

PROYECTO DE FIN DE CICLO

ERIKA ALESSIO
IES BEZMILIANA
CURSO 24/25

Implementación automatizada de WordPress con Ansible sobre Minikube y monitorización con Prometheus



1. Introducción:	1
1.1 Descripción del proyecto:	1
1.2 Objetivos generales y específicos:	2
1.3 Metodología de trabajo:	2
2. Fundamentos teóricos:	2
2.1 Virtualización y contenedores:	2
2.2 Kubernetes y Minikube:	4
2.3. WordPress y MySQL:	5
2.4 Automatización con Ansible:	6
2.5 Monitorización con Prometheus y Grafana:	7
3. Diseño del proyecto:	8
3.1 Esquema de la arquitectura:	8
3.1 Tecnologías utilizadas:	9
4. Desarrollo del proyecto:	10
4.1 Preparación del entorno:	10
4.2. Despliegue con Ansible y YAML	15
1a. Mysqlsecret.yaml:	16
1b. Mysqlvolume.yaml:	17
1c. Mysqlservice.yaml:	17
1d. MySqldeployment.yaml:	18
2a. wordpressvolume.yaml:	19
2b. wordpressservice.yaml:	20
2c. wordpressexporter.yaml:	20
2d. wordpressdeployment.yaml:	21
4.3. Configuración de WordPress	24
4.4. Integración de Prometheus:	27
4.5. Integración de Grafana:	35
5. Conclusiones	41
5.1. Resultados obtenidos	41
5.2. Dificultades encontradas	42
5.3. Mejoras futuras	42
6. Anexos	42
6.1 Enlace al repositorio GitHub	42
6.2 Webgrafía	42

1. Introducción:

1.1 Descripción del proyecto:

Este proyecto se basa en el despliegue automatizado de una infraestructura de contenedores que alojan una aplicación Wordpress con base de datos SQL y Kubernetes como orquestador. Dicha automatización se realiza mediante Ansible. Además, se implementará un sistema de monitoreo con Prometheus y Grafana para poder ver las métricas de los servicios. El entorno del proyecto será una máquina virtual creada en Proxmox y se utilizará Minikube como entorno de pruebas para Kubernetes.

1.2 Objetivos generales y específicos:

El objetivo general del proyecto es el despliegue de una aplicación web con su base de datos y su monitorización de manera automatizada.

Los objetivos específicos incluyen:

1. Instalar y configurar Docker, Prometheus, Ansible...
2. Desplegar automáticamente el servicio web y la base de datos.
3. Asegurar la persistencia de datos mediante volúmenes persistentes.
4. Configurar los exportadores para poder ver las métricas con Prometheus y Grafana.

1.3 Metodología de trabajo:

La metodología a seguir para realizar este proyecto ha sido en primer lugar la realización de un esquema básico de la infraestructura y la definición de las herramientas necesarias para el despliegue del mismo.

En segundo lugar, se ha investigado el uso de las herramientas a utilizar para darles el mejor uso posible. Tras esto, se ha creado el entorno base en Proxmox para empezar a darle forma al proyecto. En función de las necesidades, se han creado playbooks en Ansible para automatizar el despliegue de la infraestructura y también ficheros YAML organizados mediante Kustomize.

La comprobación del correcto funcionamiento se lleva a cabo mediante comandos kubectl (como “kubectl get pods”, para ver el estado de los mismos) y mediante la herramienta Grafana, desde donde podemos ver la recopilación de las métricas de forma gráfica.

2. Fundamentos teóricos:

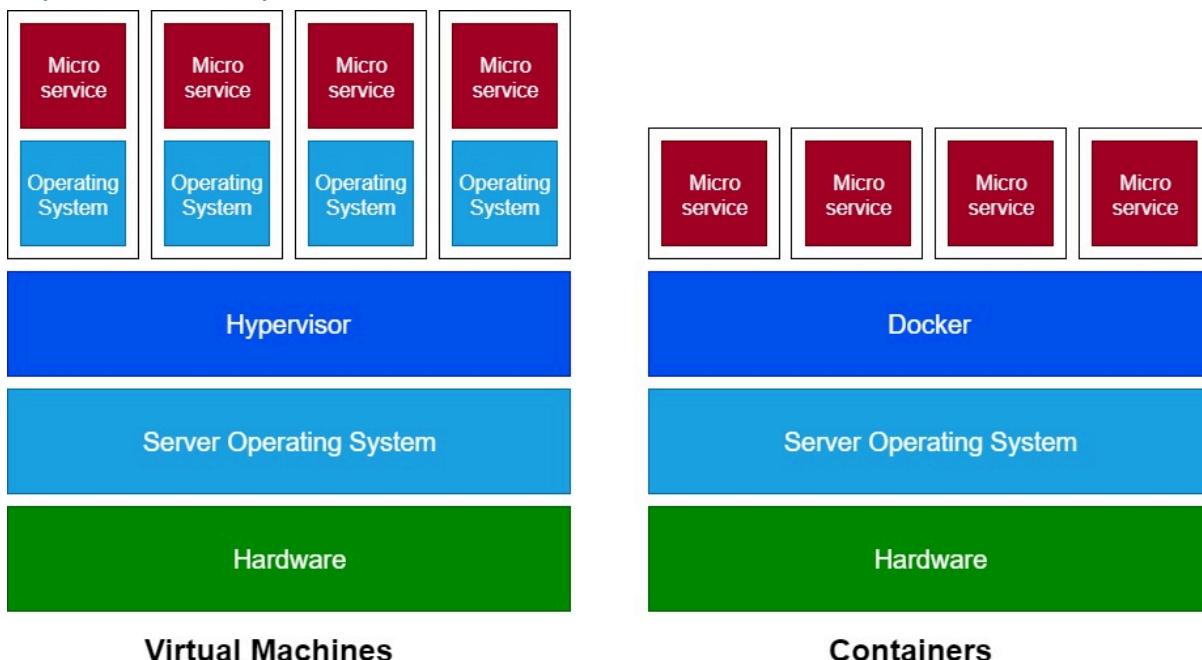
2.1 Virtualización y contenedores:

La virtualización es la manera de crear representaciones virtuales de recursos como servidores y redes a partir de máquinas físicas. Mediante la división de los recursos del hardware, se permite aumentar y potenciar la eficacia del sistema y su rentabilidad.

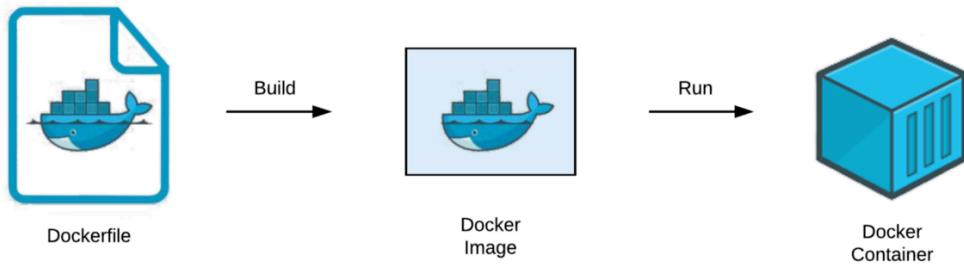
Algunas de las ventajas de la virtualización son:

- Eficiencia de los recursos: se aprovecha al máximo la capacidad del hardware físico.
- Gestión facilitada: se centraliza el trabajo y se automatizan procesos.
- Menor consumo energético: al solo necesitar un equipo físico donde se crean varias unidades virtuales, el ahorro de energía es mayor.

Un contenedor es un conjunto de procesos aislados del sistema destinados a ejecutar una aplicación. Lo fundamental de los contenedores es su portabilidad. La diferencia clave entre los contenedores y las máquinas virtuales es que estas virtualizan toda una máquina hasta las capas de hardware, y los contenedores solo virtualizan las capas de software por encima del nivel del sistema operativo. Las máquinas virtuales requieren de un hipervisor.

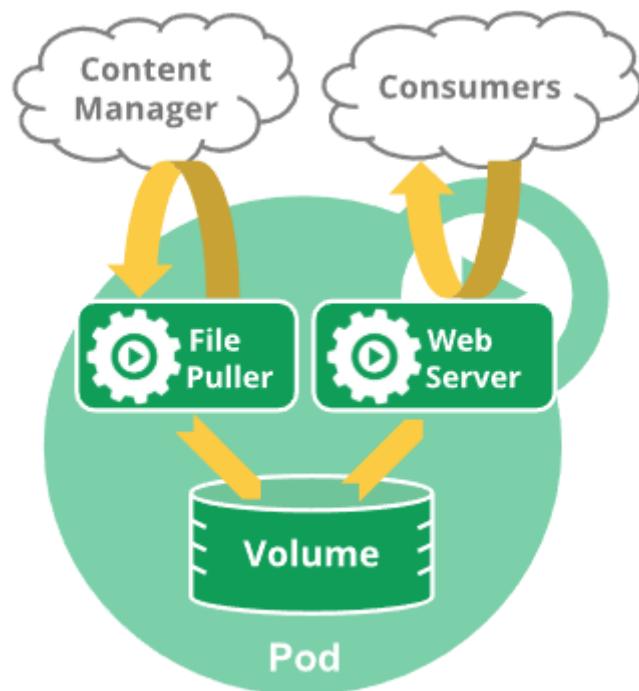


Docker es una tecnología de organización de contenedores. Como hemos visto antes, un contenedor es una instancia ejecutable de una imagen. Las imágenes son plantillas que definen al contenedor con el código y las librerías necesarias para su ejecución. Desde una imagen pueden crearse infinidad de contenedores pero un contenedor es creado a partir de una imagen.



2.2 Kubernetes y Minikube:

Kubernetes es una plataforma de código abierto para automatizar la implementación, el escalado y la administración de aplicaciones de contenedores. No opera a nivel de hardware pero sí a nivel de contenedor, organizando los recursos con etiquetas. La unidad más pequeña con la que trabaja Kubernetes es el Pod. Un pod es un objeto que contiene uno o varios contenedores que comparten recursos como almacenamiento y red. Los contenedores dentro del mismo pod comparten dirección IP y puerto y se pueden comunicarse a través del localhost. También tienen acceso a volúmenes compartidos. Un clúster es un grupo de nodos (máquinas) que trabajan en conjunto. Un nodo es una máquina que ejecuta pods. Existen dos tipos de nodos, el máster, que gestiona el clúster y los worker que son los que ejecutan las aplicaciones en pods.

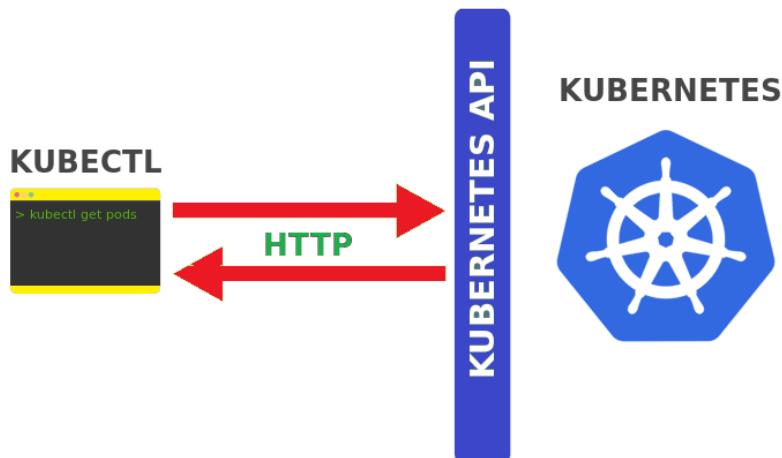


Por otro lado, Minikube es una herramienta de Kubernetes que nos permite ejecutar un clúster en una máquina local con un solo nodo. Es una manera más sencilla y ligera de

operar con Kubernetes, especialmente para entornos de desarrollo y aprendizaje a pequeña escala.

