

## 7. Instalación de Grafana

Para visualizar métricas recolectadas por Prometheus utilizando Docker Compose para desplegar un contenedor de Grafana, vamos a realizar los siguientes pasos:

### 1. Creamos el archivo docker-compose.yml en la versión 3.8

```
Sprint-1 > Ejercicio-7 > 📄 docker-compose.yml
 1  version: "3.8"
 2    ▷ Run All Services
 3  services:
 4    ▷ Run Service
 5    prometheus:
 6      image: prom/prometheus:latest
 7      container_name: prometheus
 8      ports:
 9        - "9090:9090"
10      volumes:
11        - ./prometheus.yml:/etc/prometheus/prometheus.yml
12      networks:
13        - monitoring
14
15    ▷ Run Service
16    cAdvisor:
17      image: gcr.io/cadvisor/cadvisor:latest
18      container_name: cAdvisor
19      ports:
20        - "8080:8080"
21      volumes:
22        - /:/rootfs:ro
23        - /var/run:/var/run:rw
24        - /sys:/sys:ro
25        - /dev/disk/:/dev/disk:ro
26      networks:
27        - monitoring
28
29    ▷ Run Service
30    grafana:
31      image: grafana/grafana:latest
32      container_name: grafana
33      ports:
34        - "3000:3000"
35      volumes:
36        - grafana_data:/var/lib/grafana
37      networks:
38        - monitoring
39
40      volumes:
41        - grafana_data:
42
43    networks:
44      monitoring:
45        driver: bridge
```

Vamos a ir explicando paso a paso el código del archivo:

- Vamos a ejecutar 3 servicios: prometheus, cadvisor y grafana.

- **Prometheus:**

- Usa la imagen más reciente de prometheus.
- El nombre del contenedor es prometheus.
- Se expone el puerto 9090 del contenedor al puerto 9090 de la máquina local.
- Usa un volumen `./prometheus.yml:/etc/prometheus/prometheus.yml`: monta el archivo `prometheus.yml` en la ruta de configuración `prometheus` dentro del contenedor.
- Está conectada a una red llamada monitoring que permite que todos los contenedores conectados a esa misma red se comuniquen entre sí.

- **cAdvisor:**

- Usa la imagen Docker de cAdvisor.
- El nombre del contenedor es cadvisor.
- Se expone el puerto 8080 del contenedor al puerto 8080 de la máquina local.
- Usa varios volúmenes del sistema en modo solo lectura para que cAdvisor pueda acceder a información CPU, red, etc.
- Está conectada a una red llamada monitoring que permite que todos los contenedores conectados a esa misma red se comuniquen entre sí.

- **Grafana:**

- Usa la imagen más reciente de grafana.
- El nombre del contenedor es grafana.
- Se expone el puerto 3000 del contenedor al puerto 3000 de la máquina local.
- Usa un volumen persistente llamado `grafana-data` montado en la ruta `/var/lib/grafana`
- Está conectada a una red llamada monitoring que permite que todos los contenedores conectados a esa misma red se comuniquen entre sí.

- Definimos el volumen `grafana-data`.
- Definimos la red con el nombre `monitoring`, dónde se conectarán los contenedores entre sí mediante un puente (bridge).

## 2. Creamos el archivo `prometheus.yml`

```
Sprint-1 > Ejercicio-7 > ! prometheus.yml
  1 global:
  2   |   scrape_interval: 15s
  3
  4   scrape_configs:
  5     - job_name: 'prometheus'
  6       static_configs:
  7         - targets:
  8           |           - 'prometheus:9090'
  9
 10      - job_name: 'cadvisor'
 11        static_configs:
 12          - targets:
 13            |           - 'cadvisor:8080'
```

Vamos a ir explicando paso a paso el código del archivo:

- En `global` (ajustes generales) se define:
  - `scrape_interval: 15` → Establece que prometheus recolecte métricas cada 15 segundos de los servicios definidos.
- En `scrape_configs` definimos 2 servicios de los cuales prometheus obtendrá los datos:

- **Servicio prometheus:** Prometheus va a recoger métricas de sí mismo a través del puerto 9090.
- **Servicio cAdvisor:** Prometheus va a recoger métricas del contenedor cAdvisor en el puerto 8080.

