



Grupo
TEIDE-HEASE

En colaboración con



Formación Profesional
Oficial a Distancia

Proyecto Final

**C.F.G.S. Administración de Sistemas
Informáticos en red**

Susana Vigil Muñoz de Morales



Despliegue de Infraestructuras con Kubernetes:

Desde la gestión propia con K3S hasta el despliegue

Cloud en Azure, con Helm y Terraform.

Ciclo: Administración de Sistemas Informáticos en Red

Alumna: Susana Vigil Muñoz de Morales

Tutor: José Méndez Chaves

Curso: 2023-2024



Índice

1.	ACLARACIONES SOBRE LOS TÉRMINOS USADOS.....	5
2.	INTRODUCCIÓN AL TEMA.....	6
3.	OBJETIVOS	6
4.	DESARROLLO DE CONTENIDOS	7
4.1.	CREACION Y CONFIGURACION DEL CLÚSTER DE KUBERNETES	7
4.1.1.	Requisitos previos	7
4.1.2.	Diagrama de la infraestructura	7
4.1.3.	Creación y configuración de las máquinas	8
4.1.3.1.	Configuración de cada máquina en VirtualBox	8
4.1.3.2.	Configuración de red	9
4.1.4.	Instalación de K3S – Lightweight Kubernetes	9
4.1.4.1.	Configuración de roles master y worker de cada nodo	10
4.1.4.2.	Prueba de funcionamiento y ejecución con un contenedor de Ubuntu....	13
4.1.5.	Instalación y configuración de Helm	14
4.1.5.1.	¿Qué es Helm?	14
4.1.5.2.	Instalación de Helm.....	15
4.1.5.3.	Instalación de aplicaciones con Helm	17
4.2.	CREACIÓN Y CONFIGURACIÓN DE CLÚSTER DE KUBERNETES EN AZURE CON AKS, HELM Y TERRAFORM.....	28
4.2.1.	¿Qué es Azure y AKS?.....	28
4.2.2.	Requisitos previos	28
4.2.2.1.	Instalación en local de CLI de Azure en Windows	28
4.2.2.2.	Instalación de Kubectl en Windows	29
4.2.2.3.	Instalación de Chocolatey	30
4.2.3.	Creación de clúster Kubernetes con AKS	30



4.2.3.1.	Obtención de credenciales para acceder desde el exterior sin usar Cloud Shell de Azure.....	32
4.2.3.2.	Verificación de acceso desde nuestro equipo local	32
4.2.4.	Instalación de Helm en Windows.....	33
4.2.4.1.	Instalación de Prometheus-Grafana con Helm	34
4.3. EJEMPLOS DE USO DE TERRAFORM		36
4.3.1.	¿Qué es Terraform?	36
4.3.2.	Instalación de Terraform en Windows.....	36
4.3.3.	Uso de Terraform con un clúster local virtualizado (K3s)	37
4.3.3.1.	Creación de directorio y ficheros de Terraform	37
4.3.3.2.	Edición de los ficheros.....	38
4.3.3.3.	Prueba de funcionamiento I.....	39
4.3.3.4.	Añadir nuevos recursos al despliegue de Terraform	42
4.3.3.5.	Prueba de funcionamiento II.....	43
4.3.4.	Uso de Terraform con AKS Azure	44
4.3.4.1.	Creación de directorio y ficheros de Terraform	44
4.3.4.2.	Prueba de funcionamiento.....	45
5.	CONCLUSIONES.....	46
6.	REFERENCIAS BIBLIOGRÁFICAS	47
7.	AGRADECIMIENTOS	49
8.	ANEXOS	49
8.1.	Anexo – I: Enlace a repositorio en GitHub	49
8.2.	Anexo – II: Diagrama de la infraestructura	50



1. ACLARACIONES SOBRE LOS TÉRMINOS USADOS

- **Automatización de una infraestructura TI:** Con la automatización de una estructura hacemos uso de tecnologías capaces de realizar tareas con la menor intervención manual. Permite agilizar los procesos, ampliar los entornos y crear flujos de trabajo de integración, distribución e implementación continuas (CI/CD). Hay muchos tipos de automatización, como la automatización de la TI, la automatización empresarial, la automatización robótica de los procesos, la automatización industrial, la inteligencia artificial, el aprendizaje automático y el aprendizaje profundo.
- **Kubernetes:** Es una plataforma de código abierto diseñada para automatizar la implementación, escalado y gestión de aplicaciones en contenedores.
- **Contenedor:** Un contenedor es una unidad de software que incluye una o varias aplicaciones y todas sus dependencias, las encapsulan de manera que se puedan ejecutar de manera consistente en cualquier entorno ya sea físico o virtualizado, aísla los recursos y garantiza su portabilidad, escalabilidad y eficiencia.
- **Helm:** Es un gestor de paquetes para Kubernetes que facilita el despliegue y la gestión de aplicaciones en clústeres de Kubernetes mediante la definición de "charts".
- **AKS:** Servicio de Microsoft Azure que simplifica la implementación y administración de clústeres de Kubernetes en la nube.
- **Terraform:** Es una herramienta de infraestructura como código (IaC) que permite definir y configurar la infraestructura de manera declarativa y automatizada.



2. INTRODUCCIÓN AL TEMA

En la actualidad, el uso de contenedores, especialmente a través de plataformas como Kubernetes, ha transformado radicalmente la forma en que las aplicaciones se desarrollan, implementan y escalan. Los contenedores ofrecen una solución flexible y eficiente para empaquetar, distribuir y ejecutar aplicaciones de manera consistente en diversos entornos, desde entornos locales hasta la nube.