Utilizaremos la herramienta kubectl que es la interfaz de línea de comandos de Kubernetes. Esta es útil para la comunicación entre las distintas partes a través de la API de Kubernetes. Nos permite crear, eliminar, actualizar e inspeccionar objetos de Kubernetes.

Cada vez que ejecutamos un comando con kubectl, se genera una solicitud HTTP a la API REST y la envía al servidor de la API de Kubernetes. Posteriormente, se recupera el resultado y se muestra en la terminal.



2.3. WordPress y MySQL:



Wordpress es un CMS (Content Management System) utilizado para crear sitios web y blogs. Está desarrollado en PHP y necesita un servidor web como Apache o Nginx y una base de datos para funcionar. Es aquí donde entra en la ecuación Mysql.

MySQL es un sistema gestor de bases de datos relacional. Wordpress lo utiliza para guardar información como usuarios y configuraciones. Wordpress en este caso sería el cliente de Mysql, ya que se conecta al mismo para leer o guardar datos. En el actual

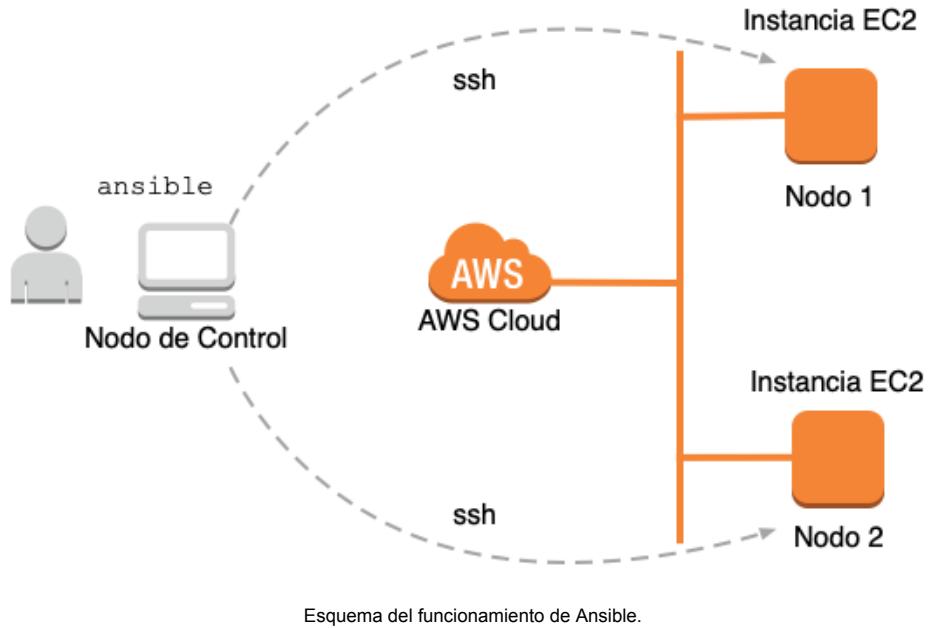
proyecto, se encuentran separados en pods distintos pero se comunican mediante kubernetes de manera segura gracias a los secrets configurados en las contraseñas.

2.4 Automatización con Ansible:



Ansible es una herramienta de automatización de tareas sin agentes (no necesita instalarse en el nodo administrado). Esta herramienta es simple e intuitiva y utiliza el protocolo SSH (en Unix) o WINRM (en Windows) para comunicarse entre el nodo de control y el nodo gestionado. Los fragmentos de código que ejecutan las tareas son llamados módulos. Estos módulos se agrupan en playbooks en formato YAML. El lenguaje YAML (“Yet Another Markup Language”) es fácilmente interpretable, lo que hace este proceso de automatización una tarea accesible.

Mediante la automatización con Ansible, se centraliza la gestión de los nodos administrados y se reducen los errores a la hora de implementar cambios en los distintos entornos, ya que se reducen las tareas repetitivas. Es una herramienta tanto para ejecutar configuraciones complejas a largo plazo como para ejecutar comandos en varios servidores simultáneamente.



Esquema del funcionamiento de Ansible.

2.5 Monitorización con Prometheus y Grafana:

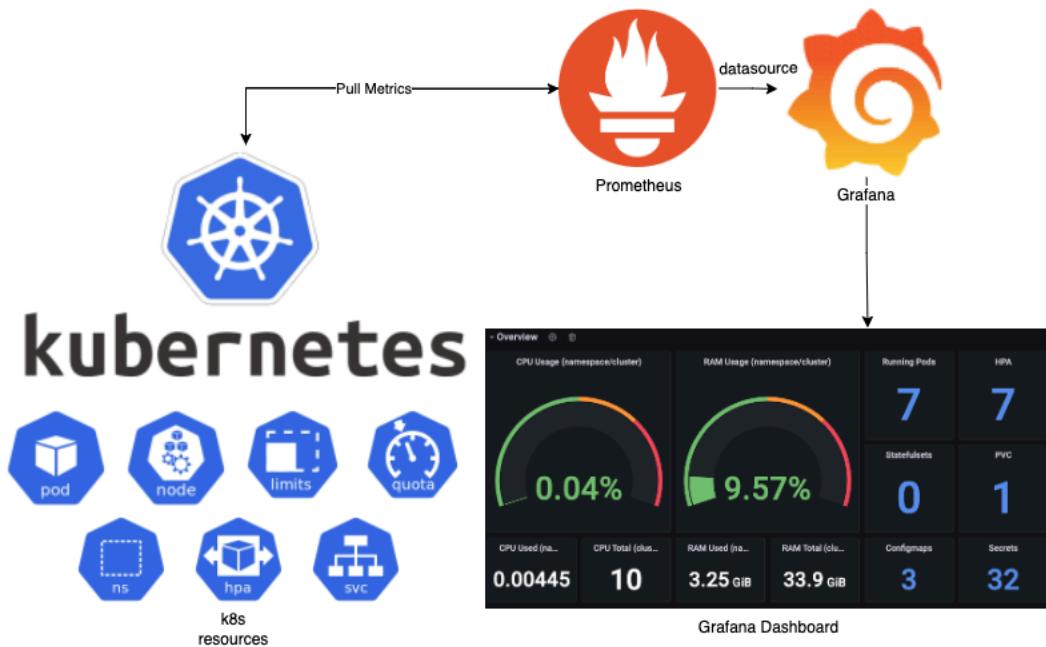


La monitorización de los elementos y servicios es fundamental para un funcionamiento óptimo de nuestra infraestructura. La monitorización proactiva permite prever posibles problemas antes de que ocurran.

A pesar de ser herramientas independientes, Prometheus y Grafana coexisten en simbiosis. Prometheus es una herramienta de código abierto que permite recopilar métricas del rendimiento de nuestro objetivo mientras que Grafana les dará forma y nos permitirá visualizarla en dashboards. Nos vamos a centrar en dos componentes fundamentales de Prometheus, el server y los exportadores. El servidor de Prometheus es el encargado de recoger los datos de las métricas mientras que los exportadores se encargan de recoger los datos de las fuentes que monitorizamos y exponerlos para que Prometheus los recoja.

Por otro lado, Grafana es una herramienta flexible a la hora de crear dashboards para interpretar la información que nos proporcionan las métricas. Grafana consta de varias partes:

- Server: gestiona la interfaz del usuario y la configuración general.
- Base de datos: almacena información sobre las métricas.
- Backend: proceso de información recolectada de diversas fuentes.
- Frontend: permite crear un dashboard personalizado por el usuario.



Esquema de relación entre Prometheus y Grafana.

3. Diseño del proyecto:

3.1 Esquema de la arquitectura:



MASTER: La máquina host, tendremos instalado Ansible, Prometheus, Grafana, Minikube y el Node-Exporter.

POD1: Contiene un contenedor con wordpress y contiene un wordpress-exporter que extrae métricas del wordpress y las manda al nodo principal para ser utilizadas por Prometheus y Grafana para la monitorización.

POD2: Contiene un contenedor con MySQL que albergará la base de datos para wordpress.

3.1 Tecnologías utilizadas:

En la siguiente tabla se podrá observar un resumen de las tecnologías utilizadas y su función en el proyecto actual:

Tecnología	Función en el proyecto
Proxmox	Creación y gestión de la máquina virtual.
Ubuntu Desktop	Sistema operativo del entorno.

Ansible	Automatización del despliegue.
Docker	Motor de los contenedores donde se ejecutan los pods.
Kubernetes	Orquestador de contenedores y gestión de pods/servicios.
Minikube	Clúster local de Kubernetes.
Wordpress	Aplicación web.
MySQL	Base de datos para Wordpress.
Prometheus	Recolección de métricas de la infraestructura.
Grafana	Visualización de métricas de manera gráfica.
Wordpress Exporter	Exportador de métricas de Wordpress.
Node Exporter	Exportador de métricas de los nodos, componente de Prometheus.

4. Desarrollo del proyecto:

4.1 Preparación del entorno:

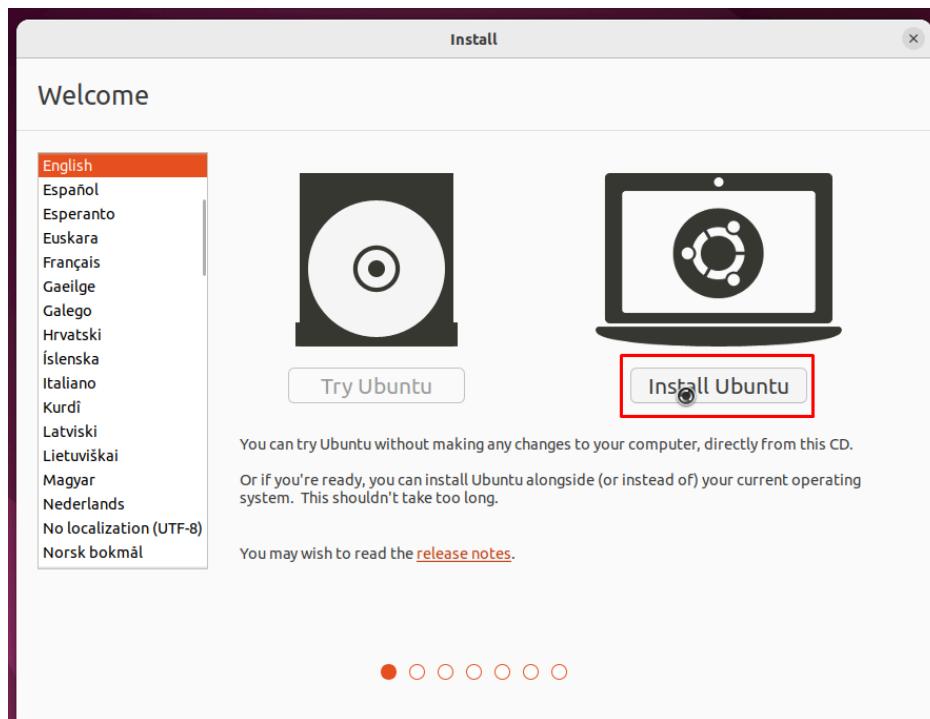
Creación de la máquina virtual en Proxmox:

The screenshot shows the Proxmox VE interface with the following details for a virtual machine:

- Summary:** Shows the VM ID (40108), name (TFG-ERIKA-ALESSIO), and node (pve2).
- Hardware:** Configuration details:
 - Memory: 7.81 GiB
 - Processors: 4 (1 sockets, 4 cores)
 - BIOS: Default (SeaBIOS)
 - Display: Default
 - Machine: Default (i440fx)
 - SCSI Controller: VirtIO SCSI
 - CD/DVD Drive (ide2): Backup.iso/ubuntu-22.04-desktop-amd64.iso, media=cdrom, size=3569294K
 - Hard Disk (scsi0): ZFSCLUSTER:vm-40108-disk-0, size=60G
 - Network Device (net0): virtio=22:5B:9C:4D:3F:95, bridge=vmbr0, firewall=1
- Console:** Shows memory usage (7.81 GiB).
- Cloud-Init:** Shows the configuration file path (/var/lib/cloud/instance/config).
- Options:** Shows the configuration file path (/etc/default/grub).
- Task History:** Shows a single task: "Install Ubuntu Server 22.04 LTS (64-bit) via CD/DVD" (Status: Success).
- Monitor:** Shows CPU usage (0.00% / 0.00% / 0.00%) and Memory usage (0.00% / 0.00%).
- Backup:** Shows a scheduled backup for the VM.
- Replication:** Shows a replication configuration.
- Snapshots:** Shows a single snapshot named 'Snapshot 1'.

