

**Universidad del Quindío**

**Facultad de Ingeniería**

**Programa de Ingeniería Electrónica**

**Daniel Alejandro Quintero  
Carlos Andrés Rendón Soto  
Sebastián Pardo Ocampo  
Valentina Garibello Lizarazo**

**Avance 1 - Seminario de grado: Despliegue y monitorización de servicios en  
redes y kubernetes**

**Armenia, agosto del 2024**

## Introducción

Las tecnologías de comunicaciones, redes y servicios a lo largo de los años se ha venido estableciendo como una de las tecnologías más importantes para el despliegue del mundo moderno, es por eso que se ha incrementado la demanda de los mismos y ha surgido la necesidad de buscar alternativas que brinden una mayor calidad de servicio a los usuarios y una mejor gestión de recursos para los proveedores, es por eso que se han venido desarrollando avances como la virtualización de funciones de red. Esto implica implementar funciones de manera virtual y computarizada que antes se desempeñaban de manera física e incluso analógica con el fin de optimizar y balancear recursos de hardware y facilitar la administración y gestión de las comunicaciones [1].

Existen diversas opciones y herramientas para desarrollar la virtualización de funciones de red y la creación de redes definidas por software como los contenedores, los cuales se someten a procesos de gestión y orquestación con el fin de elaborar una estructura de red eficaz. Se presentan algunas definiciones y términos clave para el desarrollo de esta investigación.

### Definiciones:

**Docker:** Es una tecnología que facilita el desarrollo, despliegue y gestión de aplicaciones que se ejecutan de manera aislada dentro de contenedores. Los contenedores son entornos virtualizados que permiten que una aplicación se ejecute de manera independiente del sistema operativo subyacente.

Docker facilita la creación de aplicaciones encapsulando en un solo contenedor todos los recursos necesarios para su funcionamiento. Esto incluye el código fuente de la aplicación, todas sus dependencias y configuraciones. Al estar aislada, la aplicación no requiere un sistema operativo completo para ejecutarse, lo que simplifica su despliegue en cualquier entorno informático que soporte Docker. Gracias a esta independencia del sistema operativo, los contenedores pueden desplegarse tanto localmente como en la nube.

La tecnología de contenedores en Docker ofrece una gran portabilidad, permitiendo el despliegue y ejecución de aplicaciones sin preocuparse por los requisitos específicos del sistema operativo o del hardware subyacente, ya que todo lo necesario está contenido dentro del propio contenedor. Esto lo hace muy eficiente y adaptable para ser utilizado en una amplia variedad de entornos.

Las principales ventajas de utilizar Docker incluyen la capacidad de ejecutar aplicaciones en cualquier máquina sin restricciones, lo que facilita un entorno de desarrollo portátil, seguro y limpio. Además, Docker permite la automatización de pruebas, empaquetado e integración continua. La posibilidad de empaquetar todas las dependencias junto con la aplicación

garantiza su ejecución en cualquier plataforma que soporte Docker, haciendo que el despliegue sea rápido, repetible, escalable y portátil. [2]

**Docker compose:** Docker Compose es una herramienta que se puede utilizar desde la línea de comandos o a través de un archivo de configuración, para facilitar el despliegue de aplicaciones. Permite a los usuarios definir, configurar y desplegar aplicaciones que pueden consistir en uno o más contenedores, utilizando un enfoque sencillo y claro.

Para gestionar y organizar estos despliegues, Docker Compose utiliza archivos de configuración en formato YAML. En estos archivos se definen los servicios que compondrán la aplicación, los volúmenes que se utilizarán, las redes a las que se conectarán y todas las configuraciones necesarias para la ejecución de la aplicación.

Una de las grandes ventajas de Docker Compose es que, una vez configurado el archivo YAML, solo es necesario ejecutar unos pocos comandos para crear y ejecutar los servicios definidos, simplificando significativamente el proceso de despliegue. [2]

**Imágenes:** Las imágenes contienen un conjunto de archivos que permiten crear un sistema operativo o aplicación específica. Estas imágenes son esenciales para la creación de contenedores en Docker, ya que un contenedor se basa en una imagen preexistente.

En términos simples, las imágenes funcionan como plantillas que definen el estado base o inicial de un sistema. A partir de una imagen, se puede crear un contenedor que incluye los archivos raíz, dependencias y configuraciones necesarias para la ejecución de la aplicación o servicio deseado. Esta estructura permite que las aplicaciones sean desplegadas de manera eficiente y consistente en diferentes entornos. [2]

### **Contenedor Docker:**

Un contenedor Docker se crea a partir de una imagen y se puede definir como una instancia de una aplicación que se ejecuta de manera aislada de otras aplicaciones y del sistema operativo que se ejecutan en la máquina. Al sistema que ejecuta contenedores también se le denomina "host de Docker".

Los contenedores permiten ejecutar múltiples procesos simultáneamente. Sin embargo, es considerado más eficiente ejecutar un solo proceso por contenedor para mantener la simplicidad, escalabilidad y la eficiencia en la gestión de los recursos. [2]

### **FFMPEG:**

FFmpeg puede decodificar, codificar, transcodificar, muxear, demuxear, transmitir, filtrar y reproducir una amplia gama de formatos, desde los más antiguos hasta los más modernos, una solución multimedia completa. Es compatible con una variedad de sistemas operativos, como Linux, Mac OS X y Windows, y funciona con una variedad de entornos y arquitecturas de máquina.

Las bibliotecas clave como "libavcodec", "libavutil", "libavformat", "libavfilter", "libavdevice" y "libswscale" están incluidas en el proyecto y pueden ser utilizadas por aplicaciones. Además, incluye herramientas como ffmpeg, ffplay y ffprobe para transcodificación y reproducción. [3]

## Objetivos

- Establecer una red conformada por 2 computadores punto a punto con el fin de ofrecer y acceder a un servicio mediante contenedores, donde un computador despliega un servicio en un contenedor de docker (en este caso un servidor nginx) y otro computador es un cliente que envía peticiones a dicho servidor.
- Monitorizar el tráfico y consumo de los servicios dockerizados desplegados sobre una red física con las herramientas Prometheus y Grafana.

## Desarrollo

### Configuración de la red local

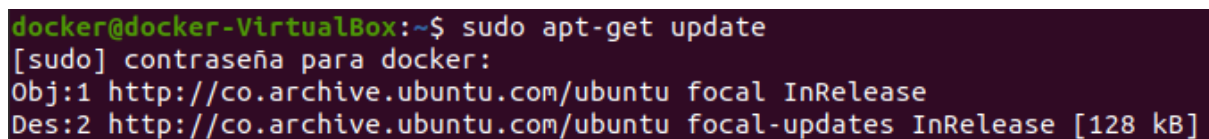
En primer lugar se realiza la configuración para la red física punto a punto, conectando un cable cruzado de un computador a otro y configurando en cada uno la misma dirección IP para establecer una red local donde los dispositivos comparten puerta de enlace.

### Configuración del servidor

El dispositivo será un servidor web nginx que transmite video desde un contenedor de Docker.

El siguiente comando actualiza la base de datos de los repositorios de software que hay en el sistema.

```
sudo apt-get update
```



```
docker@docker-VirtualBox:~$ sudo apt-get update
[sudo] contraseña para docker:
Obj:1 http://co.archive.ubuntu.com/ubuntu focal InRelease
Des:2 http://co.archive.ubuntu.com/ubuntu focal-updates InRelease [128 kB]
```

Figura 1: Actualización de recursos de Linux

Para la instalación del software curl encargado de la transferencia de los archivos a través de una URL requerida para la descarga de archivos de docker.

```
sudo apt-get install ca-certificates curl
```

```

docker@docker-VirtualBox:~$ sudo apt-get install ca-certificates curl
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  libcurl4
Se instalarán los siguientes paquetes NUEVOS:
  curl
Se actualizarán los siguientes paquetes:
  ca-certificates libcurl4
2 actualizados, 1 nuevos se instalarán, 0 para eliminar y 347 no actualizados.
Se necesita descargar 161 kB/549 kB de archivos.
Se utilizarán 440 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s

```

Figura 2: Instalación de curl

Con este comando se crea un directorio en la ubicación específica (/etc/apt/keyrings) con permisos de escritura lectura y ejecución.

```
sudo install -m 0755 -d /etc/apt/keyrings
```

```
docker@docker-VirtualBox:~$ sudo install -m 0755 -d /etc/apt/keyrings
```

Figura 3: comando de transferencia de archivos

El siguiente comando se encarga de transferir los archivos de la URL a la dirección específica (/etc/apt/keyrings/docker.asc)

```
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
```

```
docker@docker-VirtualBox:~$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
```

Figura 4: Comando para traspaso de archivos desde URL hasta directorio

Luego, se concede a todos los usuarios del sistema (propietario, grupo y otros) el permiso de lectura sobre el archivo docker.asc a través del siguiente comando:

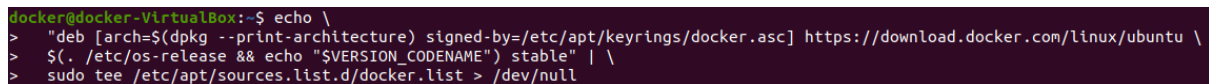
```
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

```
docker@docker-VirtualBox:~$ sudo chmod a+r /etc/apt/keyrings/docker.asc
```

Figura 5: Comando para concesión de permisos

El siguiente comando modifica un archivo de texto con las características necesarias para instalar docker.

