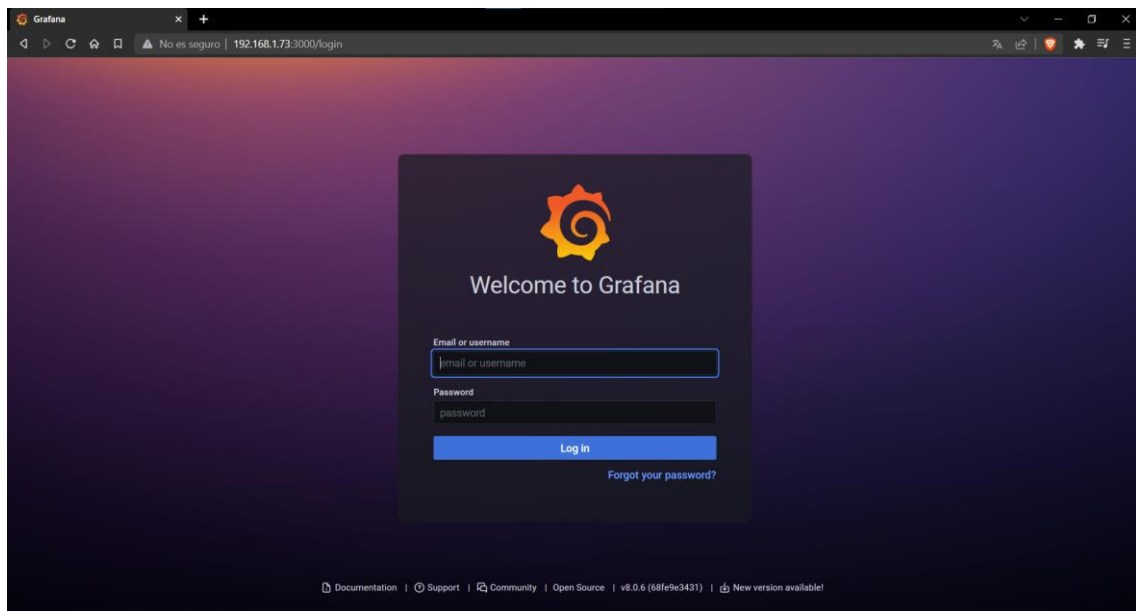
Este es uno de los servidores que forma parte de nuestro sistema de balanceo de carga para el sitio web de nuestra empresa. Como parte de este sistema, el servidor 2 trabaja junto con otros servidores para distribuir el tráfico de la web de manera equitativa, lo que ayuda a garantizar que el sitio sea rápido y esté disponible para todos los usuarios, incluso en momentos de alta demanda.

Nuestro sistema de balanceo de carga se encarga de dirigir el tráfico de manera efectiva, permitiendo que los usuarios accedan al sitio web sin interrupciones. Cada servidor que forma parte del sistema está configurado de manera idéntica, lo que significa que todos tienen la misma información y están preparados para responder a las solicitudes de los usuarios de manera uniforme.