Imagen: Información sobre el hardware de la máquina virtual.

Instalamos Ubuntu Desktop:



Proceso de instalación de Ubuntu.

Una vez instalado el sistema operativo, actualizamos los paquetes con `sudo apt update && sudo apt upgrade`.

```
usuariotfg@erika:~$ sudo apt update && sudo apt upgrade
[sudo] contraseña para usuariotfg:
Obj:1 http://security.ubuntu.com/ubuntu focal-security InRelease
Obj:2 http://es.archive.ubuntu.com/ubuntu focal InRelease
Obj:3 http://es.archive.ubuntu.com/ubuntu focal-updates InRelease
Obj:4 http://es.archive.ubuntu.com/ubuntu focal-backports InRelease
```

Instalamos ansible con `sudo apt install ansible`:

```
usuariotfg@erika:~$ sudo apt install ansible
```

Comprobamos que está correctamente instalado:

```
usuariotfg@erika:~$ ansible --version
ansible 2.9.6
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/usuariotfg/.ansible/plugins/modules',
  '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.8.10 (default, Mar 18 2025, 20:04:55) [GCC 9.4.0]
```

Creación de los ficheros YAML:

Para poder realizar el despliegue automatizado de la infraestructura, primero debemos configurar todos los ficheros YAML para cada parte. El primer script que diseñamos será para la instalación y configuración de Minikube, Docker, kubectl... Esto sentará las bases del proyecto.

Definimos el fichero Instalarentorno.yaml:

```
1 |   - name: Preparar entorno Kubernetes y Minikube
2 |     hosts: localhost
3 |     gather_facts: false
4 |     become: true
5 |
6 |   tasks:
7 |     - name: Actualizar paquetes
8 |       apt:
9 |         update_cache: yes
10 |
11 |     - name: Instalar Docker
12 |       apt:
13 |         name: docker.io
14 |         state: present
15 |
16 |     - name: Habilitar servicio Docker
17 |       systemd:
18 |         name: docker
19 |         enabled: yes
20 |
21 |     - name: Instalar curl
22 |       apt:
23 |         name: curl
24 |         state: present
25 |
26 |     - name: Descargar kubectl
27 |       shell: |
28 |         curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
29 |         install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
30 |
31 |     - name: Descargar Minikube
32 |       shell: |
33 |         curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
34 |         install minikube-linux-amd64 /usr/local/bin/minikube
35 |
36 |     - name: Añadir usuario al grupo docker
37 |       user:
38 |         name: usuario
39 |         groups: docker
40 |         append: yes
41 |
42 |     - name: Desactivar protección de archivos regulares
43 |       sysctl:
44 |         name: fs.protected_regular
45 |         value: 0
46 |         sysctl_set: yes
47 |         state: present
48 |
49 |     - name: Dar permisos a la carpeta tmp
50 |       file:
51 |         path: /tmp
52 |         mode: '0777'
53 |         recurse: yes
54 |
55 |
56 |
57 |
```

En primer lugar actualiza los paquetes, luego instala Docker y lo habilita. Posteriormente instalamos la herramienta curl y descargamos kubectl. Descarga Minikube y desactiva la protección de archivos regulares, ya que puede aparecer un error relacionado con esto.

Esta configuración desactiva una medida de seguridad del kernel que impide que los

usuarios eliminan o sobrescriban archivos regulares protegidos, especialmente desde carpetas como /tmp. Finalmente le da permisos a la carpeta tmp. Se usa –ask-become-pass para que pida la contraseña de sudo. Esto es necesario cuando en el inicio del fichero aparece “become: true”, ya que indica que varias tareas necesitarán permisos elevados de root. Así evitamos exponer la contraseña de root en texto plano dentro del playbook.

Ejecución:

```
usuari@usuari:~/proyecto$ ansible-playbook instalarentorno.yaml --ask-become-pass
BECOME password:
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'
PLAY [Preparar entorno Kubernetes y Minikube] ****
TASK [Actualizar paquetes] ****
changed: [localhost]
TASK [Instalar Docker] ****
ok: [localhost]
TASK [Habilitar servicio Docker] ****
ok: [localhost]
TASK [Instalar curl] ****
ok: [localhost]
TASK [Descargar kubectl] ****
[WARNING]: Consider using the get_url or uri module rather than running 'curl'. If you need to use command because get_url or uri is insufficient you can add 'warn: false' to this command task or set 'command_warnings=False' in ansible.cfg to get rid of this message.
changed: [localhost]
TASK [Descargar Minikube] ****
changed: [localhost]
TASK [Añadir usuario al grupo docker] ****
ok: [localhost]
TASK [Desactivar protección de archivos regulares] ****
[WARNING]: The value "0" (type int) was converted to "'0'" (type string). If this does not look like what you expect, quote the entire value to ensure it does not change.
changed: [localhost]
TASK [Dar permisos a la carpeta tmp] ****
changed: [localhost]

PLAY RECAP ****
localhost : ok=9    changed=5   unreachable=0   failed=0    skipped=0   rescued=0   ignored=0
```

Comprobamos que se ha instalado lo requerido:

```
usuariotfg@erika:/usr/local/bin$ kubectl version --client
Client Version: v1.32.3
Kustomize Version: v5.5.0
usuariotfg@erika:/usr/local/bin$ minikube version
minikube version: v1.35.0
commit: dd5d320e41b5451cdf3c01891bc4e13d189586ed-dirty
usuariotfg@erika:/usr/local/bin$ █
```

Luego, manualmente usamos el comando “newgrp docker” para poder aplicar el cambio de inmediato en el usuario y grupo Docker e inicializamos Minikube con: “minikube start –driver=docker –cpus=2 –memory=4096 –network-plugin=cni –cni=calico”.

Esto sirve para iniciar Minikube como clúster local de Kubernetes, usando docker como driver, con 2 cpus y 4gb de ram para el clúster. Se activa el plugin de CNI (Container Network Interface) y usa Calico como CNI, ya que permite la interconexión entre los pods y garantiza que el tráfico entre los contenedores sea el correcto para el proyecto.

Ejecución:

```
usuario@usuario:~/proyecto$ minikube start --driver=docker --cpus=2 --memory=4096 --network-plugin=cni --cni=calico
└─ minikube v1.36.0 on Ubuntu 22.04 (kvm/amd64)
  Using the docker driver based on user configuration
  With --network-plugin=cni, you will need to provide your own CNI. See --cni flag as a user-friendly alternative
  Using Docker driver with root privileges
  Starting "minikube" primary control-plane node in "minikube" cluster
  Pulling base image v0.0.47 ...
  Creating docker container (CPUs=2, Memory=4096MB) ...
  └─ Preparing Kubernetes v1.33.1 on Docker 28.1.1 ...
    └─ Generating certificates and keys ...
    └─ Booting up control plane ...
    └─ Configuring RBAC rules ...
  └─ Configuring Calico (Container Networking Interface) ...
  └─ Verifying Kubernetes components...
    └─ Using image gcr.io/k8s-minikube/storage-provisioner:v5
  Enabled addons: storage-provisioner, default-storageclass
  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Comprobación de que ha funcionado:

```
usuario@usuario:~/proyecto$ minikube status
minikube
  type: Control Plane
  host: Running
  kubelet: Running
  apiserver: Running
  kubeconfig: Configured

usuario@usuario:~/proyecto$ kubectl get nodes
NAME      STATUS   ROLES      AGE     VERSION
minikube  Ready    control-plane   39s    v1.33.1

usuario@usuario:~/proyecto$ kubectl config current-context
minikube
```

El contexto debe ser minikube. Un contexto es una configuración que le dice a kubectl a qué clúster conectarse, el usuario y en qué namespace trabajar. Los namespaces son divisiones lógicas del mismo clúster para aislar recursos y organizarlos mejor. Dentro de los que aparecen, el proyecto está desplegado en el default.

```
usuario@usuario:~/proyecto$ kubectl get namespaces
NAME        STATUS   AGE
default     Active   7d4h
kube-node-lease  Active   7d4h
kube-public   Active   7d4h
kube-system   Active   7d4h
```

4.2. Despliegue con Ansible y YAML

Una vez comprobamos que funciona, comenzamos a desarrollar todos los ficheros .yaml para despegar Wordpress, Mysql y los exporters.

Para simplificar, utilizaremos la herramienta kustomize, que nos permite agrupar recursos bajo el mismo fichero kustomization.yaml y nos permite modificar los recursos sin necesidad de tocar los yaml originales. Este fichero contendrá los resources, que son los ficheros yaml de los recursos que vamos a necesitar en nuestra estructura.

Dentro de nuestro kustomization.yaml incluimos lo siguiente:

```
resources:
  - mysqlsecret.yaml
  - mysqlvolume.yaml
  - mysqldeployment.yaml
  - mysqlservice.yaml
  - wordpressvolume.yaml
  - wordpressdeployment.yaml
  - wordpressexporter.yaml
  - wordpressservice.yaml
```

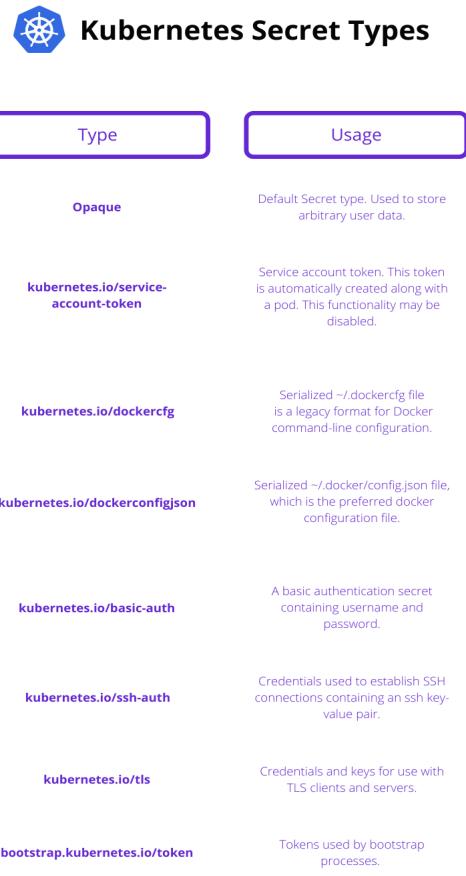
1. Para MySQL:
 - a. mysqlsecret.yaml
 - b. mysqlvolume.yaml
 - c. mysqlservice.yaml
 - d. mysqldeployment.yaml
2. Para Wordpress:
 - a. wordpressvolume.yaml
 - b. wordpressservice.yaml
 - c. wordpressdeployment.yaml
 - d. wordpressexporter.yaml

A continuación, se explicará la estructura y función de cada una de los ficheros.

1a. Mysqlsecret.yaml:

```
1  apiVersion: v1
2  kind: Secret
3  metadata:
4    name: mysql-pass
5  type: Opaque
6  stringData:
7    password: erika
```

Este fichero sirve para crear una contraseña para MySql que será “erika”. El objeto será de tipo secreto (utilizado para guardar datos sensibles como contraseñas). El metadato nombra al Secret como mysql-pass para poder hacer referencia más adelante. El tipo es opaco, ya que es el utilizado para datos arbitrarios como contraseñas. Hay otros tipos:



Utilizamos este fichero para no tener que escribir la contraseña en texto plano en el deployment directamente, si no que añadimos un paso más para garantizar nuestra seguridad. En resumen, este fichero creará un Secret llamado mysql-pass que contendrá un campo password con el valor “erika”.

1b. Mysqlvolume.yaml:

```
1  apiVersion: v1
2  kind: PersistentVolumeClaim
3  metadata:
4    name: mysql-pvc
5    labels:
6      app: wordpress
7  spec:
8    accessModes:
9      - ReadWriteOnce
10   resources:
11     requests:
12       storage: 5Gi
13
```

Nuestro objetivo con este fichero es crear un volumen persistente para Mysql de 5Gb, que será de lectura y escritura. Lo hacemos de tipo persistente para evitar la pérdida de información en caso de que se borre el pod, mantendremos los datos de Mysql. Se le pone la etiqueta de app: wordpress para que luego podamos organizar e identificarlo mejor.

1c. Mysqlservice.yaml:

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: wordpress-mysql
5    labels:
6      app: wordpress
7  spec:
8    type: NodePort
9    selector:
10      app: wordpress
11      tier: mysql
12    ports:
13      - port: 3306
14        targetPort: 3306
15        nodePort: 31313
16
```

Este fichero nos indica que es de tipo servicio, lo cual significa que vamos a exponer una aplicación. Incluye la etiqueta para la app de wordpress para que todas estén identificadas por wordpress (todas pertenecen al ecosistema de Wordpress). El port 3306 es el que se usa por defecto internamente para Mysql, el target port es al que se debe redirigir el tráfico dentro del pod (también 3306) y el Nodeport, es el que exponemos al exterior.

El NodePort nos facilita que el puerto que nosotros le añadamos, será el expuesto para que desde el nodo podamos conectarnos desde este puerto a la base de datos. Solo conectará a los pods que tengan como app: wordpress y tier: mysql. Gracias a esto, tendremos conectividad entre el nodo y los pods. El port 3306 es el que se usa por defecto internamente para Mysql, el target port es al que se debe redirigir el tráfico dentro del pod (también 3306) y el Nodeport, es el que exponemos al exterior.

1d. Mysqldeployment.yaml:

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: wordpress-mysql
5    labels:
6      app: wordpress
7  spec:
8    selector:
9      matchLabels:
10     app: wordpress
11     tier: mysql
12   strategy:
13     type: Recreate
14   template:
15     metadata:
16       labels:
17         app: wordpress
18         tier: mysql
19     spec:
20       containers:
21         - name: mysql
22           image: mysql:5.7
23           args:
24             - --default-authentication-plugin=mysql_native_password
25           env:
26             - name: MYSQL_DATABASE
27               value: wordpress
28             - name: MYSQL_ROOT_PASSWORD
29               valueFrom:
30                 secretKeyRef:
31                   name: mysql-pass
32                   key: password
33           ports:
34             - containerPort: 3306
35               name: mysql
36           volumeMounts:
37             - name: mysql-persistent-storage
38               mountPath: /var/lib/mysql
39           volumes:
40             - name: mysql-persistent-storage
41               persistentVolumeClaim:
42                 claimName: mysql-pvc
43
```

En este caso se define un deployment llamado wordpress-mysql. Se le asocia también la etiqueta wordpress. Este deployment va a administrar los pods que contengan la etiqueta app: wordpress y tier: mysql con la estrategia Recreate. Esto significa que en caso de actualización, los pods antiguos se eliminan y se sustituyen por nuevos.

La parte de Pod template hace referencia a la plantilla con la que se crearán los Pods. Se le pone etiqueta para que pueda reconocer el deployment a posteriori.

Este pod estará formado por un contenedor:

- Mysql: utilizará la imagen de Docker de mysql y se configura para que use el plugin de autenticación mysql_native_password para después usar la que almacenamos previamente en Secrets.
- Como variable de entorno le pasamos que la base de datos se llame wordpress y que la contraseña de root sea la que creamos anteriormente como Secreto llamado mysql-pass. Indicamos el puerto por defecto de mysql que es el 3306. Luego, le añadimos el volumen persistente que creamos anteriormente que estará montado en /var/lib/mysql.

2a. wordpressvolume.yaml:

```
1  apiVersion: v1
2  kind: PersistentVolumeClaim
3  metadata:
4    name: wp-pvc
5    labels:
6      app: wordpress
7  spec:
8    accessModes:
9      - ReadWriteOnce
10   resources:
11     requests:
12       storage: 5Gi
13
```

Este fichero tiene la misma estructura que el volumen utilizado para Mysql. La única diferencia es que se llama wp-pvc. El resto, como el tamaño, etiqueta y modo de acceso son idénticos.

2b. wordpressservice.yaml:

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: wordpress
5    labels:
6      app: wordpress
7  spec:
8    type: NodePort
9    ports:
10      - protocol: TCP
11        port: 80
12        nodePort: 31165
13    selector:
14      app: wordpress
15      tier: frontend
```

Este servicio llamado wordpress, cuya etiqueta es homónima, expondrá al exterior el puerto 31165 por ser de tipo NodePort y redirigirá el tráfico de los pods cuya etiqueta app sea wordpress y tier frontend al puerto 80. De esta forma habilitamos el acceso desde el exterior al servicio web de Wordpress.

2c. wordpressexporter.yaml:

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: wordpress-exporter
5    labels:
6      app: wordpress
7      tier: wp
8  spec:
9    type: NodePort
10   selector:
11     app: wordpress
12   ports:
13     - name: exporter
14       protocol: TCP
15       port: 11011
16       targetPort: 11011
17       nodePort: 31222
```

Este servicio llamado wordpress-exporter, como su propio nombre indica, exportará las métricas de Wordpress en el puerto 31222 de cara al exterior y redirigirá el tráfico al puerto 11011 del tráfico que vaya a los recursos con etiquetas de app wordpress y de tier wp.

2d. wordpressdeployment.yaml:

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: wordpress
5    labels:
6      app: wordpress
7  spec:
8    selector:
9      matchLabels:
10     app: wordpress
11     tier: frontend
12   strategy:
13     type: Recreate
14   template:
15     metadata:
16       labels:
17         app: wordpress
18         tier: frontend
19   spec:
20     containers:
21       - image: wordpress
22         name: wordpress
23         env:
24           - name: WORDPRESS_DB_HOST
25             value: wordpress-mysql
26           - name: WORDPRESS_DB_NAME
27             value: wordpress
28           - name: WORDPRESS_DB_USER
29             value: root
30           - name: WORDPRESS_DB_PASSWORD
31             valueFrom:
32               secretKeyRef:
33                 name: mysql-pass
34                 key: password
35     ports:
36       - containerPort: 80
37         name: wordpress
38     volumeMounts:
39       - name: wordpress-persistent-storage
40         mountPath: /var/www/html
```

```
41      - name: wp-exporter
42        image: ghcr.io/aorfanos/wordpress-exporter/wordpress-exporter:v0.0.8
43        ports:
44          - containerPort: 11011
45            name: wp-exporter
46        args:
47          - --host
48          - http://wordpress:80
49          - --auth.basic
50          - "false"
51      volumes:
52        - name: wordpress-persistent-storage
53        persistentVolumeClaim:
54          claimName: wp-pvc
```

Este fichero define un deployment llamado wordpress. El selector se va a centrar en los recursos que tengan como etiqueta app wordpress y tier frontend. La estrategia será Recreate, como hemos mencionado anteriormente, se eliminarán los pods en cuanto haya una actualización, sustituyéndolos por nuevos pods. Se crearán 2 contenedores:

1. Contenedor Wordpress: con la imagen de wordpress. Como variables de entorno le pasamos el host de la base de datos, el nombre de la base de datos, el usuario y la contraseña. La contraseña la recupera del Secret creado anteriormente. El tráfico se dirige al puerto 80 y se montará el volumen de 5Gb creado anteriormente en la ruta /var/www/html.
2. Contenedor wordpress exporter: cuya imagen proviene del repositorio <https://github.com/aorfanos/wordpress-exporter/pkgs/container/wordpress-exporter%2Fwordpress-exporter> con el puerto 11011. Como argumentos le indicamos que como host coja al wordpress y que no necesite autorización para acceder. Además, se monta un volumen persistente igual que los anteriores.

Una vez explicados todos los ficheros, procedemos a su despliegue. Mediante el comando “kubectl apply -k ./” desplegamos todos los recursos que contiene kustomization.yaml.

Ejecución:

```
usuario@usuario:~/proyecto$ kubectl apply -k .
secret/mysql-pass created
service/wordpress created
service/wordpress-exporter created
service/wordpress-mysql created
persistentvolumeclaim/mysql-pvc created
persistentvolumeclaim/wp-pvc created
deployment.apps/wordpress created
deployment.apps/wordpress-mysql created
```

Comprobamos cómo se van creando los pods:

```
deployment.apps/wordpress-mysql unchanged
usuario@usuario:~/proyecto$ kubectl get pods
NAME                      READY   STATUS            RESTARTS   AGE
wordpress-76cbb8cc44-kh8hs   0/2    ContainerCreating   0          2s
wordpress-mysql-59dc4b6cdd-574xm  1/1    Running           0          11m
usuario@usuario:~/proyecto$
```

Y si lo volvemos a ejecutar pasado unos segundos, veremos que ya se ha creado todo:

```
usuario@usuario:~/proyecto$ kubectl get pods
NAME                      READY   STATUS            RESTARTS   AGE
wordpress-76cbb8cc44-kh8hs   2/2    Running           0          38s
wordpress-mysql-59dc4b6cdd-574xm  1/1    Running           0          12m
usuario@usuario:~/proyecto$
```

Comprobamos toda la infraestructura del proyecto:

```
usuario@usuario:~/proyecto$ kubectl get pods
NAME                      READY   STATUS            RESTARTS   AGE
wordpress-76cbb8cc44-kh8hs   0/2    ContainerCreating   0          2s
wordpress-mysql-59dc4b6cdd-574xm  1/1    Running           0          11m
usuario@usuario:~/proyecto$ kubectl get pods
NAME                      READY   STATUS            RESTARTS   AGE
wordpress-76cbb8cc44-kh8hs   2/2    Running           0          38s
wordpress-mysql-59dc4b6cdd-574xm  1/1    Running           0          12m
usuario@usuario:~/proyecto$ kubectl get all
NAME                      READY   STATUS            RESTARTS   AGE
pod/wordpress-76cbb8cc44-kh8hs   2/2    Running           0          51s
pod/wordpress-mysql-59dc4b6cdd-574xm  1/1    Running           0          12m

NAME                TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)        AGE
service/kubernetes  ClusterIP   10.96.0.1     <none>          443/TCP       18m
service/wordpress   NodePort    10.104.172.19  <none>          80:31165/TCP  12m
service/wordpress-exporter  NodePort    10.103.11.67  <none>          11011:31222/TCP 12m
service/wordpress-mysql  NodePort    10.102.118.112 <none>          3306:31313/TCP  12m

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/wordpress   1/1      1           1           12m
deployment.apps/wordpress-mysql  1/1      1           1           12m

NAME                      DESIRED   CURRENT   READY   AGE
replicaset.apps/wordpress-6cccd54b4b5  0         0         0         12m
replicaset.apps/wordpress-76cbb8cc44  1         1         1         51s
replicaset.apps/wordpress-mysql-59dc4b6cdd  1         1         1         12m
usuario@usuario:~/proyecto$
```

El primer apartado nos indica que tenemos dos pods corriendo, uno de wordpress y otro con la base de datos MySQL.