```
echo \  
  "deb [arch=$(dpkg --print-architecture)  
signed-by=/etc/apt/keyrings/docker.asc]  
https://download.docker.com/linux/ubuntu \  
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \  
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```



```
docker@docker-VirtualBox:~$ echo \  
>  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \  
>  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \  
>  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

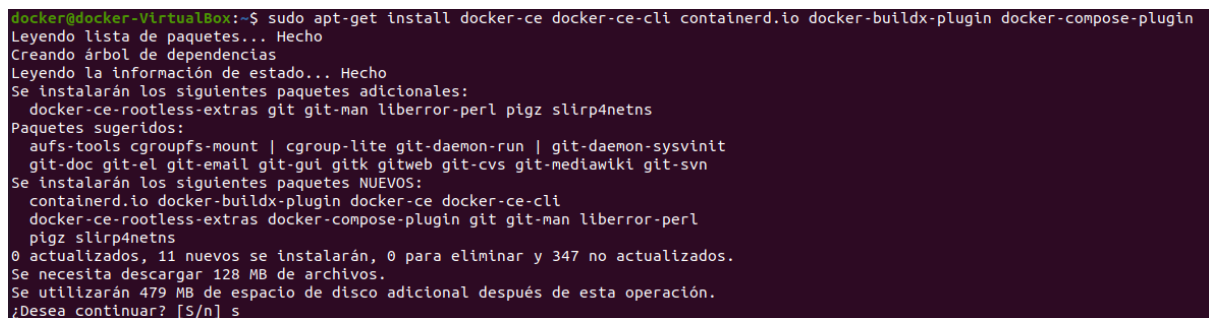
Figura 6: Comando para preparación de instalación de Docker

Se actualiza la base de datos de los repositorios.

```
sudo apt-get update
```

El siguiente comando se encarga de instalar Docker Engine como también otras aplicaciones que incluyen docker compose que más adelante se emplearán.

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



```
docker@docker-VirtualBox:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Se instalarán los siguientes paquetes adicionales:  
  docker-ce-rootless-extras git git-man liberror-perl pigz slirp4netns  
Paquetes sugeridos:  
  aufs-tools cgroupfs-mount | cgroup-lite git-daemon-run | git-daemon-sysvinit  
  git-doc git-el git-email git-gui gitk gitweb git-cvs git-mediawiki git-svn  
Se instalarán los siguientes paquetes NUEVOS:  
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli  
  docker-ce-rootless-extras docker-compose-plugin git git-man liberror-perl  
  pigz slirp4netns  
0 actualizados, 11 nuevos se instalarán, 0 para eliminar y 347 no actualizados.  
Se necesita descargar 128 MB de archivos.  
Se utilizarán 479 MB de espacio de disco adicional después de esta operación.  
¿Desea continuar? [S/n] s
```

Figura 7: Comando para instalación de Docker

Una vez instalado el gestor de contenedores Docker, se debe crear una imagen, con base a la que se levantará el contenedor nginx para el servidor:

En este caso se crea un documento de texto con el nombre “Dockerfile” [5]

```
1 FROM tiangolo/nginx-rtmp
2
3 RUN apt-get update && apt-get -y upgrade
4
5 RUN apt-get install -y ffmpeg
6
7 RUN apt-get install -y net-tools
8
9 RUN apt-get install -y iputils-ping
10
11 ADD nginx.conf /etc/nginx
12
13 EXPOSE 1935
14
15 ADD Video.mp4 /root
16
17
18
```

Figura 8: Declaración del Dockerfile

Este comando crea la imagen en un Dockerfile basado en `tiangolo/nginx-rtmp` dicha imagen está creada en el sistema operativo Debian con el servidor web nginx y los protocolos de videostream rtmp [6].

```
FROM tiangolo/nginx-rtmp
```

Ahora, se ejecuta como primera medida la actualización de repositorios (`apt-get update`) y la actualización del sistema (`apt-get -y upgrade`) el `-y` le da los permisos al contenedor para la instalación:

```
RUN apt-get update && apt-get -y upgrade.
```

Este comando instala `ffmpeg` encargado de la transmisión del video aplicando el protocolo `rtmp`.

```
RUN apt-get install -y ffmpeg
```

Después se instalan y permiten conocer las configuraciones de red como realizar ping entre contenedor a cliente necesario para comprobar la conectividad.

```
RUN apt-get install -y net-tools
RUN apt-get install -y iputils-ping
```

En la Figura 9 está el código de configuración para nginx principalmente se requiere crear un documento de texto con el nombre “`nginx.conf`”, que es el encargado de aplicar las configuraciones que requiere nginx para funcionar correctamente [7].

```

1 worker_processes auto;
2 rtmp_auto_push on;
3 events {}
4 rtmp {
5     server {
6         listen 1935;
7         listen [::]:1935 ipv6only=on;
8
9         application live {
10             live on;
11             record off;
12         }
13     }
14 }

```

Figura 9: Configuración del servidor nginx

Este comando transfiere el archivo anteriormente mencionado (nginx.conf) a la ubicación /etc/nginx en el contenedor para que funcione el documento de dockerfile debe estar en la misma carpeta que el archivo (nginx.conf)

ADD nginx.conf /etc/nginx

Se asigna el puerto 1935 a la imagen del servidor:

EXPOSE 1935

Y se agrega el video descargado a la carpeta raíz del contenedor:

ADD video.mp4/root

Este comando se encarga de crear la imagen con todas las configuraciones aplicadas en el dockerfile [8].

sudo docker build -t servidor:latest -f Dockerfile .