### 3. Comprobamos que los servicios estén funcionando

Tras crear todos los archivos, vamos a comprobar que los servicios se puedan lanzar con el comando:

```
docker compose up -d
```

```
● anabel@ubuntu:~/Sprint-1/Ejercicio-7$ docker compose up -d
WARN[0000] /home/anabel/Sprint-1/Ejercicio-7/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 4/4
 ✓ Network ejercicio-7_monitoring Created
 ✓ Container prometheus Started
 ✓ Container cadvisor Started
 ✓ Container grafana Started
 0.1s
 0.5s
 0.5s
 0.5s
○ anabel@ubuntu:~/Sprint-1/Ejercicio-7$
```

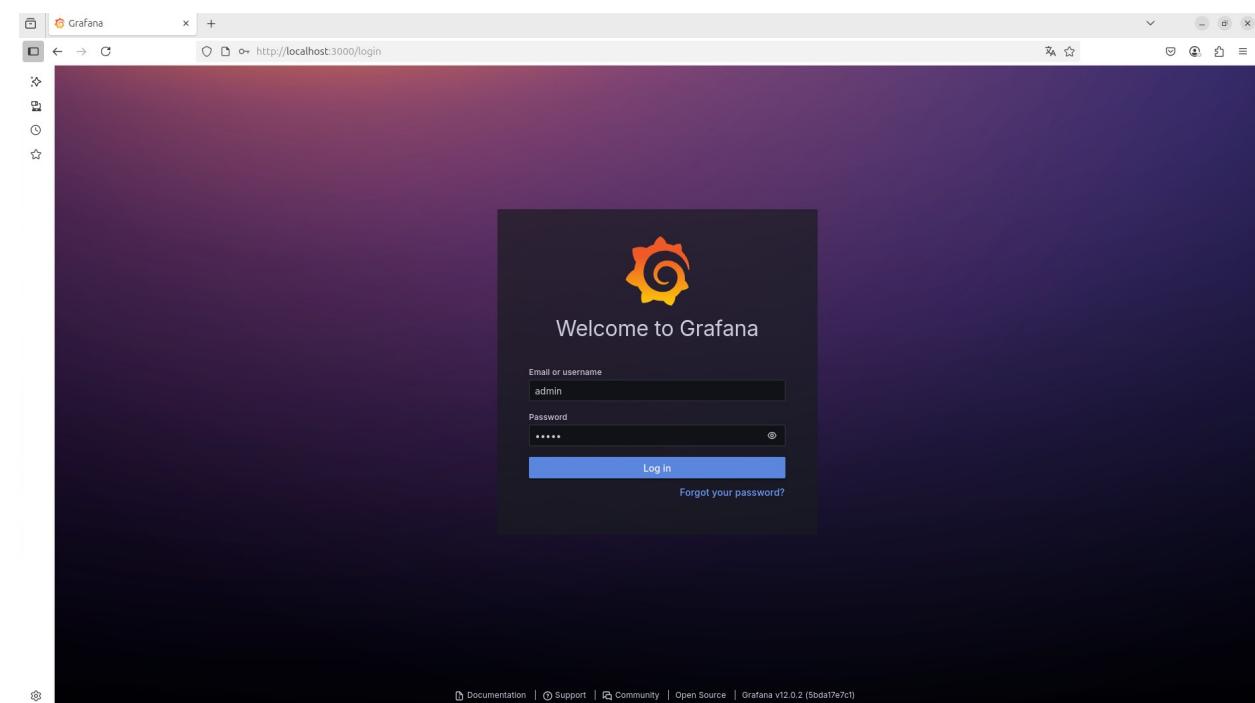
Tras lanzar los servicios, vamos a verificar que los contenedores estén en ejecución con el siguiente comando:

```
docker compose ps
```