Acceder a Grafana

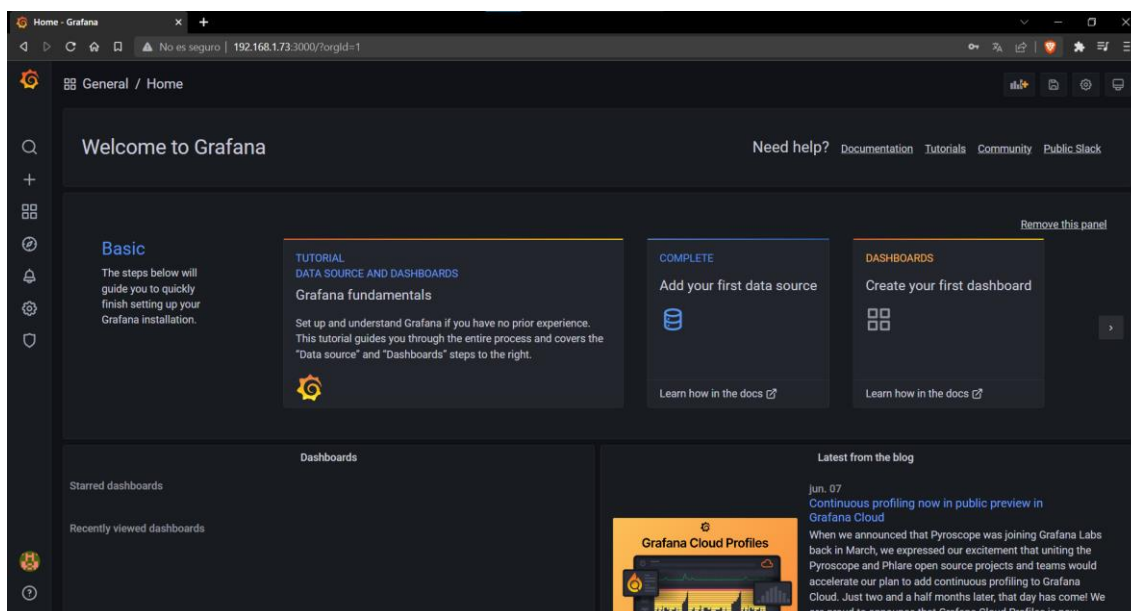
Ahora pasaremos a la unión de loki con grafana para poder visualizar los logs del servidor nginx. Para esto tendremos que ir otra vez al navegador y en este caso, tendremos que usar el puerto 3000 que está asignado a grafana.

Ilustración 17 – Grafana log in



La primera vez que nos conectemos tendremos que usar el usuario admin y contraseña admin, y tras esto, nos pedirá que cambiemos la contraseña a una nueva a nuestra elección. Al cambiarlo entraremos a la interfaz principal de grafana.

Ilustración 18 – Grafana Inicio



Una vez aquí, podremos ver las diferentes opciones que nos presenta grafana, siendo en primer lugar un apartado básico con un tutorial y las opciones de añadir una fuente de datos y crear un dashboard. Y en la barra de herramientas tendremos las siguientes opciones:

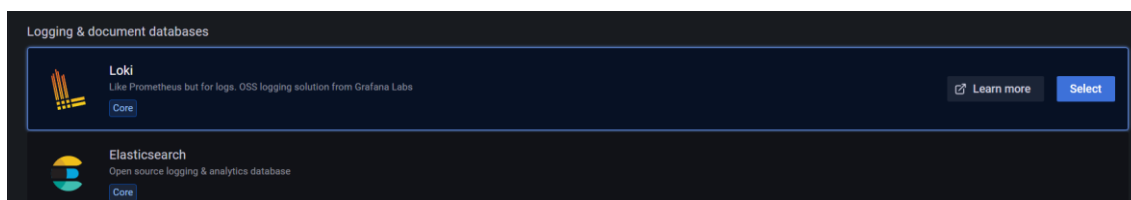
- "Home" (Inicio): Te lleva a la página de inicio de Grafana, donde puedes acceder a los paneles y explorar tus gráficos.
- "Search" (Buscar): Esta opción te permite hacer una búsqueda en todos los paneles, gráficos y fuentes de datos que hayas creado o añadido.
- "Dashboards" (Paneles): Aquí puedes crear y gestionar paneles. Puedes importar paneles predefinidos, crear nuevos paneles desde cero o modificar los existentes.
- "Data Sources" (Fuentes de datos): Esta opción te permite configurar las fuentes de datos que Grafana utilizará para recopilar información. Puedes conectar Grafana a diversas bases de datos, servicios en la nube o sistemas de monitorización.
- "Alerting" (Alertas): Aquí puedes configurar y gestionar las alertas. Puedes definir umbrales y condiciones para recibir notificaciones cuando los datos superen ciertos límites.
- "Explore": Esta opción te permite explorar y consultar tus fuentes de datos. Puedes realizar consultas y obtener resultados en forma de tablas o gráficos interactivos.
- "Configuration" (Configuración): Aquí puedes configurar los ajustes generales de Grafana, como el idioma, la autenticación, la apariencia del panel, entre otros.