```

root@docker:~/VirtualBox/~/Escritorio/Servidor$ sudo docker build -t servidor:latest -f Dockerfile .
[+] Building 98.5s (12/12) FINISHED
=> [internal] load build definition from Dockerfile
=> transferring dockerfile: 272B
=> [internal] load metadata for docker.io/tiangolo/nginx-rtmp:latest
=> [internal] load .dockerignore
=> transferring context: 2B
[1/7] FROM docker.io/tiangolo/nginx-rtmp:latest@sha256:684159f2ceb0126a0fa95c83ad0f5d20cd795ada5709f9b51d5803da1acd9b9
=> resolve docker.io/tiangolo/nginx-rtmp:latest@sha256:684159f2ceb0126a0fa95c83ad0f5d20cd795ada5709f9b51d5803da1acd9b9
=> sha256:ba83b8fca943648a883d1484b33faadf5e96a99a2b683e3bbaee912bca845 55.08MB / 55.08MB
=> sha256:4de77900edd2092d13d5c9e9cd2ef6f77463222402f7b02c10ba21f448cc 15.76MB / 15.76MB
=> sha256:6d91d1dfdd17d0b0cbb4772eb085d97e02504d89ea3e5857cb97c943b74462 54.73MB / 54.73MB
=> sha256:684159f2ceb0126a0fa95c83ad0f5d20cd795ada5709f9b51d5803da1acd9b9 1.61kB / 1.61kB
=> sha256:033a96d4c16128d11790ac7b55df2b51d57268008e9265f58fa8cbf5a298fc 2.20kB / 2.20kB
=> sha256:d20dc8b527b97f2624b87f74b0b19c23bec239dec03126d6e73f3b09d14423 5.87kB / 5.87kB
=> extracting sha256:ba83b8fca943648a883d1484b33faadf5e96a99a2b683e3bbaee912bca845 4.6s
=> sha256:54cf3060466d9d2bea930c0ee58dc927e0da651b862491af1648a4aca73 197.07MB / 197.07MB
=> sha256:716e0366f2a1a185dc0e8a129abfe0c23ba8d2a113da6293d902b4552a223ef 2.44kB / 2.44kB
=> sha256:a09c8c0baf4914313a10ealadcf02a1ee9c379709a49d95b42dbaa1808be955e 2.22MB / 2.22MB
=> sha256:b3ca71dc14c4e60f8c72d0ef07709273bf730977070309f59a6d34b785aaf 1.94MB / 1.94MB
=> sha256:b329ce11c076eeb519d5c52e7021de19714b9b9a80bb71864e1e8d1d58d709ce 2.57MB / 2.57MB
=> sha256:0dca6332cdfd96051c936da654d870edadcfe6a6f1d2b820ed7527adf5c 199B / 199B
=> sha256:0f388b5895d465d0227178fc72d64bdf5bf542b955e0bf7ad2f411a8b5c014e 287B / 287B
=> extracting sha256:4de77900edd2092d13d5c9e9cd2ef6f77463222402f7b02c10ba21f448cc 0.0s
=> extracting sha256:6d91d1dfdd17d0b0cbb4772eb085d97e02504d89ea3e5857cb97c943b74462 3.2s
=> extracting sha256:54cf3060466d9d2bea930c0ee58dc927e0da651b862491af1648a4aca73 8.5s
=> extracting sha256:716e0366f2a1a185dc0e8a129abfe0c23ba8d2a113da6293d902b4552a223ef 0.0s
=> extracting sha256:a09c8c0baf4914313a10ealadcf02a1ee9c379709a49d95b42dbaa1808be955e 0.2s
=> extracting sha256:b3ca71dc14c4e60f8c72d0ef07709273bf730977070309f59a6d34b785aaf 0.1s
=> extracting sha256:b329ce11c076eeb519d5c52e7021de19714b9b9a80bb71864e1e8d1d58d709ce 0.1s
=> extracting sha256:0dca6332cdfd96051c936da654d870edadcfe6a6f1d2b820ed7527adf5c 0.0s
=> extracting sha256:0f388b5895d465d0227178fc72d64bdf5bf542b955e0bf7ad2f411a8b5c014e 0.0s
=> [internal] load build context
=> transferring context: 7.93MB
[2/7] RUN apt-get update && apt-get -y upgrade
[3/7] RUN apt-get install -y ffmpeg
[4/7] RUN apt-get install -y net-tools
[5/7] RUN apt-get install -y iputils-ping
[6/7] ADD nginx.conf /etc/nginx
[7/7] ADD Video.mp4 /root
=> exporting layers
=> exporting layers
=> writing image sha256:c07aeb0a3d9523ca19b97390ca52807c3c5b731b52bf5bf6f2f2d04ed93a71c
=> naming to docker.io/library/servidor:latest

```



Figura 10: Resultado en consola de la construcción de la imagen de Docker

Ahora, para la monitorización de la métricas de consumo en el interior del servidor, en la siguiente figura se aprecian los archivos necesarios para el funcionamiento del servidor, es requerido que todos estén en la misma carpeta para que no ocurran errores.

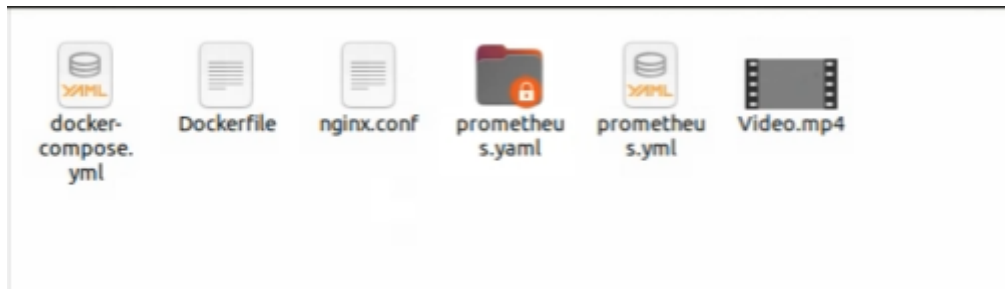


Figura 11: Elementos necesarios para la creación del servidor

Este código basado en el formato de docker compose se crea en un editor de texto el cual es necesario nombrarlo “docker-compose.yml”, donde se asigna la configuración de Prometheus y Grafana asignando los puertos estándar de estas dos aplicaciones. Para que grafana pueda obtener las métricas de prometheus se requieren dos aplicaciones que son: Node exporter y Cadvisor, su función se basa en traducir las métricas de prometheus para que grafana las entienda [9].

```
volumes:
  grafana-data:
  prometheus-data:

services:
  grafana:
    image: grafana/grafana:8.0.6
    container_name: grafana
    restart: unless-stopped
    volumes:
      - grafana-data:/var/lib/grafana
    networks:
      mi_red_privada:
        ipv4_address: 172.20.0.3
    ports:
      - "3000:3000"

  prometheus:
    image: prom/prometheus:v2.28.1
```

```

container_name: prometheus
restart: unless-stopped
volumes:
- ./prometheus.yml:/etc/prometheus/prometheus.yml
- prometheus-data:/prometheus
networks:
  mi_red_privada:
    ipv4_address: 172.20.0.4
ports:
- "9090:9090"
command:
- '--config.file=/etc/prometheus/prometheus.yml'
- '--storage.tsdb.path=/prometheus'
- '--storage.tsdb.retention.time=1y'
- '--web.enable-lifecycle'

```

```

node_exporter:
  image: quay.io/prometheus/node-exporter:latest
  container_name: node_exporter
  restart: unless-stopped
  networks:
    mi_red_privada:
      ipv4_address: 172.20.0.5
  ports:
  command:
    - '--path.procfs=/host/proc'
    - '--path.sysfs=/host/sys'
    - '--collector.filesystem.ignored-mount-points

```

```

"^(/sys|/proc|/dev|/host|/etc|/rootfs/var/lib/docker/containers|/rootfs/var/lib/d
ocker/overlay2|/rootfs/run/docker/netns|/rootfs/var/lib/docker/aufs)($$|/)"

```

```

cadvisor:
  image: gcr.io/cadvisor/cadvisor:latest
  container_name: cadvisor
  restart: unless-stopped
  networks:
    mi_red_privada:
      ipv4_address: 172.20.0.6
  expose:
  - "8080"
  volumes:
  - /:/rootfs:ro
  - /var/run:/var/run:rw
  - /sys:/sys:ro
  - /var/lib/docker:/var/lib/docker:ro

```

```

server:

```

```

image: servidor:latest
container_name: servervideo
restart: unless-stopped
networks:
  mi_red_publica:
    ipv4_address: 172.19.0.7
  mi_red_privada:
    ipv4_address: 172.20.0.7
expose:
  - "1935"

```

Continuando con el desarrollo del archivo “docker-compose.yml” esta parte del código se realiza la configuración de red de todos los contenedores usados en el docker compose, la red “mi\_red\_publica” aplica la configuración macvlan que permite que el contenedor del servidor pueda conectarse a la red ethernet del computador base y la red “mi\_red\_privada” contiene la configuración por defecto y es la red encargada de conectar a todos los contenedores para que se puedan comunicar entre ellos. Para estas dos configuraciones de red se asignan las direcciones ip 172.19.0.0/16 para “mi\_red\_publica” y 172.20.0.0/16 para “mi\_red\_privada”

```

networks:
  mi_red_publica:
    driver: macvlan
    driver_opts:
      parent: nombre del puerto ethernet usado
    ipam:
      config:
        - subnet: 172.19.0.0/16
          gateway: 172.19.0.1

  mi_red_privada:
    ipam:
      config:
        - subnet: 172.20.0.0/16
          gateway: 172.20.0.1

```

El siguiente código se crea en un editor de texto como bloc de notas y se nombra “prometheus.yml”. Este será el encargado de informarle a Prometheus en donde se encuentran los contenedores, para que este se conecte y empiece a obtener las métricas de cada uno [10].