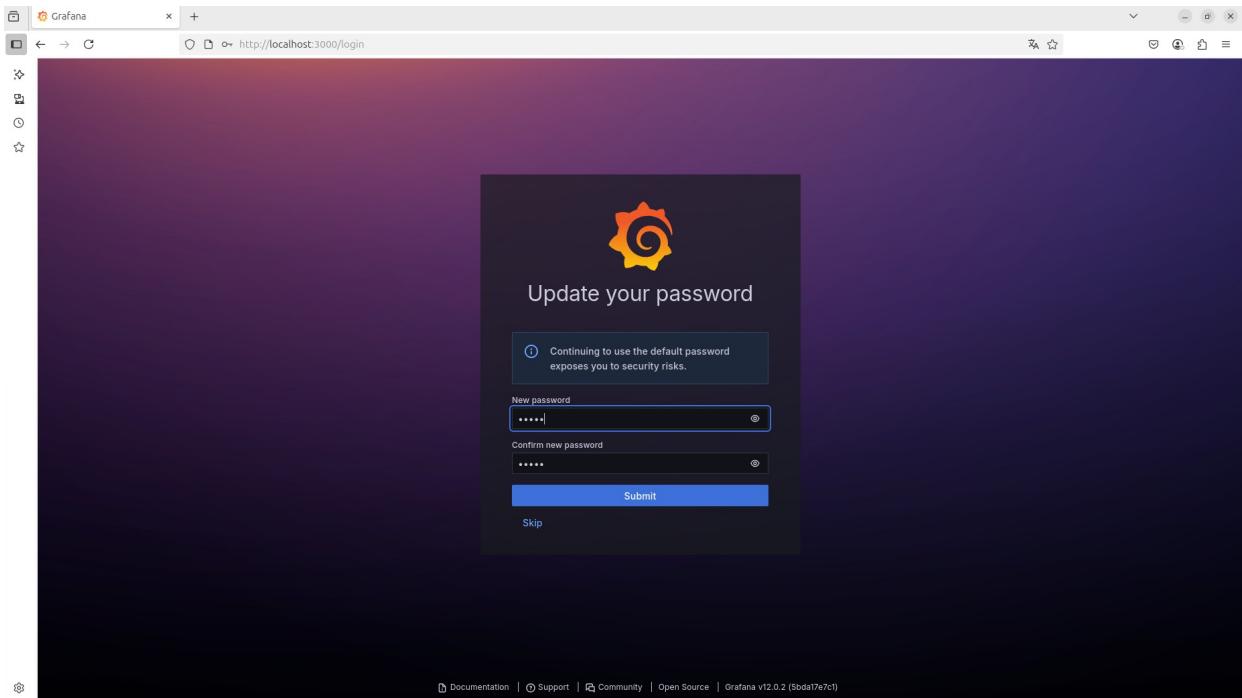
```
● anabel@ubuntu:~/Sprint-1/Ejercicio-7$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
fe6493b6b912 grafana/grafana:latest "/run.sh" About a minute ago Up About a minute 0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
grafana
211da170aac8 prom/prometheus:latest "/bin/prometheus --c..." About a minute ago Up About a minute 0.0.0.0:9090->9090/tcp, :::9090->9090/tcp
prometheus
623928a8fc72 gcr.io/cadvisor/cadvisor:latest "/usr/bin/cadvisor ..." About a minute ago Up About a minute (healthy) 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
cadvisor
○ anabel@ubuntu:~/Sprint-1/Ejercicio-7$
```

### 4. Comprobamos que la aplicación Grafana funciona

Abrimos la dirección <http://localhost:3000> desde el navegador. Introducimos el usuario ‘admin’ y la contraseña ‘admin’.



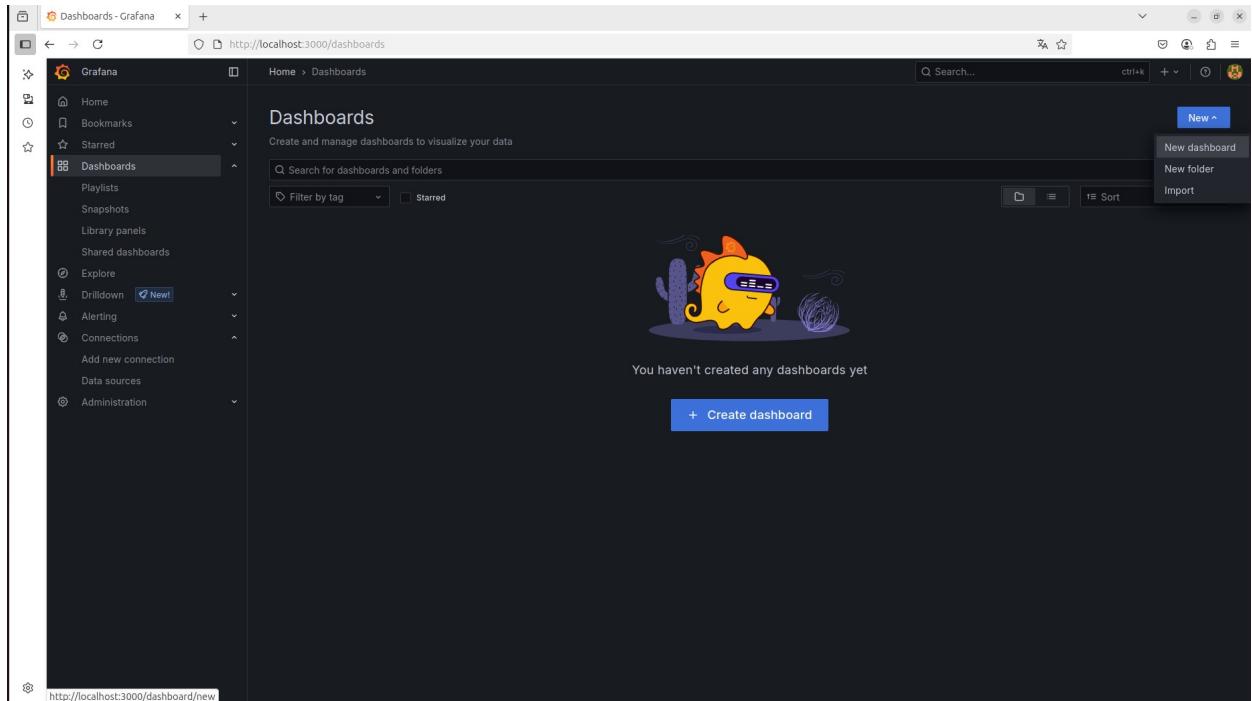
Tras esto, nos va a pedir que creamos una nueva contraseña, lo hacemos y ya podremos entrar en Grafana.