Kubernetes, un proyecto de código abierto originalmente desarrollado por Google, ha emergido como el orquestador de contenedores líder en la industria. Proporciona una plataforma robusta para la gestión automatizada de contenedores, permitiendo a los equipos de desarrollo y operaciones desplegar y gestionar aplicaciones de manera escalable y autónoma.

Esta combinación de contenedores y Kubernetes ofrece una serie de beneficios significativos, incluida una mayor portabilidad de aplicaciones, un desarrollo más rápido, una mayor eficiencia en el uso de recursos y una capacidad de escala dinámica. Además, Kubernetes facilita la implementación de prácticas modernas de desarrollo de software CI/CD y la infraestructura como código (IaC).

3. OBJETIVOS

El propósito de este proyecto es ampliar los conocimientos adquiridos durante la realización de este Ciclo Formativo, analizando las últimas tecnologías de vanguardia utilizadas en el campo de las infraestructuras y sistemas TI y aplicándolas en un caso de uso real.

Se realizará una primera aproximación al uso de contenedores y Kubernetes con el objetivo de comprender sus conceptos y funcionamiento, tecnologías ampliamente utilizadas en la industria, tanto en entornos en Nube como en Bare-metal.

En primer lugar, se instalará y configurará un clúster de Kubernetes en un entorno basado en Ubuntu y se desplegarán diversas aplicaciones basadas en contenedores en dicho clúster. Tras esto, se configurará un clúster de las mismas características en un entorno cloud y se



desplegarán nuevamente las mismas aplicaciones, demostrándose la versatilidad e idempotencia de la tecnología de contenedores.

Finalmente, se realizará una primera aproximación a Terraform, una de las aplicaciones más extendidas para la gestión de infraestructuras como código (IaC), realizándose el despliegue de dichos contenedores de forma transparente en ambos clústeres de Kubernetes anteriormente configurados.

4. DESARROLLO DE CONTENIDOS

4.1. CREACION Y CONFIGURACION DEL CLÚSTER DE KUBERNETES

4.1.1. Requisitos previos

Para la realización de este proyecto debemos contar con lo siguiente:

RAM mínima disponible de 16Gb.	VirtualBox instalado.
Procesador con VT-x Support.	ISO Ubuntu 20.04 Server Edition.
Conexión a internet.	Cliente SSH MobaXtrem instalado.

4.1.2. Diagrama de la infraestructura

La infraestructura inicial será la siguiente:

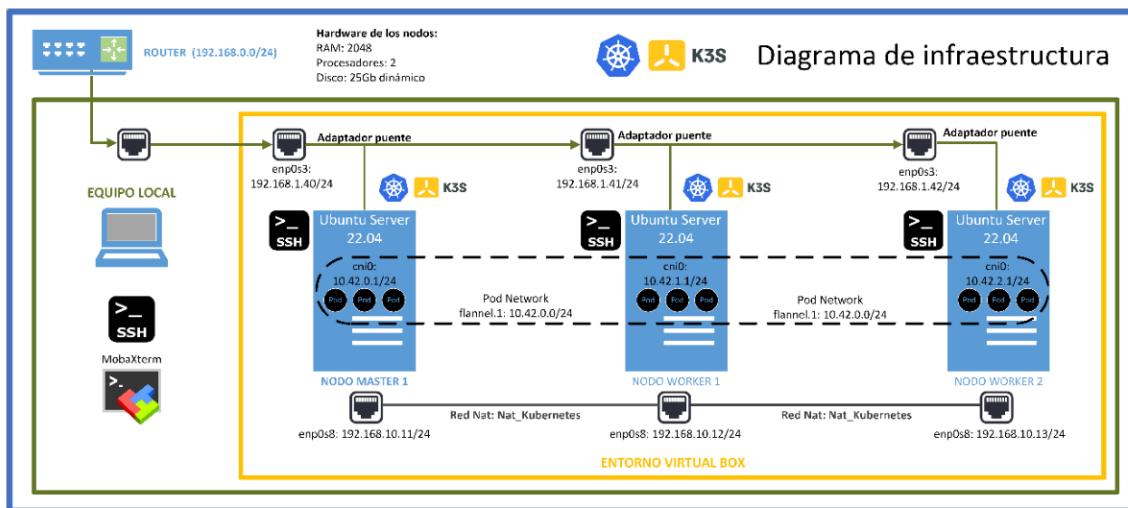


Diagrama de la infraestructura: Ampliación en Anexo II



4.1.3. Creación y configuración de las máquinas

4.1.3.1. *Configuración de cada máquina en VirtualBox*

Comenzamos con la creación de un Master y dos Workers. La configuración de las máquinas será la siguiente:



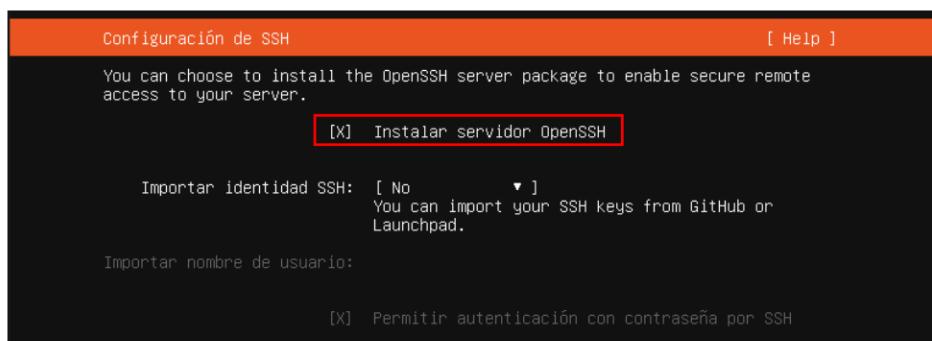
RAM: 4096
Procesadores: 2
Disco: 25Gb dinámico
S.O: Ubuntu Server 22.04
Adaptadores de red:
- Adaptador 1: Adaptador puente
- Adaptador 2: RedNAT Nat_KUBERNETES [192.168.10.0/24]



El S.O base de nuestras máquinas será **Ubuntu Server 22.04**. Ubuntu Server es una variante del sistema operativo Ubuntu diseñada específicamente para su uso en servidores. Está basado en el núcleo Linux y es desarrollado y mantenido por Canonical Ltd. Ubuntu Server ofrece una plataforma estable y segura para ejecutar servicios y aplicaciones en entornos de servidor.