```

1 global:
2   scrape_interval: 1s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
3   evaluation_interval: 1s # Evaluate rules every 15 seconds. The default is every 1 minute.
4
5 scrape_configs:
6   - job_name: 'prometheus'
7     static_configs:
8       - targets: ['prometheus:9090']
9
10  - job_name: 'cadvisor'
11    static_configs:
12      - targets: ['cadvisor:8080']
13
14  - job_name: 'node_exporter'
15    static_configs:
16      - targets: ['node_exporter:9100']
17
18  - job_name: 'server'
19    static_configs:
20      - targets: ['servervideo:1935']
21    #metrics_path: '/metrics'
22    #metrics_path: '/prometheus'

```

Figura 12: Contenido del archivo prometheus.yaml

```

global:
  scrape_interval: 1s # Set the scrape interval to every 15 seconds.
Default is every 1 minute.
  evaluation_interval: 1s # Evaluate rules every 15 seconds. The default is
every 1 minute.

```

```

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['grafana:3000']

  - job_name: 'grafana'
    static_configs:
      - targets: ['prometheus:9090']

  - job_name: 'cadvisor'
    static_configs:
      - targets: ['cadvisor:8080']

  - job_name: 'node_exporter'
    static_configs:
      - targets: ['node_exporter:9100']

  - job_name: 'server'
    static_configs:
      - targets: ['servervideo:1935']

```

Luego, se debe ejecutar el archivo de texto del docker-compose para crear los contenedores de las imágenes de Grafana, Prometheus, Node Exporter, Cadvisor y el servidor.

```
sudo docker-compose up -d --build
```

```
docker@docker-VirtualBox:~/Escritorio/Servidor$ sudo docker compose up -d --build
[+] Running 31/31
✓ cadvisor Pulled
✓ 619be1103602 Pull complete
✓ 3b8469b194b8 Pull complete
✓ 6361eeb1639c Pull complete
✓ 4f4fb700ef54 Pull complete
✓ 902eccc70f3 Pull complete
✓ prometheus Pulled
✓ aa2a8d90b84c Pull complete
✓ b45d31ee2d7f Pull complete
✓ da9de9139824 Pull complete
✓ d04e751b88d5 Pull complete
✓ 13f11ea3536c Pull complete
✓ 1d81771985c9 Pull complete
✓ d471c28936c9 Pull complete
✓ 827e29e97e58 Pull complete
✓ 9a0bd55ef653 Pull complete
✓ 16e358518d2f Pull complete
✓ bfdb42c9d185 Pull complete
Terminal 6d5e5f1b Pull complete
✓ node_exporter Pulled
✓ 9fa9226be034 Pull complete
✓ 1617e25568b2 Pull complete
✓ a7193bcb1fb2 Pull complete
✓ grafana Pulled
✓ 540db60ca938 Pull complete
✓ b1a70ac2e628 Pull complete
✓ b31f6ee16ea8 Pull complete
✓ 509c33253828 Pull complete
✓ 4d06dfbbc1c3 Pull complete
✓ f1f6b61b846e Pull complete
✓ a349d3a4a214 Pull complete
[+] Running 9/9
✓ Network servidor_mi_red_privada Created
✓ Network servidor_mi_red_publica Created
✓ Volume "servidor_prometheus-data" Created
✓ Volume "servidor_grafana-data" Created
✓ Container cadvisor Started
✓ Container servervideo Started
✓ Container grafana Started
✓ Container prometheus Started
✓ Container node_exporter Started
```

Figura 13: Creación de los contenedores correspondientes

Para confirmar el funcionamiento del sistema se ejecuta el siguiente comando.

```
sudo docker ps
```

```
docker@docker-VirtualBox:~/Escritorio/Servidor$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3d4a6436abe7	quay.io/prometheus/node-exporter:latest	"/bin/node_exporter -"	4 minutes ago	Up 4 minutes	0.0.0.0:9100->9100/tcp, :::9100->9100/tcp	node_exporter
ac42654cd99f	prom/prometheus:v2.28.1	"/bin/prometheus ->C-"	4 minutes ago	Up 4 minutes	0.0.0.0:9090->9090/tcp, :::9090->9090/tcp	prometheus
7ffd1312a9b5	gcr.io/cadvisor/cadvisor:latest	"/usr/bin/cadvisor -"	4 minutes ago	Up 4 minutes (healthy)	8080/tcp	cadvisor
c79b304566b5	grafana/grafana:8.0.6	"/run.sh"	4 minutes ago	Up 4 minutes	0.0.0.0:3000->3000/tcp, :::3000->3000/tcp	grafana
8b94e484ca0c	servidor:latest	"nginx -g 'daemon of-'"	4 minutes ago	Up 4 minutes	1935/tcp	servervideo

Figura 14: Confirmación del funcionamiento de todos los contenedores

En la Figura 14 se encuentran los contenedores del docker donde en la columna de status se define si el contenedor está activado o apagado. En este caso los contenedores están activados o en estado “up”.

## Transmisión de video

Se debe acceder a la terminal del contenedor “servervideo”

```
sudo docker exec -it servervideo /bin/bash
```

```
docker@doker-VirtualBox:~/Escritorio/Servidor$ sudo docker exec -it servervideo bin/bash
root@8b94e484ca0c:/#
```

Figura 15: Ingreso al bash del servidor de video

Para transmitir el video es necesario acceder a la ubicación del video para esto se ejecutan los comandos `cd` y `ls`, que permiten ir a un directorio y listar el contenido del mismo.

```
root@8b94e484ca0c:/# cd
root@8b94e484ca0c:~# ls
Video.mp4
root@8b94e484ca0c:~#
```

Figura 16: Confirmación de la ubicación del video a transmitir

Ahora, se transmite el video indicando la dirección ip de la transmisión, y donde se debe finalizar con el código (ID) del contenedor. Por ejemplo, se observa en la Figura 18 que el código del contenedor es 8b94e484ca0c

```
ffmpeg -re -stream_loop -1 -i "Video.mp4" -c:v copy -c:a aac -ar 44100 -ac 2 -f flv rtmp://localhost/live/codigodelcontenedor
```

```
root@8b94e484ca0c:~# ffmpeg -re -stream_loop -1 -i "Video.mp4" -c:v copy -c:a aac -ar 44100 -ac 2 -f flv rtmp://localhost/live/8b94e484ca0c
ffmpeg version 4.3.7-0+deb11u1 Copyright (c) 2000-2024 the FFmpeg developers
  built with gcc 10 (Debian 10.2.1-6)
  configuration: --prefix=/usr --extra-version=0+deb11u1 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/include/x86_64-linux-gnu --arch=amd64 --enable-gpl --disable-stripping --enable-avresample --disable-filter=resample --enable-gnutls --enable-ladspa --enable-lbaom --enable-libass --enable-libbluray --enable-libs2b --enable-libcaca --enable-libcdio --enable-libcodecs2 --enable-libdav1d --enable-libflite --enable-libfontconfig --enable-libfreetype --enable-libfribidi --enable-libgme --enable-libgsm --enable-libjack --enable-libmp3lame --enable-libmysofa --enable-libopenjpeg --enable-libopenmpt --enable-libopus --enable-libpulse --enable-librabbitmq --enable-librsvg --enable-librubberband --enable-ltshime --enable-libsnappy --enable-libsoxr --enable-libspeex --enable-lsrt --enable-libsrt --enable-libssh --enable-libtheora --enable-libtwolame --enable-libvidstab --enable-libvorbis --enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx265 --enable-libx264 --enable-libxvid --enable-libzmq --enable-libzvt --enable-lv2 --enable-omx --enable-opengl --enable-opencore --enable-opencl --enable-opengl --enable-sdl2 --enable-pocketsphinx --enable-libnfx --enable-libdc1394 --enable-libdrm --enable-libiec61883 --enable-chromaprint --enable-frei0r --enable-libx264 --enable-shared
  libavutil 56. 51.100 / 56. 51.100
  libavcodec 58. 91.100 / 58. 91.100
  libavformat 58. 45.100 / 58. 45.100
  libavdevice 58. 10.100 / 58. 10.100
  libavfilter 7. 85.100 / 7. 85.100
  libavresample 4. 0. 0 / 4. 0. 0
  libswscale 5. 7.100 / 5. 7.100
  libswresample 3. 7.100 / 3. 7.100
  libpostproc 55. 7.100 / 55. 7.100
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'Video.mp4':
```

Figura 18: Proceso de transmisión del video mediante el servidor nginx

## Configuración y visualización Prometheus y Grafana.

Para poder observar las métricas de Prometheus se busca desde cualquier navegador la url de la máquina anfitriona dirigida al puerto en el que opera Prometheus

(<http://localhost:9090>) y se presiona el botón ubicado en la parte izquierda del botón azul execute, posteriormente se oprime el botón azul “execute” [10].