Ahora, vamos a irnos a la opción *Connections* y abrimos *Data Sources*. Pulsamos *Add data sources*, elegimos *Prometheus* y en la parte de *Prometheus server URL* introducimos <http://prometheus:9090>. Tras esto, guardamos y ya estará nuestro servicio prometheus funcionando.

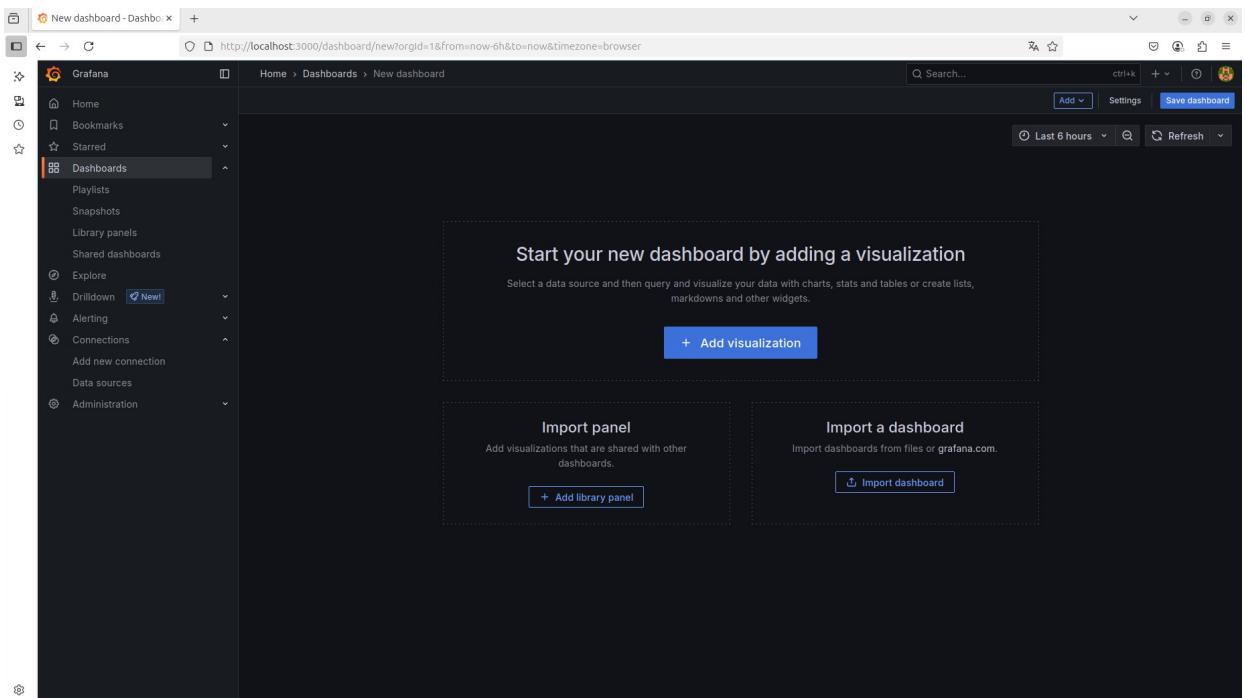
The screenshot shows the 'Data sources' configuration page for 'prometheus-1'. The 'Type' is set to 'Prometheus'. The 'Name' field contains 'prometheus-1'. Under 'Connection', the 'Prometheus server URL' is set to 'http://prometheus:9090'. Under 'Authentication', 'No Authentication' is selected. Under 'TLS settings', there is an option to 'Add self-signed certificate'. A note at the top says: 'Before you can use the Prometheus data source, you must configure it below or in the config file. For detailed instructions, view the documentation.' Fields marked with \* are required.