El motivo de los dos adaptadores de red es para tener por un lado comunicación externa y por otro lado una red privada propia para el clúster y sus nodos, ya que no tengo la posibilidad de hacer la infraestructura real con elementos físicos como switches o firewalls. La elección del 192.168.1.0/24 para la RedNAT es para evitar posibles conflictos con el router y los firewalls de Kubernetes en la virtualización.

Una vez creada la configuración de cada máquina en Virtual Box, se procederá a la instalación del sistema base de las mismas. Durante el proceso de instalación debemos confirmar que se detectan ambos adaptadores de red y seleccionamos la instalación de OpenSSH para una posterior conexión con el cliente MobaXtrem y una interacción más cómoda con las máquinas a lo largo del proyecto.





Finalizada la instalación de todas las máquinas se actualizan todos los paquetes.

```
apt update  
apt upgrade
```

4.1.3.2. Configuración de red

Verificamos las IP de las máquinas, instalamos *net-tools*, por seguridad, realizamos una copia del archivo de configuración de red y editamos el *.yaml*, aplicando IP estáticas para los adaptadores de red. Finalmente aplicamos cambios.

```
ip a  
apt install net-tools  
cp /etc/netplan/00-installer-config.yaml /etc/netplan/00-installer-config.yaml.OLD  
nano /etc/netplan/00-installer-config.yaml  
netplan apply
```

4.1.4. Instalación de K3S – Lightweight Kubernetes



K3s es una distribución ligera de Kubernetes diseñada para entornos de desarrollo, pruebas y producción con recursos limitados. Fue desarrollada por Rancher Labs y está optimizada para su uso en sistemas con poca capacidad de memoria y CPU, como dispositivos IoT, máquinas virtuales y entornos de pruebas locales.

K3s proporciona un script de instalación que proporciona una manera sencilla de instalarlo como un servicio en sistemas basados en *systemd* u *openrc*. El script lo tenemos disponible en: <https://get.k3s.io>.

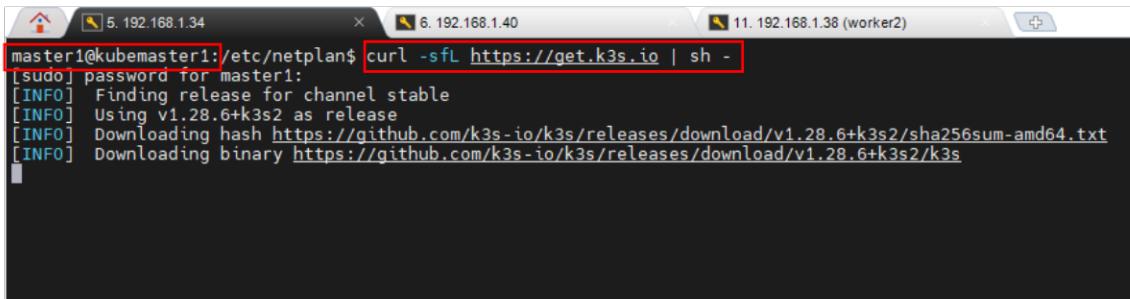
Después de ejecutar la instalación:

- El servicio K3s se configurará para reiniciarse automáticamente después de que se reinicie el nodo o si el proceso falla o finaliza.
- Se instalarán utilidades adicionales, incluidas *kubectl*, *crictl*, *ctr*, *k3s-killall.sh* y *k3s-uninstall.sh*
- Se escribirá un archivo *kubeconfig*/*etc/rancher/k3s/k3s.yaml* y el cliente de Kubernetes, *kubectl*, instalado por K3s lo usará automáticamente



Para la instalación se ejecutará el siguiente comando en los 3 nodos:

```
curl -sfL https://get.k3s.io | sh -
```

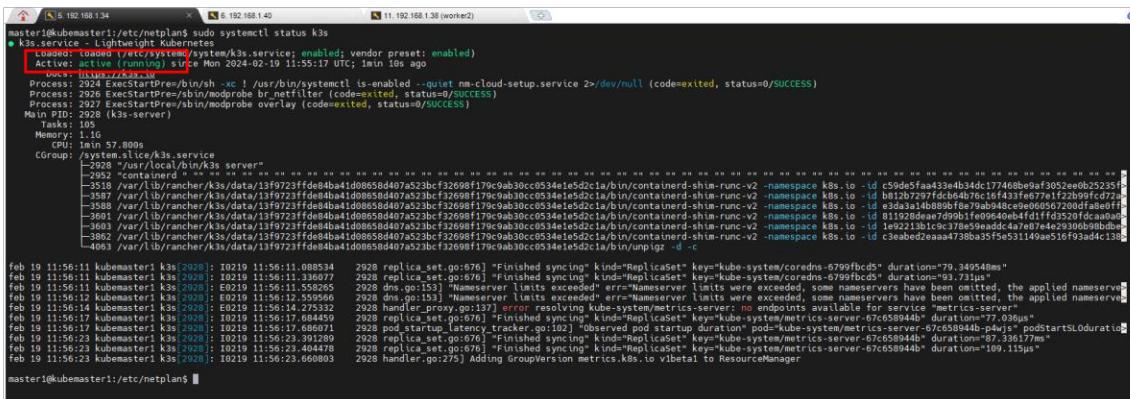


```
master1@kubemaster1:/etc/netplan$ curl -sfL https://get.k3s.io | sh -
[sudo] password for master1:
[INFO] Finding release for channel stable
[INFO] Using v1.28.6+k3s2 as release
[INFO] Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.28.6+k3s2/sha256sum-amd64.txt
[INFO] Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.28.6+k3s2/k3s
```