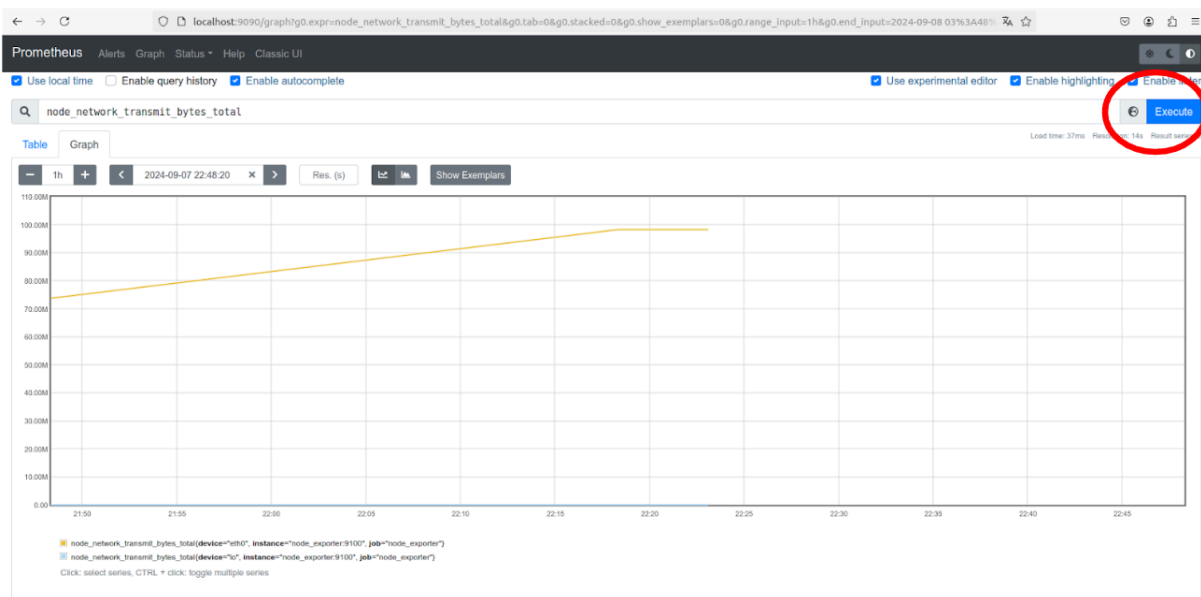


Figura 19: Interfaz gráfica de Prometheus (gráfica de las métricas)

Para poder observar las métricas de Grafana se busca en desde cualquier navegador la url de la máquina anfitriona dirigida al puerto en el que opera Grafana (<http://localhost:3000>). Por defecto, Grafana exige un usuario y una contraseña, que normalmente son “admin” para usuario y contraseña [8].

The image shows the Grafana login screen. At the top, there is the Grafana logo (a yellow gear with a blue spiral) and the text 'Welcome to Grafana'. Below this, there are two input fields: 'Email or username' with the value 'admin' and 'Password' with masked characters '.....'. A blue 'Log in' button is positioned below the password field. At the bottom right, there is a link that says 'Forgot your password?'. The background is dark blue.

Figura 20: Inicio de Grafana

Para observar las métricas se necesita configurar a Grafana la url de Prometheus, para la configuración se presiona la el botón remarcado en rojo y se selecciona la opción de “datasource” [10]

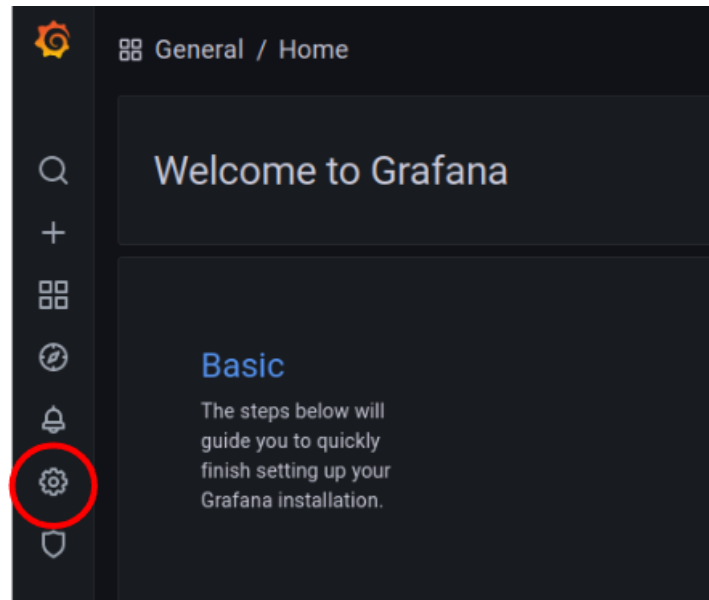


Figura 21: Panel lateral de Grafana

Posteriormente, se presiona el botón “add data source” [9].

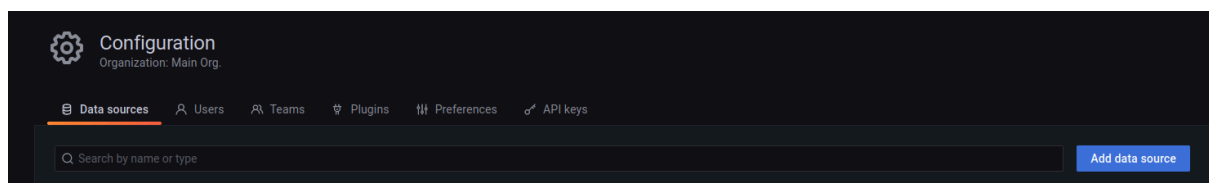


Figura 22: Configuración data source Grafana

A continuación, en la Figura 23, se escribe la url de Prometheus, en este caso es “prometheus:9090” [10].



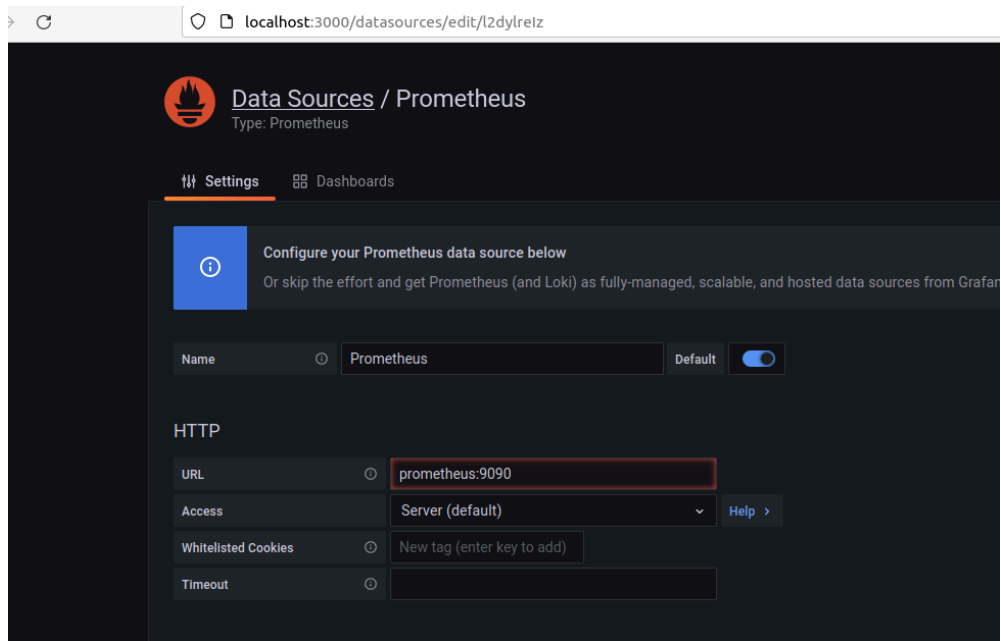


Figura 23: Parámetros para data source Grafana

Para finalizar con la configuración de Prometheus se presiona en el botón “save & test” [10]