- "Server Admin" (Administrador del servidor): Grafana es altamente extensible y cuenta con una amplia gama de complementos (plugins). En esta sección, puedes administrar los complementos instalados y configurar sus opciones.
- "User Profile" (Perfil de usuario): Aquí puedes gestionar tu perfil de usuario, cambiar la contraseña, editar la información personal y establecer tus preferencias individuales.
- "Help" (Ayuda): Esta opción proporciona acceso a recursos de ayuda, como la documentación oficial de Grafana, la comunidad de usuarios y el soporte técnico.

Conexión de Grafana y Loki

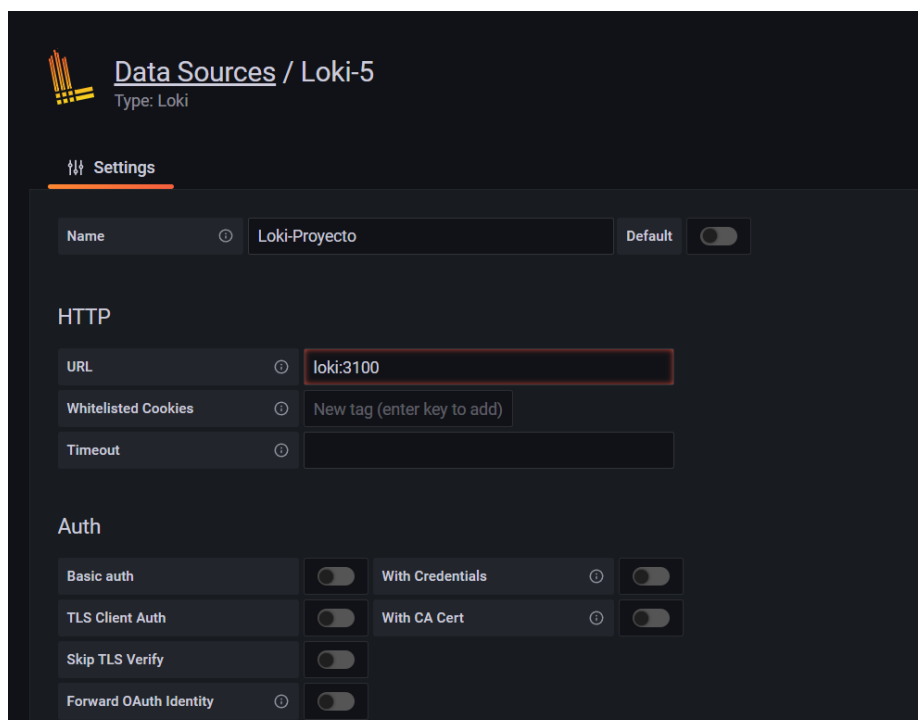
Posteriormente, para poder unir el contenedor de loki y que podamos visualizarlo en grafana, tendremos que ir a añadir una fuente de datos nueva y nos aparecerán múltiples opciones de fuentes de datos como bases de datos, logging, clouds y más variedad. En el apartado de logging nos aparecerá loki en el cual entraremos.

Ilustración 19 – Elección de fuente de datos



Una vez dentro, únicamente tendremos que asignarle un nombre y en la url le asignaremos la siguiente "loki:3100".