Finalizada la instalación comprobaremos el estado del servicio en los 3 nodos con el siguiente comando:

```
systemctl status k3s
```

Debemos comprobar que su estado en todos los nodos sea: **active (running)**



```
master1@kubemaster1:/etc/netplan$ systemctl status k3s
● k3s.service - lightweight Kubernetes
   Active: active (running) since Mon 2024-02-19 11:55:17 UTC; 1min 10s ago
     Main PID: 2928 (k3s-server)
      Tasks: 1 (since Mon 2024-02-19 11:55:17 UTC)
     Memory: 1.1G
        CPU: 1min 57.800s
       CGroup: /system.slice/k3s.service
               └─ 2928 "kubelet" "/usr/local/bin/k3s server"
      +-- 2952 "container"
      +-- 3518 "/var/lib/rancher/k3s/data/13f9723ffde4b41d0858d407a523bcf32698f179c9ab30c05341e5d2c1a/b/in/containerd-shim-runc-v2 -namespace k8s.io -id c59d65faa4394b34dc17746beafaf3052ee0b25235"
      +-- 3537 "/var/lib/rancher/k3s/data/13f9723ffde4b41d0858d407a523bcf32698f179c9ab30c05341e5d2c1a/b/in/containerd-shim-runc-v2 -namespace k8s.io -id e3da31a4b889bf6e79ab94ce0eb4e50505dfabedf1"
      +-- 3588 "/var/lib/rancher/k3s/data/13f9723ffde4b41d0858d407a523bcf32698f179c9ab30c05341e5d2c1a/b/in/containerd-shim-runc-v2 -namespace k8s.io -id e3da31a4b889bf6e79ab94ce0eb4e50505dfabedf1"
      +-- 3601 "/var/lib/rancher/k3s/data/13f9723ffde4b41d0858d407a523bcf32698f179c9ab30c05341e5d2c1a/b/in/containerd-shim-runc-v2 -namespace k8s.io -id 811928d8ea7d79b1fe09640e84fd1ffdf520fdca0a0d"
      +-- 3602 "/var/lib/rancher/k3s/data/13f9723ffde4b41d0858d407a523bcf32698f179c9ab30c05341e5d2c1a/b/in/containerd-shim-runc-v2 -namespace k8s.io -id 1e92213b1c9378e59eadd44e7e74e4e93906b98bd0"
      +-- 3603 "/var/lib/rancher/k3s/data/13f9723ffde4b41d0858d407a523bcf32698f179c9ab30c05341e5d2c1a/b/in/containerd-shim-runc-v2 -namespace k8s.io -id c3e0bed2eaaa4738a035f553149ae516f93a4d4c13"
      +-- 4003 "/var/lib/rancher/k3s/data/13f9723ffde4b41d0858d407a523bcf32698f179c9ab30c05341e5d2c1a/b/in/unping -d"

feb 19 11:59:11 kubemaster1 k3s[2928]: [0219 11:59:11.088534 2928] replica_set.go:976] "finished syncing" kind="ReplicaSet" keys="kube-system/crds-6799fbcd" duration="9.249548ms"
feb 19 11:59:11 kubemaster1 k3s[2928]: [0219 11:59:11.336077 2928] replica_set.go:976] "finished syncing" kind="ReplicaSet" keys="kube-system/crds-6799fbcd" duration="91.731us"
feb 19 11:59:11 kubemaster1 k3s[2928]: [0219 11:59:11.558265 2928] dns.go:153] "Nameserver limits exceeded" err:"Nameserver limits were exceeded, some nameservers have been omitted, the applied nameservers may not be fully functional" kind="ReplicaSet" keys="kube-system/crds-6799fbcd"
feb 19 11:59:12 kubemaster1 k3s[2928]: [0219 11:59:12.559566 2928] dns.go:153] "Nameserver limits exceeded" err:"Nameserver limits were exceeded, some nameservers have been omitted, the applied nameservers may not be fully functional" kind="ReplicaSet" keys="kube-system/crds-6799fbcd"
feb 19 11:59:13 kubemaster1 k3s[2928]: [0219 11:59:13.560676 2928] dns.go:153] "Nameserver limits exceeded" err:"Nameserver limits were exceeded, some nameservers have been omitted, the applied nameservers may not be fully functional" kind="ReplicaSet" keys="kube-system/crds-6799fbcd"
feb 19 11:59:17 kubemaster1 k3s[2928]: [0219 11:59:17.684459 2928] replica_set.go:170] "finished syncing" kind="ReplicaSet" keys="kube-system/metrics-server-67c658944b" duration="77.036us"
feb 19 11:59:17 kubemaster1 k3s[2928]: [0219 11:59:17.686671 2928] pod_startup_latency_tracker.go:102] "observed pod startup duration" pod="kube-system/metrics-server-67c658944b-p4w5" podStartsSL0duration=0.000000ms
feb 19 11:59:23 kubemaster1 k3s[2928]: [0219 11:59:23.391289 2928] replica_set.go:976] "finished syncing" kind="ReplicaSet" keys="kube-system/metrics-server-67c658944b" duration="87.336177ms"
feb 19 11:59:23 kubemaster1 k3s[2928]: [0219 11:59:23.608603 2928] replica_set.go:976] "finished syncing" kind="ReplicaSet" keys="kube-system/metrics-server-67c658944b" duration="109.115μs"
feb 19 11:59:23 kubemaster1 k3s[2928]: [0219 11:59:23.608603 2928] handler.go:273] Adding groupVersion metrics.k8s.io to vistat1 to ResourceManager

master1@kubemaster1:/etc/netplan$
```

4.1.4.1. Configuración de roles master y worker de cada nodo

Una vez instalado K3s y verificado que se encuentra activo en todos los nodos, debemos declarar los ROLES master y worker.