Nos vamos ahora al apartado de *Dashboard*, pulsamos *New* y *New dashboard*.



The screenshot shows the Grafana interface with the title bar "Dashboards - Grafana". The left sidebar is open, showing categories like Home, Bookmarks, Starred, and Dashboards. Under Dashboards, there are sub-options: Playlists, Snapshots, Library panels, Shared dashboards, Explore, Drilldown (which has a "New!" badge), Alerting, Connections, Add new connection, Data sources, and Administration. The main content area is titled "Dashboards" with the sub-instruction "Create and manage dashboards to visualize your data". It features a search bar "Search for dashboards and folders", a filter "Filter by tag", and a "Starred" checkbox. A cartoon character is displayed in the center. Below the character, the message "You haven't created any dashboards yet" is shown, followed by a prominent blue "Create dashboard" button.

Pulsamos *Add visualization* y en *Data Source* seleccionamos la que hemos creado anteriormente, **prometheus-1**.



The screenshot shows the "New dashboard" creation page. The title bar is "New dashboard - Dashboards". The left sidebar is identical to the previous screenshot. The main area is titled "Start your new dashboard by adding a visualization" with the instruction "Select a data source and then query and visualize your data with charts, stats and tables or create lists, markdowns and other widgets.". Below this is a large blue "Add visualization" button. At the bottom of the main area, there are two sections: "Import panel" (with a "+ Add library panel" button) and "Import a dashboard" (with a "Import dashboard" button). The top right of the main area includes "Add", "Settings", and "Save dashboard" buttons, along with a time range selector "Last 6 hours" and refresh controls.

Definimos las consultas:

The image displays four separate instances of the Prometheus Query Editor interface, labeled A, B, C, and D, each showing a query builder for a specific metric.