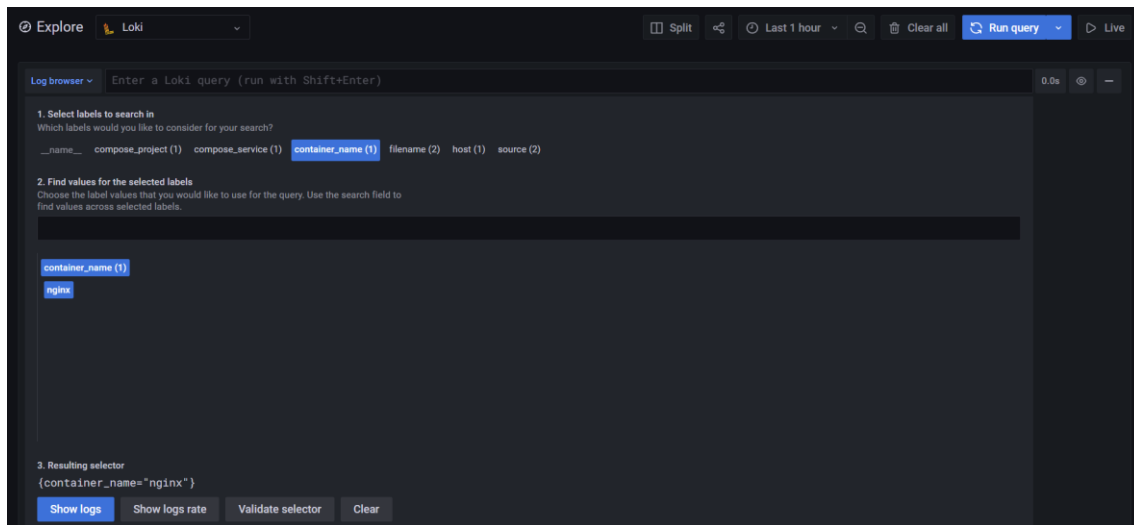
Ilustración 20 – Conexión a Loki



DOCKER

Luego, en el apartado de explorar nos aparecerá para poder buscar una consulta a partir del nombre del contenedor. Ahí nos aparecerá el contenedor de nginx que seleccionamos y al darle a mostrar logs, aparecerá un registro de estos.

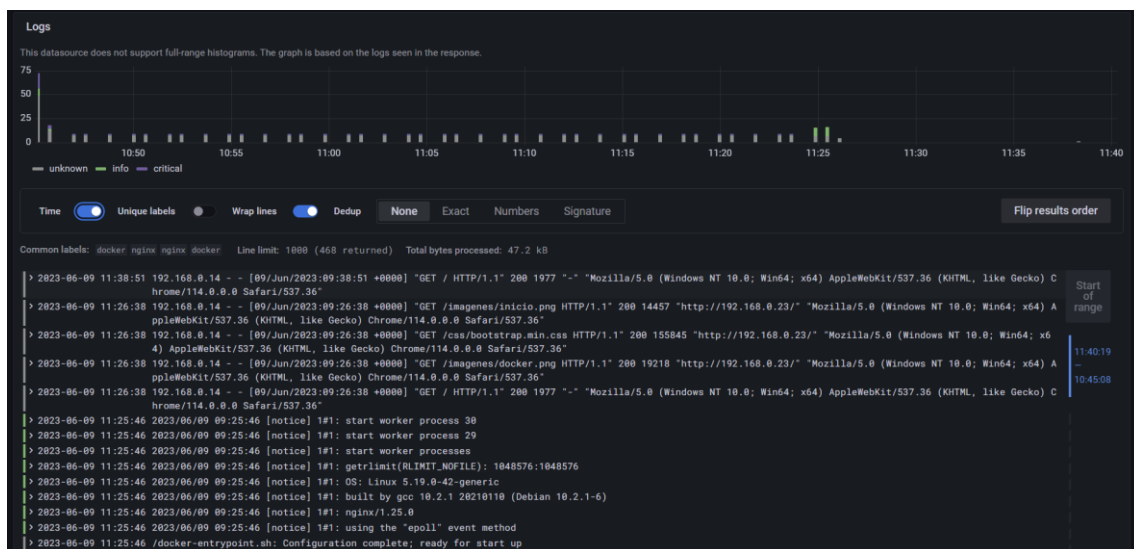
Ilustración 21 – Conectar el contenedor de nginx



Comprobación de logs

Tras esto, ya podremos visualizarlos y nos dará varias opciones para seleccionar la información que veamos necesaria de los logs.

Ilustración 22 – Comprobación de logs



En esta última imagen, se puede ver como hemos accedido desde el navegador a la ip donde tenemos montado docker.

11 - Seguimiento del proyecto

El seguimiento para un buen uso de docker implica supervisar el progreso, la calidad y el rendimiento de las aplicaciones en contenedores, así como gestionar eficientemente los recursos y las interdependencias. Al hacer un mantenimiento de docker y tener un plan para las posibles incidencias en este, podrás garantizar una implementación exitosa de tus aplicaciones, maximizar la eficiencia operativa y reducir los tiempos de desarrollo y despliegue.