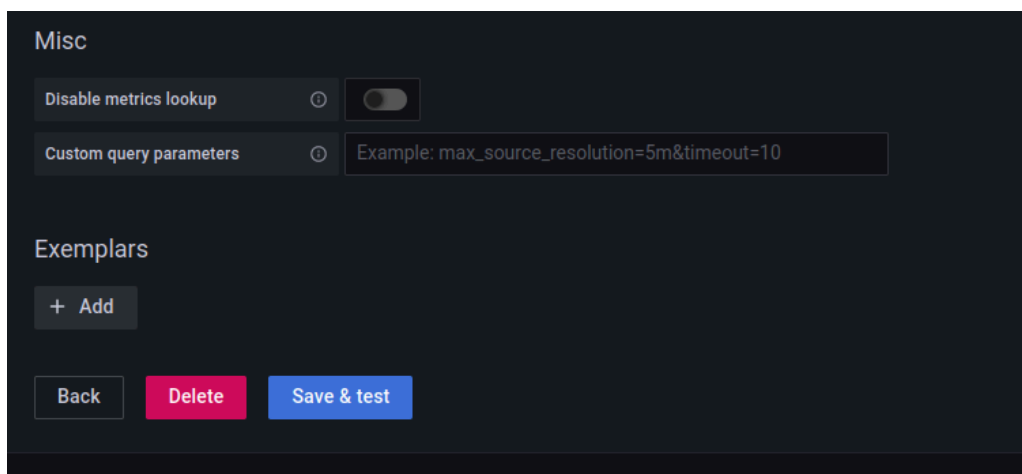


Figura 24: Guardar los cambios para el data source de Grafana

Para observar de forma ordenada y estética las métricas, se presiona el botón remarcado en rojo y se selecciona la opción de “manage” [11].

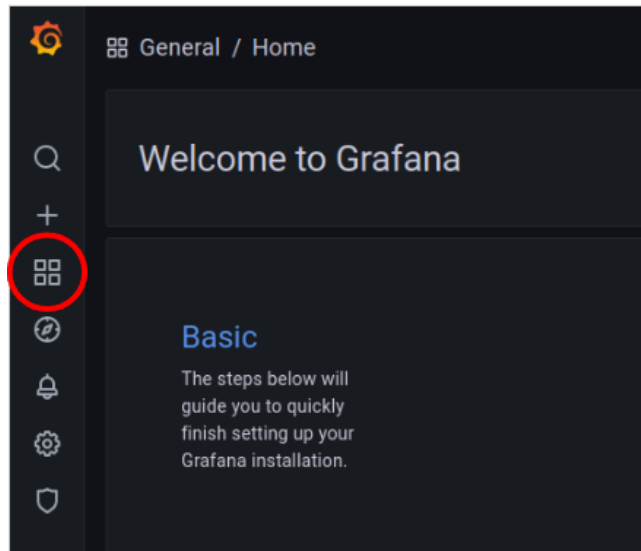


Figura 25: Panel lateral de Grafana

Posteriormente en la Figura 26 se presiona el botón “import” ubicado en la esquina superior derecha [11].

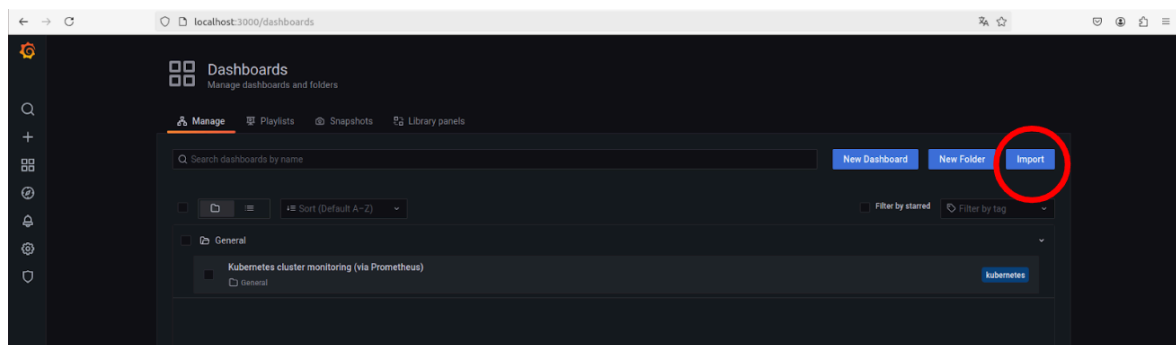


Figura 26: Importar Dashboard para Grafana

Seguidamente en la Figura 27 se escribe en el recuadro “import via grafana.com” el código 3119, que es una plantilla con configuración de métricas predeterminada por Grafana [11].

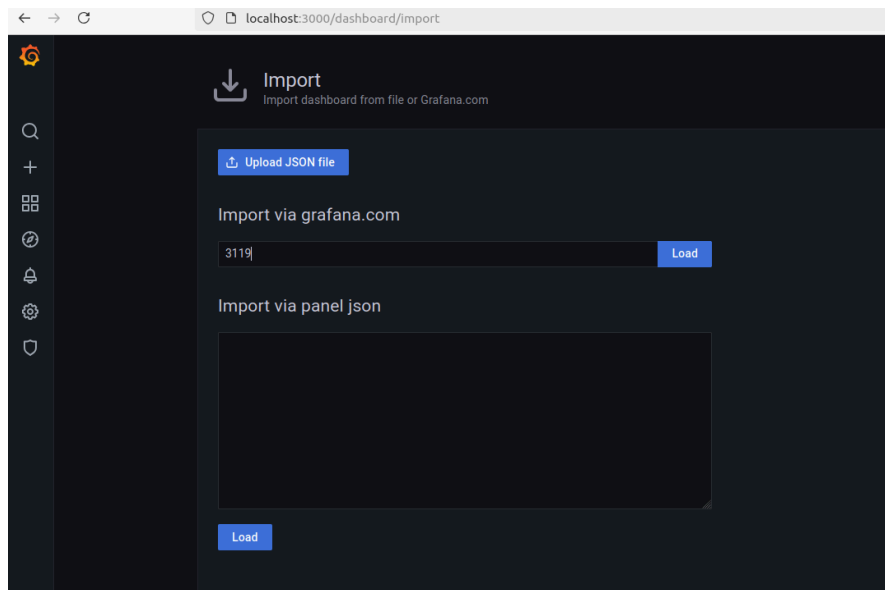


Figura 27: Importar la plantilla 3119 para el dashboard de Grafana

Y finalmente en la Figura 28 se presiona el botón de importar [11].