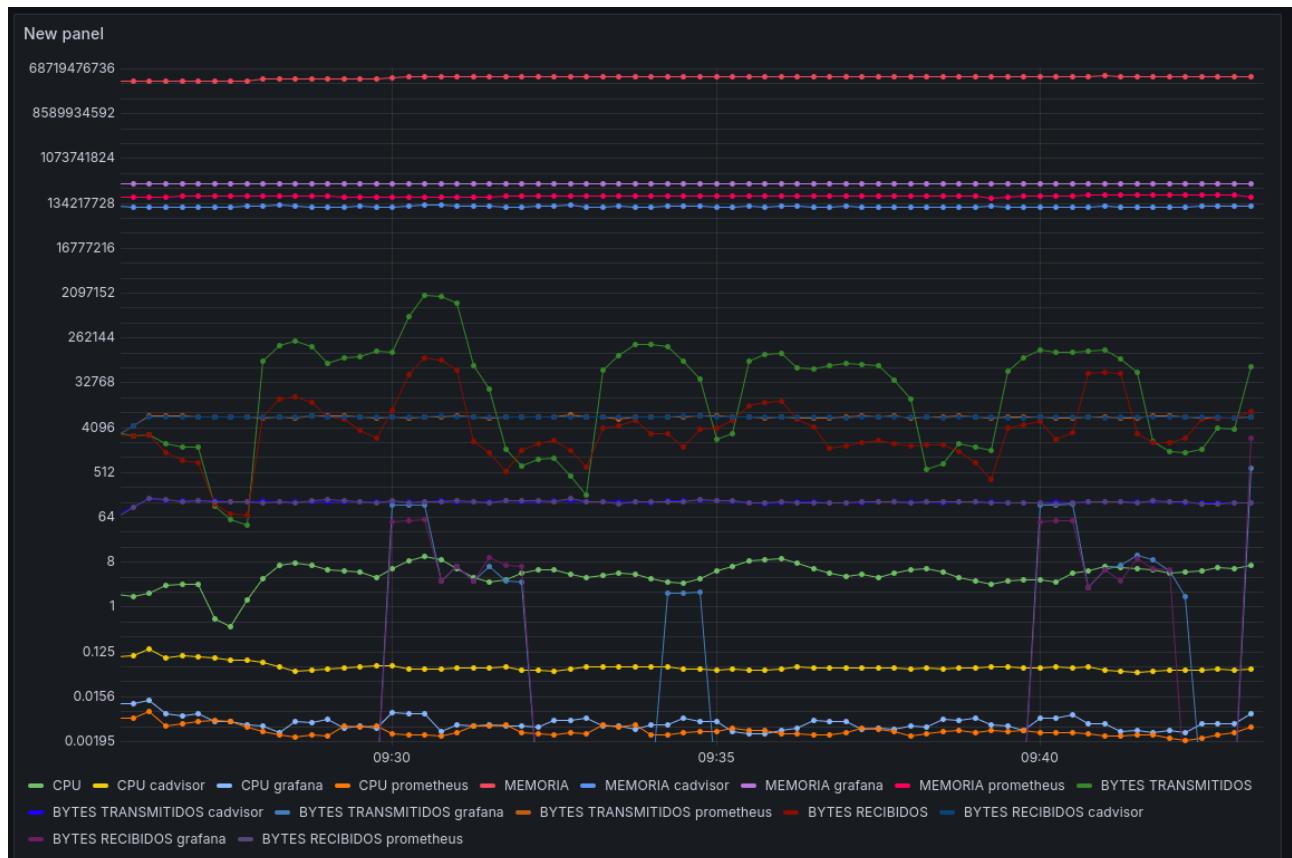
- Query A (CPU Usage):** `sum by(container_label_com_docker_compose_service) (rate(container_cpu_usage_seconds_total[$__rate_interval]))`
- Query B (Memory Usage):** `sum by(container_label_com_docker_compose_service) (container_memory_usage_bytes)`
- Query C (Network Receive Bytes):** `sum by(container_label_com_docker_compose_service) (rate(container_network_receive_bytes_total[$__rate_interval]))`
- Query D (Network Transmit Bytes):** `sum by(container_label_com_docker_compose_service) (rate(container_network_transmit_bytes_total[$__rate_interval]))`

- **Consulta A:** `sum by(container_label_com_docker_compose_service) (rate(container_cpu_usage_seconds_total[$__rate_interval]))`  
→ Muestra cuánta CPU está usando cada servicio en tiempo real.

→ En la leyenda se ha modificado para que aparezca como :  
 CPU {{container\_label\_com\_docker\_compose\_service}}

- **Consulta B:** sum by(container\_label\_com\_docker\_compose\_service)  
 (container\_memory\_usage\_bytes)
  - Muestra cuántos bytes de memoria está usando cada servicio.
  - En la leyenda se ha modificado para que aparezca como :
 MEMORIA {{container\_label\_com\_docker\_compose\_service}}
- **Consulta C:** sum by (container\_label\_com\_docker\_compose\_service)  
 (rate(container\_network\_receive\_bytes\_total[\$\_\_rate\_interval]))
  - Muestra cuántos bytes de entrada ha recibido la red en tiempo real por cada servicio.
  - En la leyenda se ha modificado para que aparezca como :
 BYTES RECIBIDOS {{container\_label\_com\_docker\_compose\_service}}
- **Consulta D:** sum by (container\_label\_com\_docker\_compose\_service)  
 (rate(container\_network\_transmit\_bytes\_total[\$\_\_rate\_interval]))
  - Muestra cuántos bytes de salida ha transmitido la red en tiempo real por cada servicio.
  - En la leyenda se ha modificado para que aparezca como :
 BYTES TRANSMITIDOS {{container\_label\_com\_docker\_compose\_service}}

Tras realizar las consultas, pulsamos *Run queries* y se nos muestra el gráfico con los resultados. Después, pulsamos *Save* para guardar el dashboard creado.



En este gráfico se ha usado la escala logarítmica para poder visualizar todos los servicios, ya que cada uno maneja rangos de valores muy diferentes: el uso de CPU es mucho más pequeño en comparación con el consumo de la memoria o el tráfico de red. De esta manera, podemos ver los datos obtenidos de la CPU, la memoria y el tráfico de red tanto de Prometheus, cAdvisor y Grafana como de la métrica total de todos.