El nodo MASTER será el encargado de correr los contenedores exclusivos para la orquestación y gestión del clúster por parte de Kubernetes. Los nodos WORKERS serán los encargados de correr los contenedores de aplicación.



Iremos al nodo MASTER y lanzaremos el siguiente comando para declarar el ROLE de MASTER.

Debemos colocar la **IP de la red privada (RedNat) de nuestro nodo MASTER** por la que conectan los nodos y el puerto por donde se comunicarán :**6443**.

```
./k3s agent --server https://192.168.10.11:6443 &
```

```
Last login: Mon Feb 19 11:06:19 2024 from 192.168.1.36
master1@kubemaster1:~$ sudo ./k3s server --node-ip=192.168.10.11 &
[1] 6852
master1@kubemaster1:~$
```

Con el siguiente comando podemos comprobar los nodos y sus roles:

```
sudo k3s kubectl get nodes
```

Podemos ver que nuestro master figura ya con su role y como control-plane.

NAME	STATUS	ROLES	AGE	VERSION
kubeworker2	Ready	<none>	8m6s	v1.28.6+k3s2
kubemaster1	Ready	control-plane, master	142m	v1.28.6+k3s2
kubeworker1	Ready	<none>	76m	v1.28.6+k3s2

Al realizar la instalación de K3s, se crea un elemento llamado **TOKEN** para cada máquina. En el contexto de Kubernetes, un **TOKEN** es una credencial utilizada para la autenticación de los componentes del clúster y de los usuarios que intentan acceder al clúster. El *token* se utiliza para autenticar la identidad del cliente que realiza la solicitud y para autorizar las acciones que puede realizar en el clúster.

La ruta donde se ubica el fichero que contiene el *token* de cada nodo es la siguiente:

```
/var/lib/rancher/k3s/server/agent-token
```

Podemos obtener el token con el siguiente comando:

```
cat /var/lib/rancher/k3s/server/agent-token
```



```
root@kubemaster1:/var/lib/rancher/k3s/server# ls
agent-token  cred  db  etc  kine.sock  manifests  node-token  static  tls  token
root@kubemaster1:/var/lib/rancher/k3s/server# cat /var/lib/rancher/k3s/server/agent-token
K10ccb870849e95c5ba274a1bb24907771ad29972a11620c44b6e24ed3ff5107cb8::server:6da64c0e84e94aa0294c428d81eb567
root@kubemaster1:/var/lib/rancher/k3s/server#
```

Una vez obtenido el ID del TOKEN MASTER, iremos a nuestros WORKER y ejecutamos el siguiente comando para unirnos al agente master.

```
curl -sfL https://get.k3s.io | K3S_URL=https://myserver:6443 K3S_TOKEN=mynodetoken sh -
```

En “*myserver*” colocaremos la IP privada de nuestro nodo máster y en “*mynodetoken*” el ID obtenido anteriormente.

```
worker2@ubeworker2:~$ sudo curl -sfL https://get.k3s.io | K3S_URL=https://192.168.10.11:6443 K3S_TOKEN=K10ccb870849e95c5ba274a1bb24907771ad29972a11620c44b6e24ed3ff5107cb8::server:6da64c0e84e94aa0294c428d81eb567
[INFO]  Finding release for channel stable
[INFO]  Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.28.6+k3s2/sha256sum-amd64.txt
[INFO]  Skipping binary download, installed k3s matches hash
[INFO]  Skipping installation of SELinux RPM
[INFO]  Skipping SELinux policy link to k3s, already exists
[INFO]  Skipping /usr/local/bin/crictl symbolic link to k3s, already exists
[INFO]  Skipping /usr/local/bin/ctr symbolic link to k3s, already exists
[INFO]  Creating k3sctl script /usr/local/bin/k3sctl-all-in-one
[INFO]  Creating k3sctl script /usr/local/bin/k3sctl-uninstall.sh
[INFO]  env: Creating environment file /etc/systemd/system/k3s-agent.service.env
[INFO]  systemd: Creating service file /etc/systemd/system/k3s-agent.service
[INFO]  systemd: Starting k3s-agent
Job for k3s-agent.service failed because the control process exited with error code.
See "systemctl status k3s-agent.service" and "journalctl -xeu k3s-agent.service" for details.
worker2@ubeworker2:~$
```

Es probable que ejecutando de nuevo el comando `sudo k3s kubectl get nodes` siga sin figurar el role, en ese caso podemos solucionarlo con el siguiente comando para cada nodo worker:

```
sudo kubectl label nodes kubeworker1 node-role.kubernetes.io/worker=worker
sudo kubectl label nodes kubeworker2 node-role.kubernetes.io/worker=worker
```

Ahora comprobamos que nuestros nodos se ven sin problema y cada uno figura con su role correspondiente, por lo que ya tenemos finalizado el clúster.

```
root@kubemaster1:/var/lib/rancher/k3s/server# sudo k3s kubectl get nodes
NAME      STATUS   ROLES      AGE      VERSION
kubemaster1 Ready    control-plane,master   9h      v1.28.6+k3s2
kubeworker2 Ready    worker     7h7m    v1.28.6+k3s2
kubeworker1 Ready    worker     8h      v1.28.6+k3s2
root@kubemaster1:/var/lib/rancher/k3s/server#
```



4.1.4.2. Prueba de funcionamiento y ejecución con un contenedor de Ubuntu.

Existen contenedores predefinidos que están disponibles públicamente en repositorios de contenedores como Docker Hub, Quay.io, y otros. Estos contenedores predefinidos son imágenes que contienen sistemas operativos, aplicaciones y servicios comunes listos para ser ejecutados en contenedores.