El segundo apartado nos indica los servicios que tenemos activos, su ip interna (Cluster IP) y los puertos externos, además del tiempo que llevan funcionando. Por ejemplo, para acceder al servicio Wordpress habría que poner la ip de minikube (con el comando “minikube ip”) y el puerto que exponemos (31165). Los servicios se crean como NodePort para asegurar su acceso desde el exterior.

El tercer apartado trata de los deployments. Como hemos configurado anteriormente, tenemos 2 deployments, uno para wordpress y otro para MySQL. Están listos, actualizados y disponibles, por lo que los pods se están manteniendo correctamente.

El último apartado trata de los replicsets. Aseguran que el número de pods que hayamos configurado en el deployment estén funcionando correctamente. Por lo que actualmente hay 1 deseado, 1 creado y 1 funcionando.

```
usuario@usuario:~/proyecto$ minikube ip
192.168.49.2
```

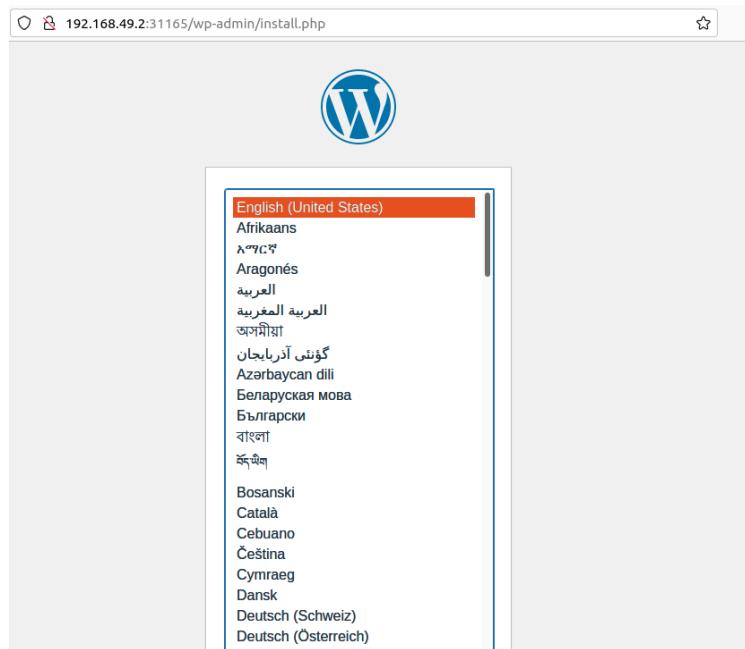
Para saber qué dirección debemos usar para acceder a nuestros servicios, usamos el comando “minikube service wordpress –url” por ejemplo:

```
usuario@usuario:~/proyecto$ minikube service wordpress --url  
http://192.168.49.2:31165
```

Si quisiésemos usar la dirección 127.0.0.1:80, deberíamos estar dentro del propio contenedor de Wordpress.

4.3. Configuración de WordPress

Como hemos comprobado, ya están todos los pods funcionando. Accedemos a Wordpress poniendo la ip de Minikube y el puerto de Wordpress (192.168.49.2:31165):



Configuramos nuestro Wordpress:

192.168.49.2:31165/wp-admin/install.php?step=1

Este es el简单o proceso de instalacióo de WordPress en cinco minutos. Simplemente completa la información siguiente y estarás a punto de usar la más enriquecedora y potente plataforma de publicación personal del mundo.

Información necesaria

Por favor, proporciona la siguiente información. No te preocupes, siempre podrás cambiar estos ajustes más tarde.

Título del sitio	proyectoerika
Nombre de usuario	erika
Contraseña	erika Muy débil
Ocultar	
Confirmar la contraseña	<input checked="" type="checkbox"/> Confirma el uso de una contraseña débil.
Tu correo electrónico	alessioerika1998@gmail.com
Visibilidad en los motores de búsqueda	<input checked="" type="checkbox"/> Pedir a los motores de búsqueda que no indexen este sitio Depende de los motores de búsqueda atender esta petición o no.

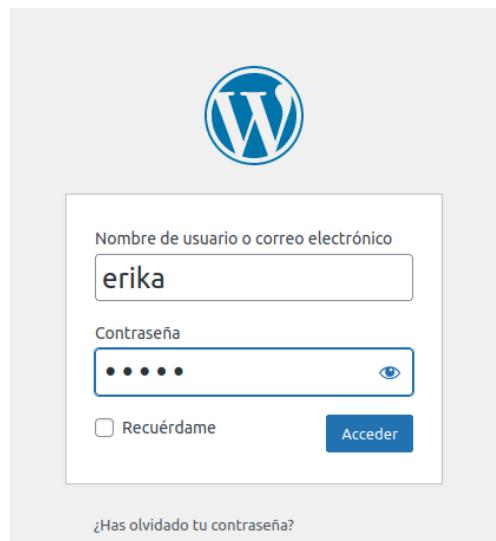
Importante: Necesitas esta contraseña para acceder. Por favor, guárdala en un lugar seguro.

Nota: La contraseña está expuesta con fines educativos, no se debe exponer en un entorno real.

Vamos a comprobar que se ha configurado correctamente:



Nos logueamos:

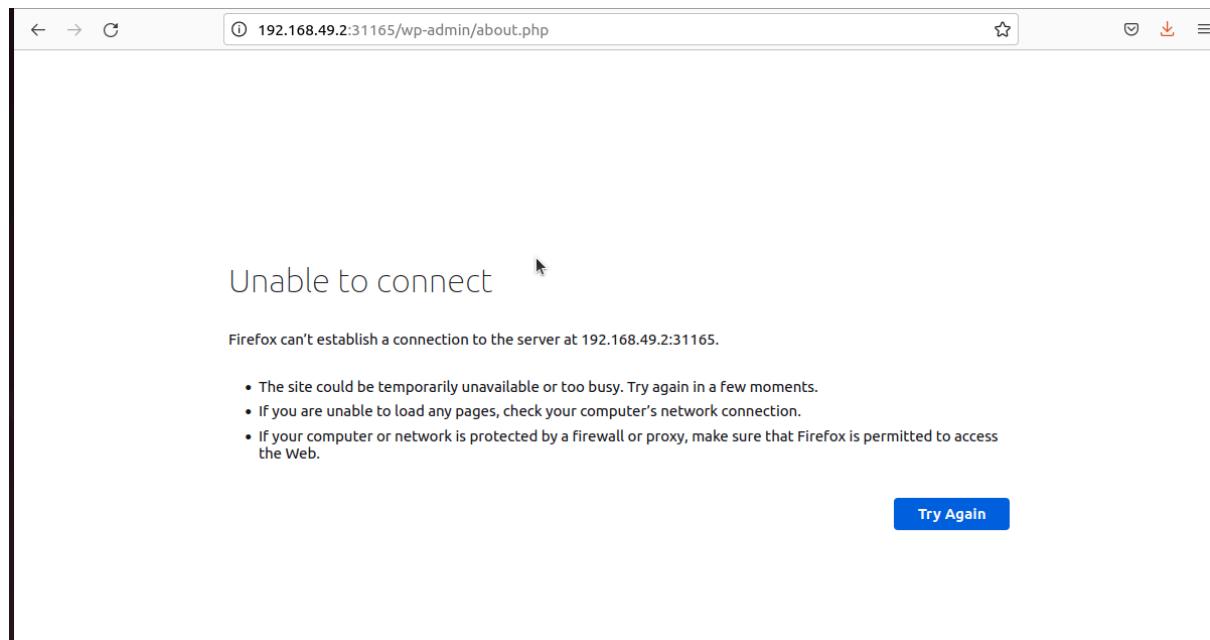


Y funciona correctamente:

Ahora, probamos a borrar el pod de Wordpress:

```
usuario@usuario:~/proyecto$ kubectl delete pod wordpress-76cbb8cc44-kh8hs
pod "wordpress-76cbb8cc44-kh8hs" deleted
```

Accedemos a la página y vemos que ha dejado de funcionar:



Comprobamos el estado de los pods y vemos que se ha vuelto a levantar Wordpress automáticamente gracias al deployment definido previamente:

```
usuario@usuario:~/proyecto$ kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
wordpress-76cbb8cc44-tk52v   2/2     Running   0          37s
wordpress-mysql-59dc4b6cdd-574xm   1/1     Running   0          16m
```

Accedemos nuevamente a la página de Wordpress y vemos que está funcionando:



Esta es la ventaja que nos proporciona Kubernetes respecto al resto de orquestadores.

4.4. Integración de Prometheus:

Para la instalación de Prometheus definiremos un fichero instalarprometheus.yaml que contendrá lo siguiente:

```
1 - name: Configuracion de Prometheus
2   become: true
3   become_method: sudo
4   hosts: localhost
5
6   tasks:
7     - name: Crear usuario prometheus
8       user:
9         name: prometheus
10        shell: /bin/false
11        createhome: no
12
13    - name: Crear directorios de Prometheus
14      file:
15        path: "{{ item }}"
16        state: directory
17        owner: prometheus
18        group: prometheus
19        with_items:
20          - /etc/prometheus
21          - /var/lib/prometheus
22
23    - name: Descargar Prometheus directamente en `/usr/local/bin`
24      get_url:
25        url: https://github.com/prometheus/prometheus/releases/download/v2.43.0/prometheus-2.43.0.linux-amd64.tar.gz
26        dest: /usr/local/bin/prometheus.tar.gz
27
28    - name: Extraer Prometheus en `/usr/local/bin`
29      unarchive:
30        src: /usr/local/bin/prometheus.tar.gz
31        dest: /usr/local/bin
32        remote_src: yes
33        extra_opts:
34          - --strip-components=1
35
36
37    - name: Eliminar archivo comprimido
38      file:
39        path: /usr/local/bin/prometheus.tar.gz
40        state: absent
41
```

```
41      - name: Copiar archivo de configuración
42        copy:
43          src: /home/usuario/proyecto/prometheus.yaml
44          dest: /etc/prometheus/prometheus.yaml
45          owner: prometheus
46          group: prometheus
47          remote_src: yes
48
49
50      - name: Crear archivo de servicio systemd para Prometheus
51        copy:
52          dest: /etc/systemd/system/prometheus.service
53          content: |
54            [Unit]
55            Description=Prometheus
56            Wants=network-online.target
57            After=network-online.target
58
59            [Service]
60            User=prometheus
61            Group=prometheus
62            Type=simple
63            ExecStart=/usr/local/bin/prometheus \
64              --config.file /etc/prometheus/prometheus.yaml \
65              --storage.tsdb.path /var/lib/prometheus/ \
66              --web.console.templates=/etc/prometheus/consoles \
67              --web.console.libraries=/etc/prometheus/console_libraries
68
69            [Install]
70            WantedBy=multi-user.target
71
72      - name: Recargar systemd
73        systemd:
74          daemon_reload: yes
75
76      - name: Iniciar servicio Prometheus
77        systemd:
78          name: prometheus
79          state: started
80          enabled: yes
81
82      - name: Crear usuario `node_exporter`
83        user:
84          name: node_exporter
```

```

84      name: node_exporter
85      shell: /bin/false
86      createhome: no
87      become: yes
88
89 - name: Descargar y extraer `node_exporter`
90   get_url:
91     url: https://github.com/prometheus/node_exporter/releases/download/v1.5.0/node_exporter-1.5.0.linux-amd64.tar.gz
92   dest: /usr/local/bin/node_exporter.tar.gz
93   become: yes
94
95 - name: Extraer `node_exporter` en `/usr/local/bin`
96   unarchive:
97     src: /usr/local/bin/node_exporter.tar.gz
98     dest: /usr/local/bin
99     remote_src: yes
100    extra_opts:
101      - --strip-components=1
102    become: yes
103
104 - name: Eliminar archivo comprimido de `node_exporter`
105   file:
106     path: /usr/local/bin/node_exporter.tar.gz
107     state: absent
108   become: yes
109
110 - name: Crear archivo de servicio `node_exporter`
111   copy:
112     dest: /etc/systemd/system/node_exporter.service
113     content: |
114       [Unit]
115       Description=Node Exporter
116       Wants=network-online.target
117       After=network-online.target
118
119       [Service]
120       User=node_exporter
121       Group=node_exporter
122       Type=simple
123       ExecStart=/usr/local/bin/node_exporter
124
125       [Install]
126       WantedBy=multi-user.target
127
128
129 - name: Arrancar y habilitar `node_exporter`
130   service:
131     name: node_exporter
132     state: started
133     enabled: yes
134
135

```

En primer lugar vamos a definir dónde se instalará Prometheus, que será en el localhost y se aplicarán permisos de root. Se crea un usuario sin shell ni home que se llamará prometheus. También se crearán directorios de prometheus (/etc/prometheus y /var/lib/prometheus), en uno irá el archivo de configuración y en el otro, los datos de las métricas. Descarga y descomprime Prometheus del repositorio oficial. La opción strip-components=1 permite que Prometheus se descomprima en /usr/local/bin directamente y no dentro de “prometheus-2.43.0.linux-amd64”. Se elimina el archivo comprimido.