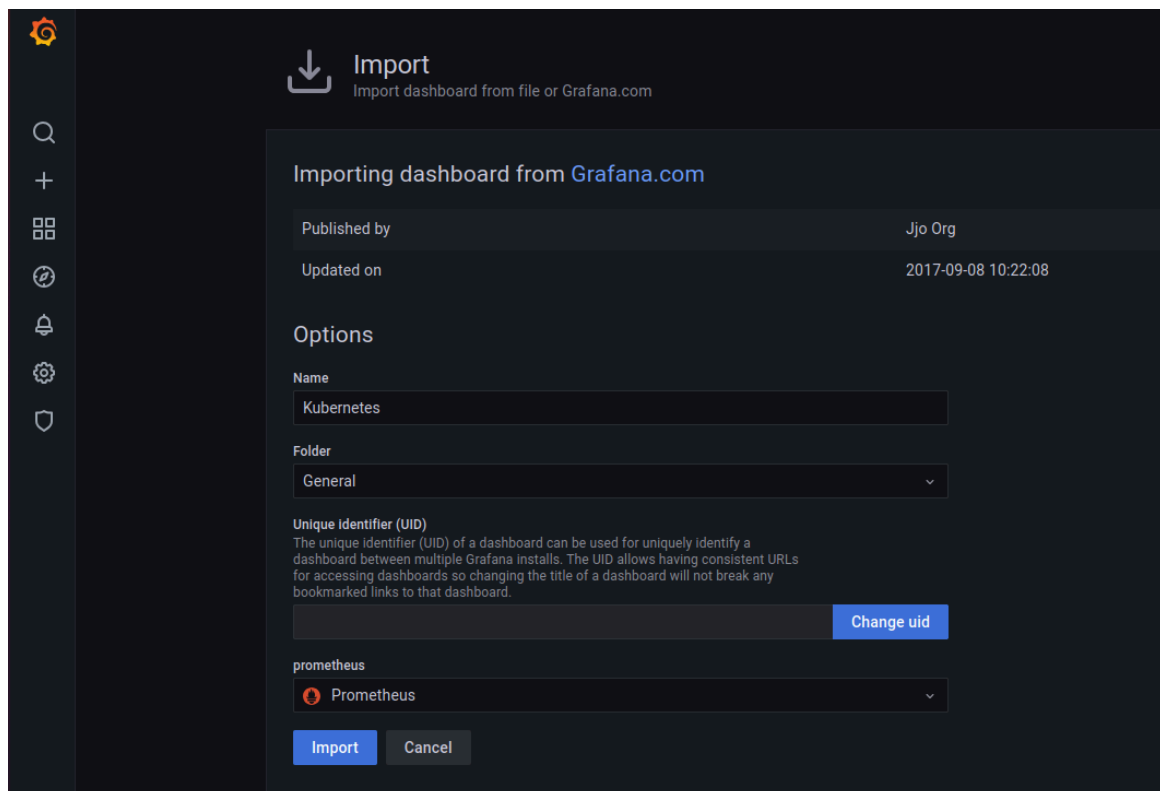


Figura 28: Configuración final para el dashboard de Grafana

Al crear el dashboard, como se muestra en la Figura 29, Grafana arroja métricas de uso de CPU, memoria usada y uso de internet.



Figura 29: dashboard de Grafana

## Configuración del cliente

Para iniciar la creación del cliente que recibirá la transmisión multimedia utilizando el protocolo RTMP, se procede con la elaboración de un Dockerfile con los siguientes parámetros:

```
FROM ubuntu:latest
```

```
# Actualizar los paquetes de Ubuntu y luego instalar FFmpeg y las utilidades de ping
```

```
RUN apt-get update && \
    apt-get install -y ffmpeg iputils-ping && \
    apt-get clean
```

```
# Comando predeterminado
```

```
CMD ["bash"]
```

```

1 FROM ubuntu:latest
2
3 # Actualizar los paquetes de Ubuntu y luego instalar FFmpeg y las utilidades de ping
4 RUN apt-get update && \
5     apt-get install -y ffmpeg iputils-ping && \
6     apt-get clean
7
8 # Comando predeterminado
9 CMD ["bash"]
10

```

Figura 30: Dockerfile del cliente

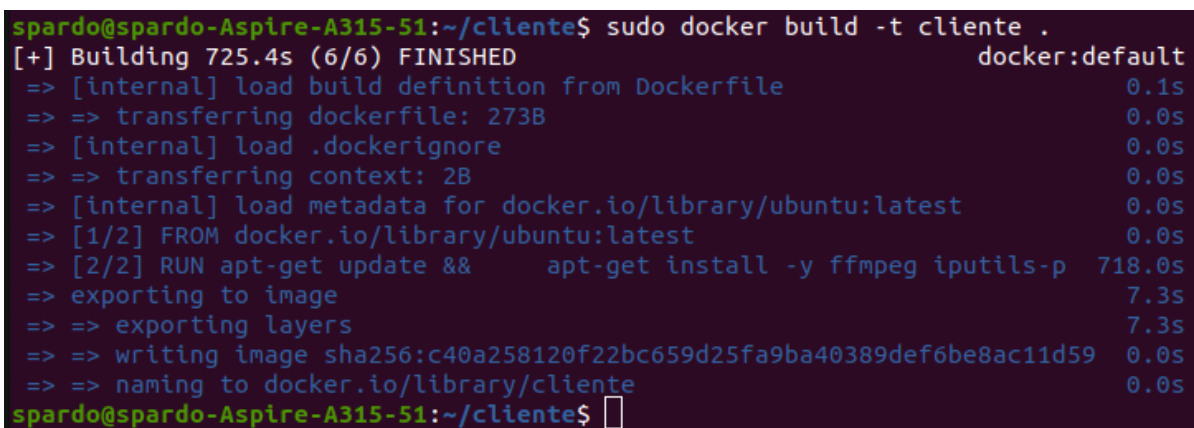
Con el Dockerfile anterior las instrucciones especifican que, al crear una nueva imagen, se utilizará como imagen base `ubuntu:latest` (la última versión disponible de la imagen de Ubuntu) y que se incluirán las utilidades y funciones de ffmpeg (para video streaming), así como las herramientas para verificar la conectividad como ping.

Para construir la imagen, se emplea el siguiente comando:

```
sudo docker build -t "Nombre de la imagen" .
```

En donde:

- `docker build`: Este es el comando básico de Docker para construir una imagen a partir de un Dockerfile.
- `-t "Nombre de la imagen"`: La opción `-t` (o `--tag`) se utiliza para etiquetar (o nombrar) la imagen que estás construyendo. En este caso, "Nombre de la imagen" es un marcador de posición para el nombre que deseas asignar a la imagen. Este nombre es útil para identificar y ejecutar la imagen más tarde (por ejemplo, "Mi-imagen:v1.0").
- `.`: El punto "." al final del comando especifica el contexto de construcción, que es el directorio donde Docker buscará el Dockerfile y los archivos necesarios para construir la imagen. En este caso, el punto indica que el Dockerfile está en el directorio actual. Por ende para la creación del dockerfile se tiene que estar en la carpeta donde se encuentre dicho archivo y así poder usar el comando ".".



```
spardo@spardo-Aspire-A315-51:~/cliente$ sudo docker build -t cliente .
[+] Building 725.4s (6/6) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile              0.1s
=> => transferring dockerfile: 273B                             0.0s
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                    0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest 0.0s
=> [1/2] FROM docker.io/library/ubuntu:latest                  0.0s
=> [2/2] RUN apt-get update && apt-get install -y ffmpeg iputils-ping 718.0s
=> exporting to image                                           7.3s
=> => exporting layers                                           7.3s
=> => writing image sha256:c40a258120f22bc659d25fa9ba40389def6be8ac11d59 0.0s
=> => naming to docker.io/library/cliente                       0.0s
spardo@spardo-Aspire-A315-51:~/cliente$
```

Figura 31: Construcción de la imagen del cliente

Una vez que se ha creado la imagen, se verifica su presencia en la lista de imágenes de la máquina donde se desplegará el cliente. El comando utilizado para ver el listado de imágenes es:

docker images

```
spardo@spardo-Aspire-A315-51:~/cliente$ docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
cliente              latest      c40a258120f2  3 minutes ago  821MB
```

Figura 32: Confirmación de la creación de la imagen del cliente

A continuación, se crea el contenedor a partir de la imagen recién creada. Para ello, se utiliza el siguiente comando:

```
docker run -it -d --network=bridge --name clienteRTMP cliente
```

```
spardo@spardo-Aspire-A315-51:~/cliente$ docker run -it -d --network=bridge --name clienteRTMP cliente
8c1b84c1e809bf5bb60abfb46dbcf3592d9ca7ae598828aabc7c0e848577f950
```

Figura 33: Creación del contenedor del cliente

Explicación del comando anterior:

- `docker run` es el comando principal para crear y ejecutar un nuevo contenedor a partir de una imagen Docker.
- `-it` (interactive) mantiene la entrada estándar abierta para el contenedor, lo que permite que el contenedor reciba entradas interactivas. `(-it)` permite que se pueda interactuar con el contenedor desde la línea de comandos. Es útil para sesiones interactivas, como cuando se utiliza `bash` en el contenedor.
- `-d` (detached): Ejecuta el contenedor en segundo plano (modo "detached"). Esto significa que el contenedor se ejecutará en el fondo, y el comando `docker run` terminará inmediatamente después de iniciar el contenedor. No se verá la salida del contenedor en la terminal actual, pero el contenedor seguirá ejecutándose en segundo plano.
- `--network=bridge` especifica que el contenedor debe conectarse a una red llamada `bridge`. En Docker, `bridge` es el nombre predeterminado para la red de tipo puente (bridge network).
- `--name clienteRTMP` asigna un nombre al contenedor "clienteRTMP". Esto facilita la referencia al contenedor en lugar de usar el ID del contenedor, que es una cadena larga y poco amigable. Se usa el nombre para gestionar el contenedor, como detenerlo o eliminarlo.