Como primera prueba, se creará un **POD** basado en la imagen de Ubuntu. Para ello, se lanza el siguiente comando en el nodo máster:

```
kubectl run ubuntu --image=ubuntu --restart=Never --command sleep infinity
```

A terminal window titled '2. Master1' shows the command being run. The output shows 'pod/ubuntu created'. The entire command and its output are highlighted with a red box.

Para comprobar los pods creados y su estado en el clúster, se lanza el siguiente comando:

```
kubectl get pods
```

A terminal window titled '2. Master1' shows the command being run. The output shows 'pod/ubuntu created' and then lists the pod 'ubuntu' with status 'Running'. The entire command and its output are highlighted with a red box.

Pero obtener más detalle, como el nodo donde está, la IP del Network Pod, reinicios, etc., se lanzará con el argumento **-o wide**:

```
kubectl get pods -o wide
```

A terminal window titled '2. Master1' shows the command being run. The output shows 'pod/ubuntu created' and then lists the pod 'ubuntu' with detailed information including 'NODE' (kubeworker2), 'IP' (10.42.2.4), and 'READINESS GATES' (<none>). The entire command and its output are highlighted with a red box.

Ahora tenemos un pod de Ubuntu que corre en el nodo worker2 y podemos acceder a su terminal desde cualquiera de los nodos lanzando el siguiente comando:



```
kubectl exec -it ubuntu -- /bin/bash
```

Este comando ejecuta un terminal interactivo (-it) dentro del contenedor del pod llamado "ubuntu". `/bin/bash` y especifica que queremos usar el **shell Bash** dentro del contenedor. En la captura podemos comprobar que estaríamos dentro y que es exactamente igual que manejar un SO Ubuntu.

```
root@kubemaster1:/home/master1# kubectl get pods -o wide
NAME      READY   STATUS    RESTARTS   AGE     IP          NODE      NOMINATED NODE   READINESS GATES
ubuntu   1/1     Running   0          16m    10.42.2.4   kubeworker2   <none>        <none>
root@kubemaster1:/home/master1# kubectl exec -it ubuntu -- /bin/bash
root@ubuntu:# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@ubuntu:#
```

```
root@ubuntu:# ls
etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@ubuntu:# cd /etc/
root@ubuntu:# ls
adduser.conf      cloud.conf      default.conf  fstab  host.conf  issue.net  legal.conf  machine-id  opt  profile.d  rc3.d  resolv.conf  shells  sysctl.d
alternatives.conf cron.d        default       gai.conf  hostname  kernel    libaudit.conf  mke2fs.conf  os-release  profile.d  rc4.d  rmt  skel
bash.bashrc       cron.daily    default       gpk.conf  group    hosts     ld.so.cache  login.defs  netconsole  pam.conf   rc0.d  rc5.d  security  subuid  systemd
bashrc            dhclient.conf  default       gshadow  init.d    issue    ld.so.conf    logrotate.d  network   pam.conf   rc1.d  rc6.d  selinux  subuid  terminal
bindresport.blacklist debconf.conf  default       gss.conf  libaudit  issue    ld.so.conf.d  logrotate.d  nsswitch.conf  passwd  rc2.d  rc5.d  shadow  subuid  update-motd
root@ubuntu:/etc#
```

```
root@ubuntu:/etc# cd ..
root@ubuntu:# apt update
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:3 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1463 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/main amd64 Packages [1792 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [44.6 kB]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [1796 kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [17.5 MB]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [1070 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [266 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy/restricted amd64 Packages [164 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1343 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [50.4 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [1834 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1742 kB]
Get:17 http://archive.ubuntu.com/ubuntu jammy-backports/main amd64 Packages [50.4 kB]
Get:18 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [28.1 kB]
Fetched 29.7 MB in 10s (2966 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
3 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@ubuntu:#
```

4.1.5. Instalación y configuración de Helm

4.1.5.1. ¿Qué es Helm?



HELM es una herramienta que simplifica la gestión de aplicaciones en Kubernetes, permitiendo empaquetar, distribuir e instalar aplicaciones de manera estandarizada usando "**charts**", que son paquetes preconfigurados de recursos de Kubernetes. Facilita la automatización y el mantenimiento de aplicaciones en entornos de contenedores.



4.1.5.2. Instalación de Helm

Las formas de instalar HELM son las siguientes:

- **Instalación desde paquetes precompilados:** la página oficial de HELM ofrece una serie de paquetes listos para ejecutar.
- **Instalación con gestores de paquetes:** Algunos gestores de paquetes son Homebrew para macOS o Chocolatey para Windows para instalar HELM.
- **Instalación mediante script:** HELM proporciona scripts de instalación que se pueden ejecutar en la terminal para instalar la herramienta y sus dependencias de forma automática.
- **Usar un clúster de Kubernetes gestionado:** Servicios en la nube como Google Kubernetes Engine (GKE) o Amazon Elastic Kubernetes Service (EKS) ofrecen integración con HELM y permiten instalarlo fácilmente en un clúster gestionado.
- **Instalación en entornos de desarrollo local:** Si trabajamos en un entorno de desarrollo local, podemos utilizar herramientas como Minikube o Kind (Kubernetes in Docker) para crear un clúster de Kubernetes local y luego instalar HELM en ese clúster.