Posteriormente, se copia el archivo de configuración prometheus.yaml (se explicará más adelante) a la carpeta de /etc/prometheus. Se define un servicio del sistema (systemd) que se ejecutará como prometheus y usará los datos que le proporcionamos en el fichero y estará listo tras el arranque. Se inicia el servicio indicado anteriormente.

En segundo lugar, se crea un usuario node_exporter y descargamos y extraemos node_exporter del repositorio oficial. Lo movemos a /usr/local/bin para que pueda ser ejecutado y establecemos permisos (rwx para el propietario, rw para el grupo y otros).

Al igual que con Prometheus, creamos un servicio del sistema con node_exporter, que también se iniciará tras el arranque.

El archivo prometheus.yaml mencionado anteriormente, contiene lo siguiente:

```
1   global:
2     scrape_interval: 10s
3     evaluation_interval: 10s
4
5   scrape_configs:
6     - job_name: 'node_exporter'
7       scrape_interval: 5s
8
9       static_configs:
10      - targets: ['localhost:9100']
11
12     - job_name: wordpress_exporter
13       honor_timestamps: true
14       scrape_interval: 5s
15
16       metrics_path: /metrics
17       scheme: http
18       static_configs:
19         - targets: ["192.168.49.2:31222"]
20
21
22
```

Lo que se define en este fichero es que globalmente cada 10 segundos recolecte métricas de los targets y se evalúen las reglas que haya cada 10 segundos también.

Del bloque node_exporter, va a hacer scraping cada 5 segundos y el target es en el localhost:9100.

Por último, el bloque de wordpress_exporter recogerá métricas cada 5 segundos, usando las marcas de tiempo del exporter. Las métricas del servicio estarán en /metrics y se define la ip diana: 192.168.49.2:31222.

Procedemos a ejecutar el playbook:

Erika Alessio

```
usuario@usuario:~/proyecto$ ansible-playbook instalarprometheus.yaml --ask-become-pass
BECOME password:
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [Configuración de Prometheus] ****
TASK [Gathering Facts] ****
ok: [localhost]

TASK [Crear usuario prometheus] ****
ok: [localhost]

TASK [Crear directorios de Prometheus] ****
ok: [localhost] => (item=/etc/prometheus)
ok: [localhost] => (item=/var/lib/prometheus)

TASK [Descargar Prometheus directamente en `/usr/local/bin`] ****
changed: [localhost]

TASK [Extraer Prometheus en `/usr/local/bin`] ****
ok: [localhost]

TASK [Eliminar archivo comprimido] ****
changed: [localhost]

TASK [Copiar archivo de configuración] ****
changed: [localhost]

TASK [Crear archivo de servicio systemd para Prometheus] ****
changed: [localhost]

TASK [Recargar systemd] ****
ok: [localhost]

TASK [Iniciar servicio Prometheus] ****
changed: [localhost]

TASK [Crear usuario `node_exporter`] ****
changed: [localhost]

TASK [Descargar y extraer `node_exporter`] ****
changed: [localhost]

TASK [Extraer `node_exporter` en `/usr/local/bin`] ****
changed: [localhost]

TASK [Eliminar archivo comprimido de `node_exporter`] ****
changed: [localhost]

TASK [Crear archivo de servicio `node_exporter`] ****
changed: [localhost]

TASK [Arrancar y habilitar `node_exporter`] ****
changed: [localhost]

TASK [Configurar archivo de Prometheus] ****
ok: [localhost]

PLAY RECAP ****
localhost          : ok=17   changed=11   unreachable=0   failed=0   skipped=0   rescued=0   ignored=0

usuario@usuario:~/proyecto$
```

Comprobamos que nuestros servicios del sistema funcionan:

```
usuario@usuario:~/proyecto$ systemctl status prometheus
● prometheus.service - Prometheus
    Loaded: loaded (/etc/systemd/system/prometheus.service; enabled; vendor preset: enabled)
      Active: active (running) since Sun 2025-05-25 14:17:18 CEST; 1 week 0 days ago
        Main PID: 35516 (prometheus)
           Tasks: 10 (limit: 9215)
         Memory: 181.7M
            CPU: 25min 537ms
          CGroup: /system.slice/prometheus.service
                  └─35516 /usr/local/bin/prometheus --config.file /etc/prometheus/prometheus.yaml --storage.tsdb.path /var/lib/prometheus

jun 01 17:00:03 usuario prometheus[35516]: ts=2025-06-01T15:00:03.437Z caller=compact.go:519 level=info component=tsdb msg="write b"
jun 01 17:00:03 usuario prometheus[35516]: ts=2025-06-01T15:00:03.441Z caller=head.go:1269 level=info component=tsdb msg="Head GC c"
jun 01 19:00:02 usuario prometheus[35516]: ts=2025-06-01T17:00:02.332Z caller=compact.go:519 level=info component=tsdb msg="write b"
jun 01 19:00:02 usuario prometheus[35516]: ts=2025-06-01T17:00:02.342Z caller=head.go:1269 level=info component=tsdb msg="Head GC c"
jun 01 19:00:02 usuario prometheus[35516]: ts=2025-06-01T17:00:02.345Z caller=checkpoint.go:100 level=info component=tsdb msg="Create"
jun 01 19:00:02 usuario prometheus[35516]: ts=2025-06-01T17:00:02.449Z caller=head.go:1241 level=info component=tsdb msg="WAL check"
jun 01 19:00:02 usuario prometheus[35516]: ts=2025-06-01T17:00:02.559Z caller=compact.go:460 level=info component=tsdb msg="compact"
jun 01 19:00:02 usuario prometheus[35516]: ts=2025-06-01T17:00:02.560Z caller=db.go:1548 level=info component=tsdb msg="Deleting ob"
jun 01 19:00:02 usuario prometheus[35516]: ts=2025-06-01T17:00:02.564Z caller=db.go:1548 level=info component=tsdb msg="Deleting ob"
jun 01 19:00:02 usuario prometheus[35516]: ts=2025-06-01T17:00:02.569Z caller=db.go:1549 level=info component=tsdb msg="Deleting ob"
```

```
usuario@usuario:~/proyecto$ systemctl status node_exporter
● node_exporter.service - Node Exporter
   Loaded: Loaded (/etc/systemd/system/node_exporter.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2025-05-25 14:28:49 CEST; 1 week 0 days ago
     Main PID: 39851 (node_exporter)
       Tasks: 5 (limit: 9215)
      Memory: 9.2M
        CPU: 51min 48.014s
       CGroup: /system.slice/node_exporter.service
               └─39851 /usr/local/bin/node_exporter

may 25 14:28:49 usuario node_exporter[39851]: ts=2025-05-25T12:28:49.835Z caller=node_exporter.go:117 level=info collector=thermal>
may 25 14:28:49 usuario node_exporter[39851]: ts=2025-05-25T12:28:49.835Z caller=node_exporter.go:117 level=info collector=time
may 25 14:28:49 usuario node_exporter[39851]: ts=2025-05-25T12:28:49.835Z caller=node_exporter.go:117 level=info collector=timex
may 25 14:28:49 usuario node_exporter[39851]: ts=2025-05-25T12:28:49.835Z caller=node_exporter.go:117 level=info collector=udp_queue
may 25 14:28:49 usuario node_exporter[39851]: ts=2025-05-25T12:28:49.835Z caller=node_exporter.go:117 level=info collector=uname
may 25 14:28:49 usuario node_exporter[39851]: ts=2025-05-25T12:28:49.835Z caller=node_exporter.go:117 level=info collector=vmstat
may 25 14:28:49 usuario node_exporter[39851]: ts=2025-05-25T12:28:49.835Z caller=node_exporter.go:117 level=info collector=xfs
may 25 14:28:49 usuario node_exporter[39851]: ts=2025-05-25T12:28:49.835Z caller=node_exporter.go:117 level=info collector=zfs
may 25 14:28:49 usuario node_exporter[39851]: ts=2025-05-25T12:28:49.835Z caller=tls_config.go:232 level=info msg="listening on " ad>
may 25 14:28:49 usuario node_exporter[39851]: ts=2025-05-25T12:28:49.835Z caller=tls_config.go:235 level=info msg="TLS is disabled.">
lines 1-20/20 (END)
```

Ahora, accedemos a Prometheus (<http://localhost:9090>) y posteriormente en Status → Targets para ver las métricas recopiladas:

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9100/metrics	UP	instance="localhost:9100" job="node_exporter"	360.000ms ago	20.638ms	

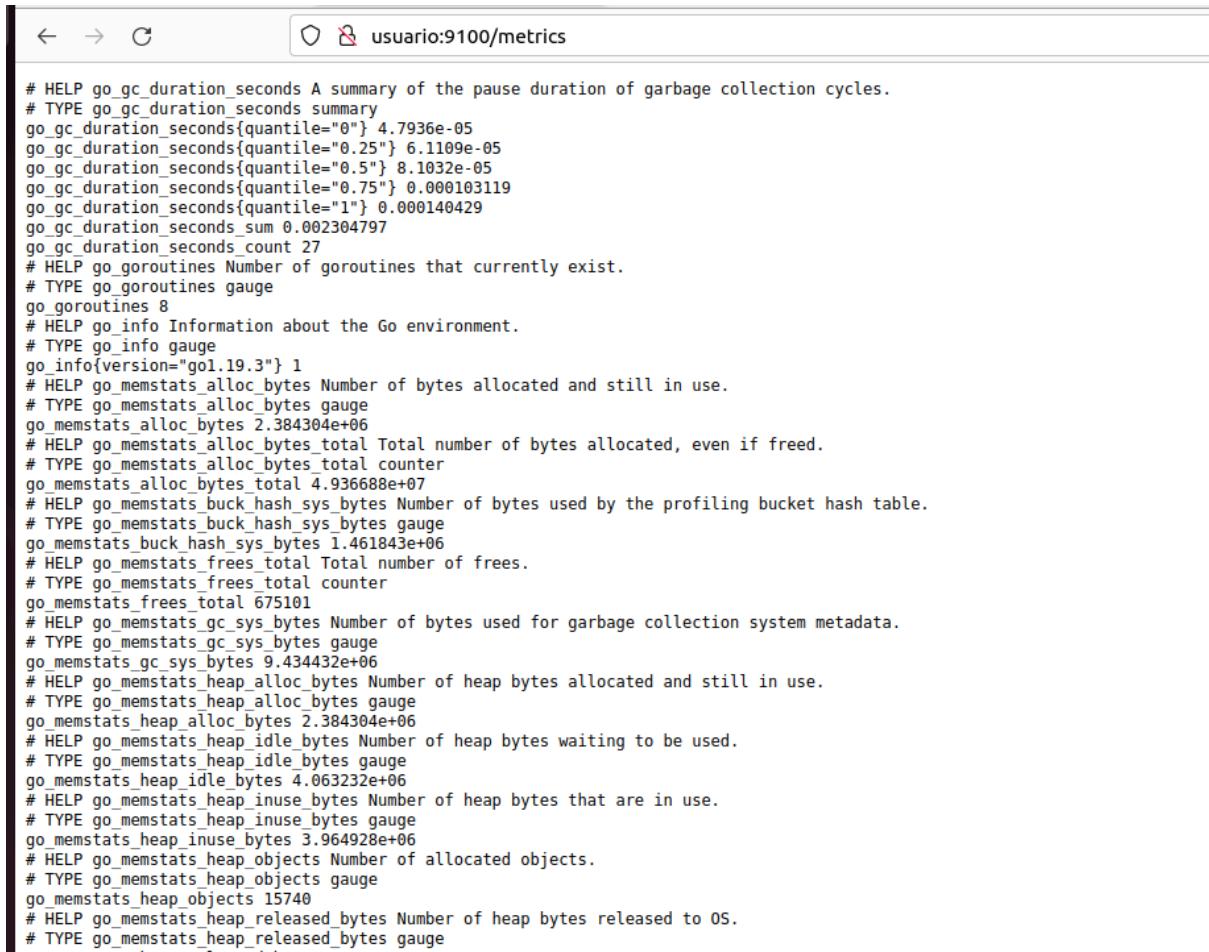
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://192.168.49.2:31222/metrics	UP	instance="192.168.49.2:31222" job="wordpress_exporter"	4.216s ago	278.111ms	