- cliente el nombre de la imagen Docker que se usará para crear el contenedor. Docker buscará esta imagen localmente en la máquina. Si no encuentra la imagen localmente, intentará descargarla del Docker Hub.

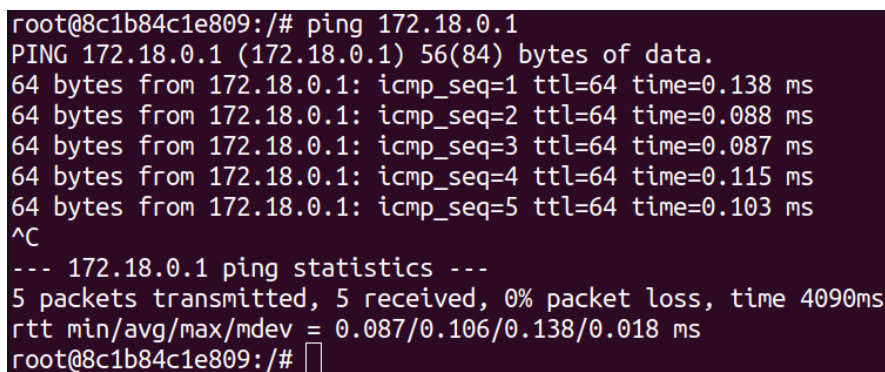
Con el contenedor creado, se establece la conexión punto a punto al servidor mediante un cable UTP con configuración cruzada. Posteriormente, se accede al terminal bash del contenedor utilizando el siguiente comando:

```
docker exec -it clienteRTMP bash
```

- docker exec es el comando que se utiliza para ejecutar un nuevo comando en un contenedor que ya está en ejecución. A diferencia de docker run, que se usa para iniciar un nuevo contenedor, docker exec permite ejecutar comandos en un contenedor que ya está activo.
- bash es el comando que se desea ejecutar dentro del contenedor. En este caso, se está iniciando el shell bash.

Al ingresar al bash del contenedor, se presenta un cambio hacia la terminal interactiva. Una vez dentro del contenedor, se realiza un ping hacia la IP de la máquina del servidor, que es “172.18.0.1”. Para llevar a cabo el ping, se utiliza el siguiente comando:

```
ping 172.18.0.1
```



```
root@8c1b84c1e809:/# ping 172.18.0.1
PING 172.18.0.1 (172.18.0.1) 56(84) bytes of data.
64 bytes from 172.18.0.1: icmp_seq=1 ttl=64 time=0.138 ms
64 bytes from 172.18.0.1: icmp_seq=2 ttl=64 time=0.088 ms
64 bytes from 172.18.0.1: icmp_seq=3 ttl=64 time=0.087 ms
64 bytes from 172.18.0.1: icmp_seq=4 ttl=64 time=0.115 ms
64 bytes from 172.18.0.1: icmp_seq=5 ttl=64 time=0.103 ms
^C
--- 172.18.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4090ms
rtt min/avg/max/mdev = 0.087/0.106/0.138/0.018 ms
root@8c1b84c1e809:/#
```

Figura 35: Verificación de conexión entre el contenedor y el anfitrión

Posteriormente, se realiza un ping al servidor (contenedor) “nginx-rtmp”, que tiene la dirección IP `172.17.0.1`, utilizando el mismo método anteriormente descrito.

```
ping 192.168.1.13
```

```
root@8c1b84c1e809:/# ping 192.168.1.13
PING 192.168.1.13 (192.168.1.13) 56(84) bytes of data.
64 bytes from 192.168.1.13: icmp_seq=1 ttl=64 time=0.068 ms
64 bytes from 192.168.1.13: icmp_seq=2 ttl=64 time=0.066 ms
64 bytes from 192.168.1.13: icmp_seq=3 ttl=64 time=0.085 ms
64 bytes from 192.168.1.13: icmp_seq=4 ttl=64 time=0.061 ms
64 bytes from 192.168.1.13: icmp_seq=5 ttl=64 time=0.079 ms
^C
--- 192.168.1.13 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4085ms
rtt min/avg/max/mdev = 0.061/0.071/0.085/0.008 ms
root@8c1b84c1e809:/#
```

Figura 36: Verificación de conexión entre cliente y servidor

Con lo anterior es posible verificar que hay comunicación entre contenedores y que los dispositivos ya están listos para establecer un proceso servidor-cliente.

Con el comando `ffmpeg rtmp://172.17.0.1 /live/stream/video.mp4` se comienza a demandar recursos del dispositivo donde está alojado el servidor. Este proceso arroja métricas captadas por Prometheus y plasmadas gráficamente por Grafana en la vista de la Figura 29.

## Conclusiones

- Es posible realizar conexiones entre servidores y clientes virtualizados mediante contenedores con el fin de optimizar recursos de hardware y mejorar la gestión de los mismos al no establecerse directamente desde las máquinas locales.
- Gracias a la virtualización de servicios y comunicaciones cliente-servidor es posible obtener métricas referentes al estado de la red, los recursos consumidos y la calidad del servicio mediante herramientas como Prometheus y Grafana que pueden servir para definir el estado de algún servidor y monitorear el proceso o servicio sin la necesidad de recurrir a un dispositivo externo. Se gestionan recursos de hardware, de red y se centraliza el manejo de la información.

## Referencias

[1] J. Casierra Cavada, X. Quiñónez Ku, L. Herrera Izquierdo, y C. Egas Acosta, "Virtualización de redes y servidores emulando infraestructuras tecnológicas," Rev. Hallazgos21, vol. 3, suplemento especial, 2018. [En línea]. Disponible en: <http://revistas.pucese.edu.ec/hallazgos21>



- [2]. J. J. S. Hernández, *Aprender Docker, un enfoque práctico*. Marcombo, 2022.
- [3]. “FFmpeg”. FFmpeg. [En línea]. Disponible: <https://ffmpeg.org/>
- [4]. “Ubuntu”. Docker Documentation. Accedido el 22 de agosto de 2024. [En línea]. Disponible: <https://docs.docker.com/engine/install/ubuntu/>
- [5]. “Dockerfile reference”. Docker Documentation. Accedido el 15 de septiembre de 2024. [En línea]. Disponible: <https://docs.docker.com/reference/dockerfile/>
- [6]. S. Ramírez. Docker Hub Container Image Library | App Containerization. Accedido el 15 de septiembre de 2024. [En línea]. Disponible: <https://hub.docker.com/r/tiangolo/nginx-rtmp/dockerfile>
- [7]. S. Ramírez. “GitHub - tiangolo/nginx-rtmp-docker: Docker image with Nginx using the nginx-rtmp-module module for live multimedia (video) streaming.” GitHub. Accedido el 15 de septiembre de 2024. [En línea]. Disponible: <https://github.com/tiangolo/nginx-rtmp-docker/blob/master/nginx.conf>
- [8]. Jason Cheung. *Build local images for docker-compose ( Docker Go & Redis example)*. (12 de febrero de 2021). Accedido el 15 de septiembre de 2024. [Video en línea]. Disponible: <https://www.youtube.com/watch?v=HYhREpHFdo0>
- [9]. “observability/01-monitoreo-recursos/docker-compose.yml at main · caosbinario/observability”. GitHub. Accedido el 15 de septiembre de 2024. [En línea]. Disponible: <https://github.com/caosbinario/observability/blob/main/01-monitoreo-recursos/docker-compose.yml>
- [10]. Caos Binario. *STACK MONITOREO con GRAFANA Y PROMETHEUS (y más)*. (15 de septiembre de 2021). Accedido el 15 de septiembre de 2024. [Video en línea]. Disponible: <https://www.youtube.com/watch?v=PCJwJpbln6Q>
- [11]. milunadev. 🚀 *INSTALANDO GRAFANA Y PROMETHEUS EN MINIKUBE - MONITORING STACK* 🤖. (5 de mayo de 2024). Accedido el 15 de septiembre de 2024. [Video en línea]. Disponible: <https://www.youtube.com/watch?v=jWI8RpU0ps0>