4.1.5.2.1. Instalación mediante script

La instalación puede realizarse desde cualquier máquina que tenga configurado el archivo **KUBECONFIG** del clúster de Kubernetes.

```
curl -fsSL -o get_helm.sh  
https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3  
chmod 700 get_helm.sh  
./get_helm.sh
```

A screenshot of a Linux terminal window titled "Master1". The terminal shows the following command being run:

```
root@kubemaster1:/home/master1# curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3  
root@kubemaster1:/home/master1# chmod 700 get_helm.sh  
root@kubemaster1:/home/master1# ./get_helm.sh  
Downloading https://get.helm.sh/helm-v3.14.2-linux-amd64.tar.gz  
Verifying checksum... Done.  
Preparing to install helm into /usr/local/bin  
helm installed into /usr/local/bin/helm  
root@kubemaster1:/home/master1#
```

Con estos pasos ya tendríamos instalado Helm. Para ver la versión instalada ejecutamos el siguiente código:

```
helm version
```



```
root@kubemaster1:/home/master1# helm version
version.BuildInfo{Version:"v3.14.2", GitCommit:"c309b6f0ff63856811846ce18f3bdc93d2b4d54b", GitTreeState:"clean", GoVersion:"go1.21.7"}
root@kubemaster1:/home/master1#
```

4.1.5.2.2. Helm Stable Charts

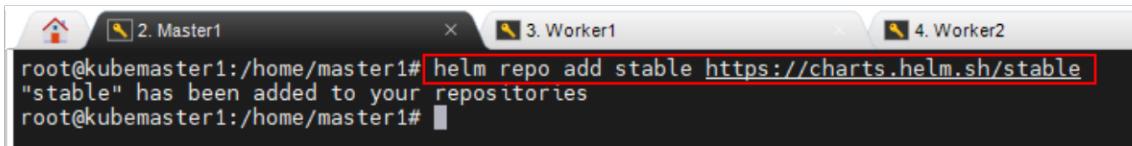
Helm Stable Charts es un repositorio oficial de charts (gráficos) mantenidos por la comunidad de Helm y contienen una amplia variedad de charts listos para usar, que abarcaban desde bases de datos y servidores web hasta herramientas de monitoreo y logging. Algunos ejemplos son MySQL, WordPress, Prometheus y Grafana, entre otros.

Estos charts proporcionan una forma rápida y sencilla de desplegar aplicaciones y servicios comunes en los clústeres sin necesidad de configurar manualmente cada componente.

A partir de Helm 3, la comunidad de Helm ha migrado hacia un enfoque de "repositorios incubator" en lugar de un repositorio estable. Esto significa que los charts de Helm ya no están centralizados en un solo repositorio oficial, sino que se pueden encontrar y utilizar charts de diversos repositorios mantenidos por la comunidad de Helm.

A continuación, vamos a añadir un repositorio de Stable Charts, para lo que utilizamos el siguiente comando:

```
helm repo add stable https://charts.helm.sh/stable
```



```
2. Master1 3. Worker1 4. Worker2
root@kubemaster1:/home/master1# helm repo add stable https://charts.helm.sh/stable
"stable" has been added to your repositories
root@kubemaster1:/home/master1#
```

Añadir otros charts.

A continuación, se dan los pasos para añadir más repositorios en Helm y tener más opciones de búsqueda de paquetes.

- [Artifact-hub](#)

Añadir el repositorio:

```
helm repo add artifact-hub https://charts.helm.sh/stable
```

Actualizar repositorio:

```
helm repo update
```



```
2. Master1 x 3. Worker1 x 4. Worker2 x
root@kubemaster1:/home/master1# helm repo add artifact-hub https://charts.helm.sh/stable
"artifact-hub" has been added to your repositories
root@kubemaster1:/home/master1# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "stable" chart repository
...Successfully got an update from the "artifact-hub" chart repository
Update Complete. *Happy Helm-ing!*
root@kubemaster1:/home/master1#
```

- Bitnami

Añadir el repositorio:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

Actualizar repositorio:

```
helm repo update
```

```
2. Master1 x 3. Worker1 x 4. Worker2 x
root@kubemaster1:/home/master1# helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
root@kubemaster1:/home/master1# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "stable" chart repository
...Successfully got an update from the "artifact-hub" chart repository
...Successfully got an update from the "bitnami" chart repository
Update Complete. *Happy Helm-ing!*
root@kubemaster1:/home/master1#
```

4.1.5.3. Instalación de aplicaciones con Helm

Como anteriormente se comentó, con Helm se puede desplegar una gran variedad de aplicaciones de distinta finalidad. A continuación, se exponen algunos ejemplos.

4.1.5.3.1. Instalación de Prometheus - Grafana



Prometheus

Prometheus es una herramienta de monitoreo de código abierto que recopila datos sobre el rendimiento y el estado de sistemas y servicios en tiempo real. Permite almacenar, consultar y alertar sobre estas métricas, lo que facilita la detección de problemas y la toma de medidas correctivas en entornos de producción.



Cuando Prometheus se implementa en un clúster, como por ejemplo en un entorno de Kubernetes como el que se está realizando, podemos monitorear aspectos relacionados con la infraestructura, los servicios y las aplicaciones que se ejecutan en ese clúster, lo que ayuda a garantizar su salud y rendimiento óptimo. Algunas de los recursos que puede monitorear incluyen:

- **Recursos de hardware:** Como el uso de CPU, memoria y almacenamiento en los nodos del clúster.
- **Salud de los nodos:** Para asegurarse de que los nodos del clúster estén funcionando correctamente y no estén experimentando problemas como fallas o sobrecargas.
- **Recursos de red:** Como el tráfico de red y la latencia entre los nodos del clúster.
- **Aplicaciones y servicios desplegados:** Para monitorear el estado y el rendimiento de las aplicaciones y servicios que se ejecutan dentro del clúster, como aplicaciones web, bases de datos, brokers de mensajería, etc.
- **Cargas de trabajo y distribución de recursos:** Para asegurarse de que las cargas de trabajo se distribuyan de manera equitativa entre los nodos y que los recursos se utilicen de manera eficiente.