Entramos en los scrapeos, por ejemplo, el de Wordpress:

```
process_start_time_seconds 1.74817269852e+09
# HELP process_virtual_memory_max_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 7.37583104e+08
# HELP process_virtual_memory_max_bytes Maximum amount of virtual memory available in bytes.
# TYPE process_virtual_memory_max_bytes gauge
process_virtual_memory_max_bytes 1.8446744073709552e+19
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 210
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
# HELP wordpress_category_count WordPress category count
# TYPE wordpress_category_count gauge
wordpress_category_count{instance="http://wordpress:80"} 1
# HELP wordpress_comment_count WordPress comments count
# TYPE wordpress_comment_count gauge
wordpress_comment_count{instance="http://wordpress:80"} 1
# HELP wordpress_media_count WordPress media files count
# TYPE wordpress_media_count gauge
wordpress_media_count{instance="http://wordpress:80"} 0
# HELP wordpress_page_count WordPress pages count
# TYPE wordpress_page_count gauge
wordpress_page_count{instance="http://wordpress:80"} 1
# HELP wordpress_plugin_count WordPress plugin count
# TYPE wordpress_plugin_count gauge
wordpress_plugin_count{instance="http://wordpress:80"} -1
# HELP wordpress_post_count WordPress posts count
# TYPE wordpress_post_count gauge
wordpress_post_count{instance="http://wordpress:80"} 1
# HELP wordpress_tag_count WordPress tags count
# TYPE wordpress_tag_count gauge
wordpress_tag_count{instance="http://wordpress:80"} 0
# HELP wordpress_taxonomy_count WordPress taxonomy count
# TYPE wordpress_taxonomy_count gauge
wordpress_taxonomy_count{instance="http://wordpress:80"} -1
# HELP wordpress_taxonomer_count WordPress theme count
# TYPE wordpress_taxonomer_count gauge
wordpress_taxonomer_count{instance="http://wordpress:80"} -1
# HELP wordpress_theme_count WordPress theme count
# TYPE wordpress_theme_count gauge
wordpress_theme_count{instance="http://wordpress:80"} -1
```

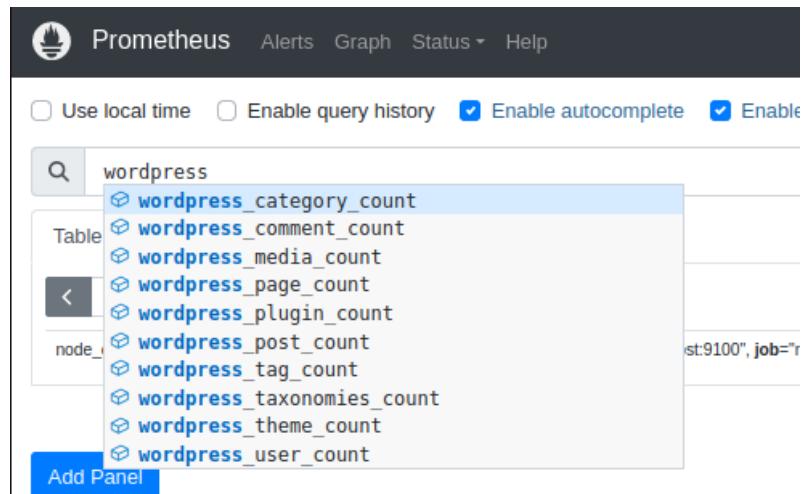
Si sale un fallo al abrir esta página es porque justo estaba scrapeando y no había información disponible en ese momento. Toda esta información proviene de los exportadores y será utilizada para crear gráficas en Grafana.

Entramos en las del node_exporter:



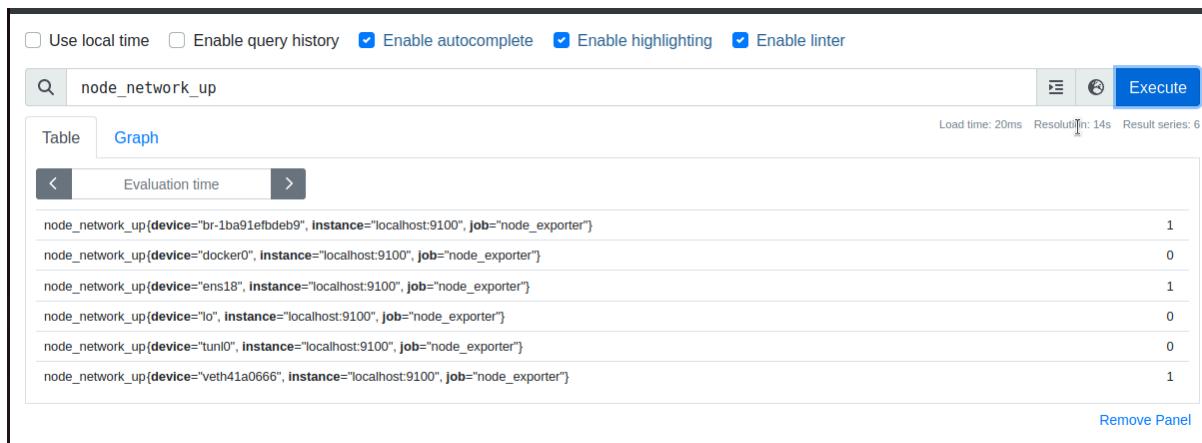
```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 4.7936e-05
go_gc_duration_seconds{quantile="0.25"} 6.1109e-05
go_gc_duration_seconds{quantile="0.5"} 8.1032e-05
go_gc_duration_seconds{quantile="0.75"} 0.000103119
go_gc_duration_seconds{quantile="1"} 0.000140429
go_gc_duration_seconds_sum 0.002304797
go_gc_duration_seconds_count 27
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 8
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.19.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.384304e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 4.936688e+07
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.461843e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 675101
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 9.434432e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 2.384304e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 4.063232e+06
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 3.964928e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 15740
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge
```

También podemos ver qué métricas se están recogiendo de cada parte:



Podemos ver que se registra el número de comentarios, de páginas, de etiquetas, de temas, etc...

En el node_exporter, por ejemplo, filtramos por la métrica que deseemos:



En este caso, filtramos por node_network_up y nos muestra las distintas interfaces de red y su estado (0-inactivo/1-activo).

4.5. Integración de Grafana:

La instalación de Grafana la llevamos a cabo con un script de Ansible llamado instalargrafana.yaml:

```
1  ---
2  - name: Instalacion de grafana y configuracion de exporter
3    hosts: localhost
4    become: yes
5    become_method: sudo
6
7  tasks:
8    - name: Instalar paquetes necesarios para Grafana
9    apt:
10      name:
11        - apt-transport-https
12        - software-properties-common
13        - wget
14      state: present
15      become: yes
16
17    - name: Importar clave GPG de Grafana
18    shell: |
19      mkdir -p /etc/apt/keyrings/
20      wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor > /etc/apt/keyrings/grafana.gpg
21      become: yes
22
23    - name: Agregar repositorio de Grafana
24    apt_repository:
25      repo: deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main
26      state: present
27      become: yes
28
29    - name: Actualizar cache de apt
30    apt:
31      update_cache: yes
32      become: yes
33
34    - name: Instalar Grafana
35    apt:
36      name: grafana
37      state: present
38      become: yes
39
40    - name: Habilitar y arrancar Grafana
41    service:
42      name: grafana-server
43      state: started
44      enabled: yes
45      become: yes
46
```

Este playbook irá con privilegios de sudo. En primer lugar descargamos los paquetes necesarios para Grafana. Luego, crea la carpeta /etc/apt/keyrings/ si no existe y descarga la clave pública de Grafana y la convierte al formato adecuado.

Posteriormente, añade el repositorio de Grafana e indica la firma grafana.gpg. Se actualiza la caché y se instala el paquete. Finalmente habilita un servicio del sistema con el nombre grafana-server, que se iniciará al arrancar el sistema.