Mantenimiento

Para el correcto mantenimiento de docker y de este servidor debemos tener en cuenta las siguientes pautas que nos ayudaran a un buen funcionamiento tanto del servidor como de docker:

- Actualizaciones de software: Mantén tus componentes actualizados con las últimas versiones estables de Docker, Nginx, Apache, Grafana y Loki. Esto incluye tanto el sistema operativo subyacente como las imágenes y contenedores que utilizas.
- Supervisión continua: Configura herramientas de supervisión como Grafana, que tenemos ya instalada, para controlar el rendimiento del sistema, la utilización de recursos y los posibles problemas.
- Programación de tareas: Define tareas de mantenimiento regular, como copias de seguridad, limpieza de registros antiguos y purga de contenedores y recursos no utilizados.

Incidencias

En el caso de que haya alguna incidencia, gracias a la facilidad de docker para poder volver a lanzar de nuevo los contenedores, podríamos ejecutarlo otra vez para poder seguir trabajando y ver cuál fue el error en el caso anterior.

Para solucionar los errores que podamos tener, haremos un registro y seguimiento de las que surjan tanto en el servidor como en docker, todo esto para posteriormente hacer un análisis para identificar la causa de la incidencia. Será posible gracias a la ayuda de Loki y Grafana.

Por último, buscaremos una solución adecuada y la pondremos a prueba para comprobar que vuelve a estar en correcto funcionamiento y no han aparecido unos nuevos errores.

12 - Conclusión

En conclusión, el proyecto de implementar un servidor Nginx como proxy inverso y balanceador de cargas, con dos servidores Apache y un sistema de monitoreo de registros en Docker, ofrece numerosos beneficios y oportunidades.

Mediante el uso de contenedores Docker, pudimos aprovechar las ventajas de portabilidad, eficiencia y escalabilidad que ofrece esta tecnología. Docker nos permitió encapsular y desplegar de manera eficiente los servidores Nginx y Apache, asegurando un aislamiento adecuado de las aplicaciones y un uso optimizado de los recursos del sistema operativo.

La configuración del proxy inverso y el balanceo de cargas con Nginx brinda la capacidad de distribuir la carga de trabajo de manera equitativa y mejorar la disponibilidad y el rendimiento de las aplicaciones. Esto permite una gestión eficiente del tráfico y una mejor experiencia para los usuarios finales.

Además, la implementación del sistema de monitoreo de registros en el servidor Nginx nos brinda una visibilidad valiosa sobre las solicitudes, tiempos de respuesta y posibles problemas en el flujo de tráfico. Esto facilita la detección y resolución temprana de problemas, lo que contribuye a una mejor optimización y rendimiento del sistema en general.

Debilidades

- **Complejidad en la gestión:** El uso de Docker y la configuración de un sistema de proxy inverso y balanceo de cargas puede ser complejo y requerir experiencia técnica adecuada. La gestión de múltiples contenedores y su interacción puede representar un desafío adicional.
- **Posibles problemas de rendimiento:** Si no se realiza una planificación y configuración adecuadas, es posible que se presenten problemas de rendimiento, como cuellos de botella o retrasos en el procesamiento de solicitudes. Se debe prestar especial atención a la configuración de recursos y al monitoreo continuo del sistema.

Amenazas

- **Seguridad:** Al compartir el mismo kernel del sistema operativo subyacente, existe el riesgo de que un fallo de seguridad pueda afectar a todos los contenedores. Es importante implementar medidas de seguridad adecuadas y mantener las actualizaciones del sistema operativo al día.
- **Competencia:** La implementación de servidores Nginx y Apache es una práctica común en el desarrollo de aplicaciones web. Existe la posibilidad de competencia con otras soluciones o enfoques similares en el mercado.