Lo ejecutamos con ansible-playbook instalargrafana.yaml:

```
usuario@usuario:~/proyecto$ ansible-playbook instalagrafana.yaml --ask-become-pass
BECOME password:
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [Instalación de grafana y configuracion de exporter] ****
TASK [Gathering Facts] ****
ok: [localhost]

TASK [Instalar paquetes necesarios para Grafana] ****
changed: [localhost]

TASK [Importar clave GPG de Grafana] ****
[WARNING]: Consider using the file module with state=directory rather than running 'mkdir'. If you need to use command because
file is insufficient you can add 'warn: false' to this command task or set 'command_warnings=False' in ansible.cfg to get rid of
this message.
changed: [localhost]

TASK [Agregar repositorio de Grafana] ****
changed: [localhost]

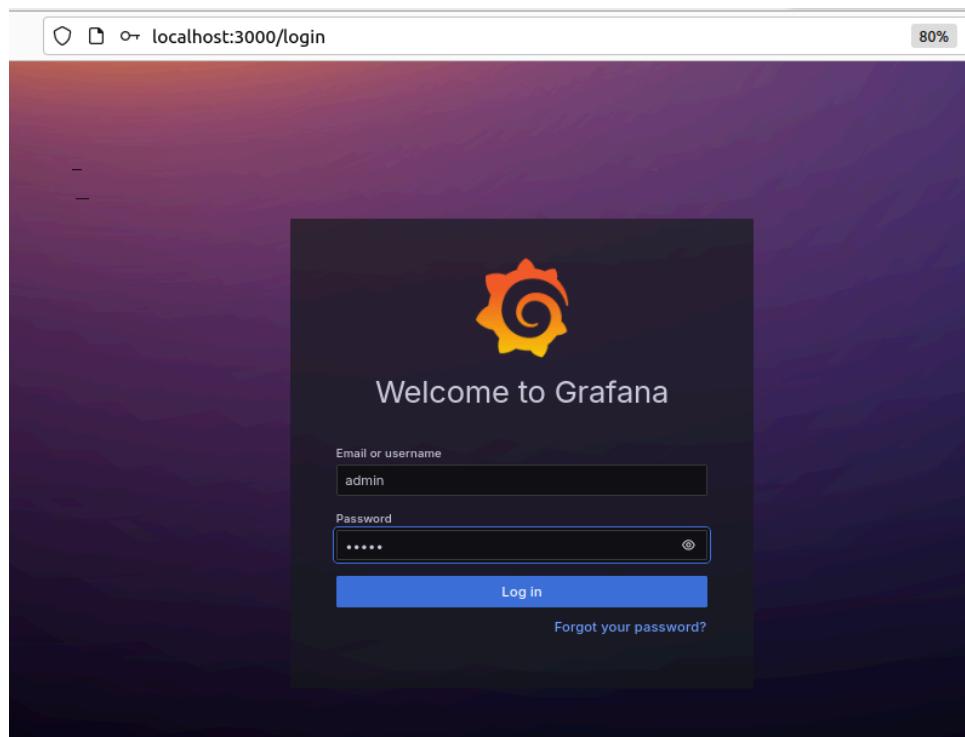
TASK [Actualizar cache de apt] ****
changed: [localhost]

TASK [Instalar Grafana] ****
changed: [localhost]

TASK [Habilitar y arrancar Grafana] ****
changed: [localhost]

PLAY RECAP ****
localhost          : ok=7    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

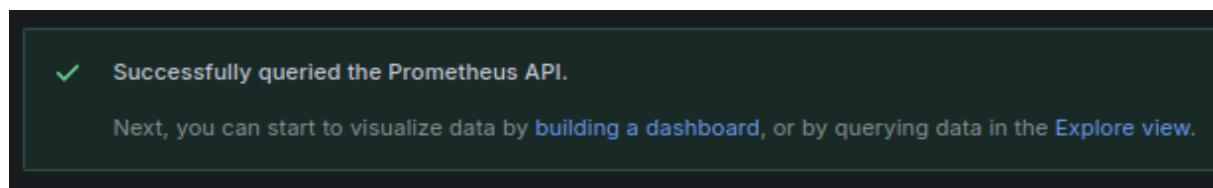
Procedemos a entrar a Grafana desde <http://localhost:3000> :



Vamos a conectar Prometheus con Grafana. Desde connection → Data source, le damos a Add data source:

The screenshot shows the Grafana interface with the left sidebar expanded. The 'Data sources' section is highlighted with a red box. Under 'Connections', there is a single entry labeled 'prometheus'. The main panel displays the 'prometheus' data source configuration. It shows the type as 'Prometheus' and the status as 'Supported'. The 'Settings' tab is selected. A note at the top says: 'Configure your Prometheus data source below' or 'Or skip the effort and get Prometheus (and Loki) as fully-managed, scalable, and hosted data sources from Grafana Labs with the free-forever Grafana Cloud plan.' Below this, the 'Name' field is set to 'prometheus' with a 'Default' toggle switch turned on. A note below the name field states: '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'. The 'Connection' section shows the 'Prometheus server URL' field containing 'http://localhost:9090', which is also highlighted with a red box.

Nos tiene que aparecer este mensaje:



En el apartado Dashboards podremos crear nuestras propias gráficas o bien importar dashboards ya creados (se puede importar desde un fichero json, incrustando el código o con un id de Grafana).

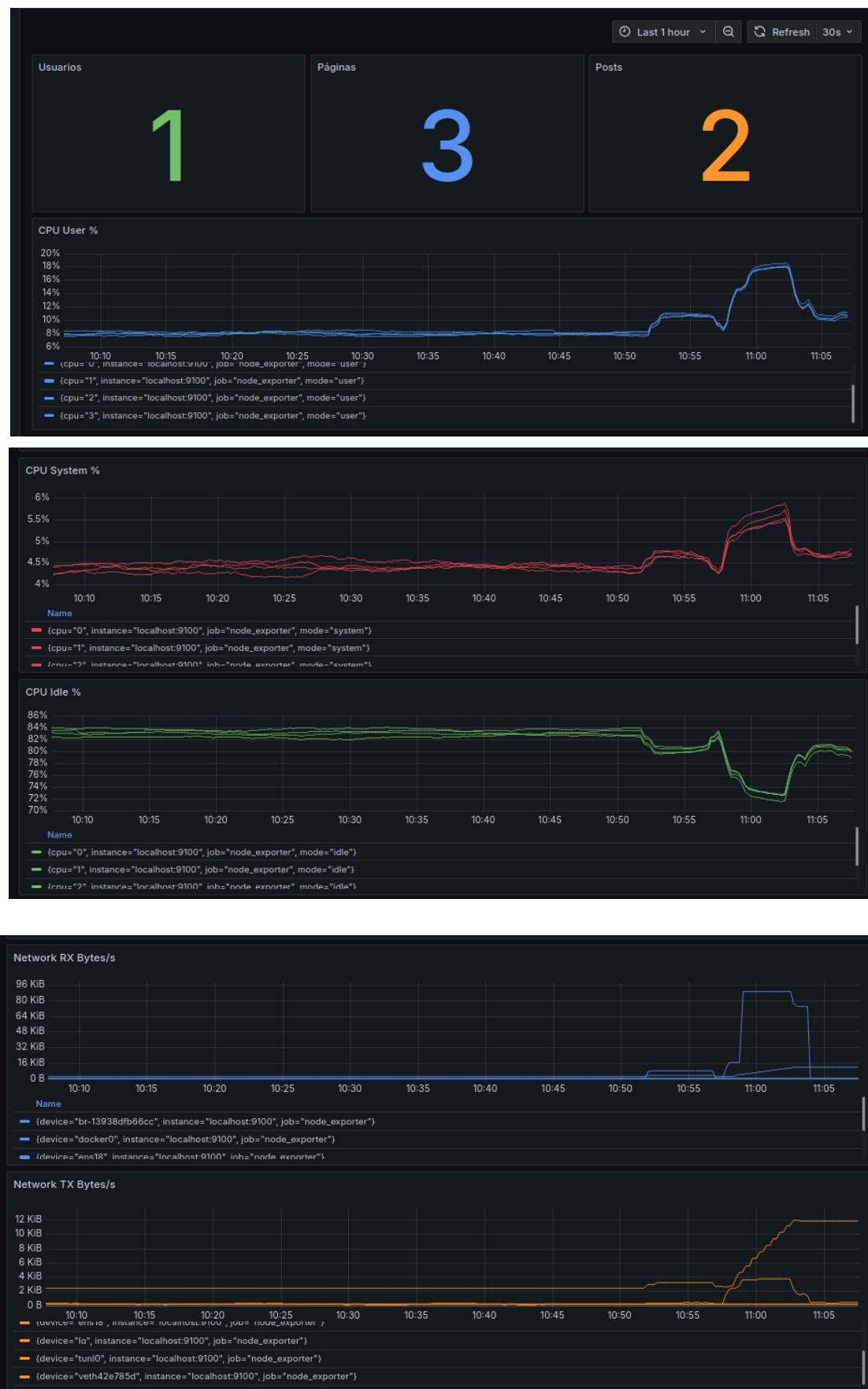
En este caso vamos a importar un dashboard a partir de un fichero json que combina tanto gráficas para los datos del node-exporter como los datos provenientes del wordpress-exporter. Dicho fichero se ha generado mediante IA Generativa ChatGPT partiendo de dos ficheros base de Grafana ya creados para Wordpress y para el Node-exporter pero adaptado a las métricas que generamos nosotros.

Dicho fichero contiene la siguiente información, pero no es el objetivo de este estudio:

```
C:\> Users > Erika > Downloads > proyecto > (1) WordPress_NodeExporter_Dashboard.json > ...  
1  {  
2      "title": "TFG - WordPress + Node Exporter",  
3      "uid": "tfg-dashboard-clean",  
4      "schemaVersion": 38,  
5      "style": "dark",  
6      "tags": [  
7          "TFG",  
8          "WordPress",  
9          "Node Exporter"  
10     ],  
11     "time": {  
12         "from": "now-1h",  
13         "to": "now"  
14     },  
15     "refresh": "30s",  
16     "timezone": "browser",  
17     "__inputs": [  
18         {  
19             "name": "DS_PROMETHEUS",  
20             "label": "Prometheus",  
21             "description": "",  
22             "type": "datasource",  
23             "pluginId": "prometheus",  
24             "pluginName": "Prometheus"  
25         }  
26     ],  
27     "panels": [  
28         {  
29             "type": "stat",  
30             "title": "Usuarios",  
31             "id": 1,  
32             "gridPos": {  
33                 "x": 0,  
34                 "y": 0,  
35                 "w": 8,  
36                 "h": 6  
37             },  
38             "datasource": {  
39                 "type": "prometheus",  
40                 "uid": "${DS_PROMETHEUS}"  
41             },  
42             "targets": [  
43                 {  
44                     "expr": "wordpress_user_count",  
45                     "refId": "A"  
46                 }  
47             ],  
48             "..."  
49         }  
50     ]  
51 }
```

Se puede ver que el origen de los datos es Prometheus y que diseña los paneles en función de la forma de exponer el dato, el título, la posición del panel, los colores...

Una vez importamos dicho json, podremos ver las gráficas de Grafana ya funcionando:



Como podemos observar, nos presenta la información expuesta en Prometheus, a su vez recogida por los exporters.

Aparece información sobre los usuarios, páginas y posts por parte de Wordpress. Por la parte del Node-Exporter, podemos ver información de la máquina host, como es el uso de la CPU tanto por el usuario como por la máquina, el % de inactividad de la CPU, la velocidad en bytes/segundo de la recepción y la transmisión desde la red...

Toda esta información nos será de gran utilidad a la hora de gestionar nuestro entorno, así como para prever futuros problemas de rendimiento mediante la monitorización.

5. Conclusiones

5.1. Resultados obtenidos

A lo largo del proyecto se ha conseguido el despliegue de una aplicación Wordpress funcional dentro de un entorno de K8s local (Minikube), utilizando la automatización como clave mediante scripts de Ansible. Se ha gestionado el ciclo de vida de los pods, servicios y volúmenes. Mediante exportadores de métricas, se ha podido integrar Prometheus y para la visualización de las mismas, Grafana. Se puede decir que se han cumplido los objetivos iniciales del proyecto.

5.2. Dificultades encontradas

Uno de los mayores retos fue entender la lógica del despliegue, incluyendo los servicios y volúmenes. También supuso un reto la configuración de los exporters, verificando que Prometheus pudiera acceder correctamente a las métricas. Lo que más atención ha requerido por mi parte, ha sido la gestión de los puertos expuestos e internos para que toda la comunicación fluyera de manera correcta.

5.3. Mejoras futuras

Como futura mejora, se podría añadir la configuración de alarmas en Grafana para que avisen en situaciones críticas, como una caída del servicio o el uso elevado de CPU. También sería buena idea integrar nuevos exporters para poder monitorizar con más detalle el despliegue por completo, como un exportador de métricas para apache o para las consultas a la base de datos. Cuanta más monitorización, mejor será la prevención ante posibles fallos.

6. Anexos

6.1 Enlace al repositorio GitHub

<https://github.com/erikavalentina21/TFG>

6.2 Webgrafía

- <https://kubernetes.io/docs/concepts/configuration/secret/>
- <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>
- https://github.com/prometheus/mysql_exporter
- <https://prometheus.io/docs/introduction/overview/>
- <https://manpages.debian.org/unstable/prometheus-mysqld-exporter/prometheus-mysqld-exporter.1.en.html>
- <https://elpuig.xeill.net/Members/vcarceler/articulos/recolectores-de-prometheus-node-exporter-en-ubuntu-20-04-systemd>
- <https://janakiev.com/blog/prometheus-setup-systemd/>
- <https://grafana.com/grafana/dashboards/1860>
- <https://docs.ansible.com/>