Fortalezas

- **Flexibilidad y escalabilidad:** El uso de Docker permite una fácil replicación y distribución de contenedores, lo que facilita la escalabilidad horizontal según las demandas del tráfico. Además, la arquitectura basada en contenedores brinda flexibilidad para ajustar los recursos asignados a cada contenedor según las necesidades.
- **Optimización de recursos:** Docker utiliza un enfoque liviano y compartido, lo que resulta en un uso eficiente de los recursos del sistema operativo subyacente. Esto maximiza la capacidad de procesamiento y minimiza los costos operativos.
- **Mayor disponibilidad:** Al utilizar un servidor Nginx como proxy inverso y balanceador de cargas, se puede mejorar la disponibilidad del sistema al distribuir el tráfico de manera equitativa entre los servidores Apache y redirigir las solicitudes de manera eficiente.

Oportunidades

- **Escalabilidad del negocio:** La capacidad de escalar horizontal y verticalmente brinda oportunidades para expandir el negocio y adaptarse a un aumento en la demanda de tráfico, lo que puede llevar a un crecimiento y éxito a largo plazo.
- **Innovación y mejora continua:** El proyecto ofrece la oportunidad de explorar nuevas tecnologías, herramientas y enfoques para mejorar el rendimiento, la seguridad y la eficiencia del sistema implementado.

Posibles mejoras

Una posible mejora sería el añadir un certificado ssl a nuestro servidor Nginx. Al implementar un certificado SSL en el servidor Nginx, se obtienen varios beneficios significativos. En primer lugar, se mejora la seguridad al proteger la comunicación entre el servidor y los usuarios finales. Esto es especialmente importante al transmitir datos confidenciales, como información personal, contraseñas o datos de pago.

Además, agregar un certificado SSL tiene un impacto positivo en la confianza del usuario. Los visitantes del sitio web verán el icono del candado en la barra de direcciones de su navegador, lo que indica que la conexión es segura y autenticada. Esto genera confianza en los usuarios y puede influir en su disposición para compartir información y realizar transacciones en el sitio.

Otro beneficio clave de utilizar SSL es el impacto positivo en el posicionamiento en los motores de búsqueda. Los principales motores de búsqueda, como Google, consideran la seguridad del sitio web como un factor de clasificación. Por lo tanto, tener un certificado SSL puede mejorar la visibilidad y el posicionamiento en los resultados de búsqueda.

Escalabilidad

Una de las ventajas clave de utilizar Docker en este proyecto es la facilidad con la que se puede escalar horizontalmente. Al utilizar contenedores en lugar de máquinas virtuales tradicionales, podemos replicar y distribuir los contenedores de manera rápida y sencilla según las necesidades del tráfico. Esto significa que podemos aumentar o disminuir la capacidad de nuestros servidores de forma dinámica y ágil.

13 - Bibliografía

SoftZone: "Docker - Descarga, instalación y uso de Docker en Windows" (2021).

URL: <https://www.softzone.es/programas/sistema/docker/>

RedesZone: "¿Qué es Docker y cómo funciona la virtualización por contenedores?" (2016).

URL: <https://www.redeszone.net/2016/02/24/docker-funciona-la-virtualizacion-contenedores/>

IONOS: "Tutorial Docker: instalación y primeros pasos" (s.f.).

URL: <https://www.ionos.es/digitalguide/servidores/configuracion/tutorial-docker-instalacion-y-primeros-pasos/>

Docker Documentation: (s.f.).

URL: <https://docs.docker.com/>

Docker Engine Documentation: (s.f.).

URL: <https://docs.docker.com/engine/>

NGINX Blog: "Deploying NGINX and NGINX Plus with Docker" (s.f.).

URL: <https://www.nginx.com/blog/deploying-nginx-nginx-plus-docker/>

Portainer Documentation: (s.f.).

URL: <https://docs.portainer.io/>

LinuxBlog: "Grafana y Loki en el mismo contenedor Docker" (s.f.).

URL: <https://linuxblog.xyz/posts/grafana-loki/>

YouTube: "Docker Tutorial for Beginners - A Full DevOps Course on How to Run Applications in Containers" por FreeCodeCamp (2017).

URL: <https://www.youtube.com/watch?v=gD1zUESRucs&t=97s>