Grafana es una plataforma de visualización y análisis de datos de código abierto utilizada principalmente para monitorear y analizar métricas y registros en tiempo real. Algunas características clave incluyen:

- **Paneles Interactivos:** Permite la creación de paneles interactivos y personalizables que muestran datos en tiempo real en forma de gráficos, tablas y otros elementos visuales.
- **Conectores de Datos:** Ofrece una amplia gama de conectores de datos que permiten la integración con numerosas fuentes de datos, incluidas bases de datos de series temporales como Prometheus, bases de datos SQL, sistemas de almacenamiento de registros y más.
- **Alertas:** Permite configurar alertas basadas en umbrales y condiciones predefinidas, lo que facilita la detección de anomalías y la notificación de incidentes críticos a través de



varios canales, como correo electrónico, Slack o integraciones con sistemas de gestión de incidentes.

- **Exploración de Datos:** Facilita la exploración y el análisis de datos históricos mediante herramientas de consulta y filtrado, lo que permite a los usuarios investigar tendencias y patrones en los datos.
- **Escalabilidad y Extensibilidad:** Grafana es altamente escalable y se puede integrar fácilmente con otras herramientas y sistemas a través de su arquitectura modular y su amplia gama de complementos y extensiones.

Primero debemos crear el “*namespace*” donde instalaremos Prometheus:

```
kubectl create namespace prometheus
```

```
root@kubemaster1:/home/master1# kubectl create namespace prometheus
namespace/prometheus created
root@kubemaster1:/home/master1#
```

Buscaremos cuál es la última versión con el siguiente comando:

```
helm search repo prometheus-community/kube-prometheus-stack
```

NAME	CHART VERSION	APP VERSION	DESCRIPTION
prometheus-community/kube-prometheus-stack	57.0.1	v0.72.0	kube-prometheus-stack collects Kubernetes manif...

Una vez hechos los pasos anteriores, se añade el repositorio Helm de Prometheus, indicando la versión más reciente encontrada, con el siguiente comando:

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

```
root@kubemaster1:/home/master1# helm repo add stable https://charts.helm.sh/stable
"stable" has been added to your repositories
root@kubemaster1:/home/master1# helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" has been added to your repositories
root@kubemaster1:/home/master1#
```

Una vez añadido el repositorio, instalamos *kube-stack-prometheus* (Prometheus), que incluye un despliegue incrustado de **Grafana**, **Node Exporter**, **kube-state-metrics**, y **Alertmanager**.



```
helm install stable prometheus-community/kube-prometheus-stack --version 57.0.1 -n prometheus
```

Durante la ejecución de las prácticas, no se configuró correctamente el archivo KUBECONFIG en el sistema, como se puede observar en la siguiente imagen.

```
root@kubemaster1:/# helm install stable prometheus-community/kube-prometheus-stack --version 57.0.1 --kube-apiserver https://127.0.0.1:6443 -n prometheus --insecure-skip-tls-verify
error: INSTALLATION FAILED: Kubernetes cluster unreachable: Get "https://127.0.0.1:6443/version": tls: failed to verify certificate: x509: certificate signed by unknown authority
root@kubemaster1:/#
```

Para solucionarlo es necesario configurar la variable de entorno con este comando o copiar el fichero en la carpeta *HOME/.kube/config*. En esta ocasión se escoge la primera aproximación:

```
export KUBECONFIG=/etc/rancher/k3s/k3s.yaml
```

Una vez hecho esto, podemos comprobar que ya tenemos Prometheus instalado:

```
root@kubemaster1:/# helm install stable prometheus-community/kube-prometheus-stack --version 57.0.1 --kube-apiserver https://127.0.0.1:6443 -n prometheus --insecure-skip-tls-verify
error: INSTALLATION FAILED: Kubernetes cluster unreachable: Get "https://127.0.0.1:6443/version": tls: failed to verify certificate: x509: certificate signed by unknown authority
root@kubemaster1:/# export KUBECONFIG=/etc/rancher/k3s/k3s.yaml
root@kubemaster1:/# helm install stable prometheus-community/kube-prometheus-stack --version 57.0.1 -n prometheus
NAME: prometheus
LAST DEPLOYED: Mon Mar 11 20:03:37 2024
NAMESPACE: prometheus
STATUS: deployed
REVISION: 1
NOTES:
The kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace prometheus get pods -l "release=stable"
Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.
root@kubemaster1:/#
```

Para listar todos los pods en un el namespace *Prometheus* junto con información adicional, como el nombre del nodo en el que se están ejecutando, ejecutamos el siguiente comando:

```
kubectl get pods -n prometheus -o wide
```

El siguiente comando proporcionará información sobre los servicios en el espacio de nombres *Prometheus*, incluyendo detalles como los puertos expuestos y las direcciones IP.

```
kubectl get svc -n prometheus
```

```
root@kubemaster1:/# kubectl get pods -n prometheus -o wide
NAME                               READY   STATUS    RESTARTS   AGE     IP          NODE
stable-prometheus-node-exporter-nginx7   1/1    Running   0          6m25s  192.168.10.13  kubeworker2
stable-prometheus-node-exporter-5n6rm   1/1    Running   0          6m25s  192.168.10.11  kubemaster1
stable-kube-prometheus-sta-operator-697547bc9d-lt66c  1/1    Running   0          6m24s  10.42.1.9   kubeworker1
stable-kube-state-metrics-5988bf67d5-msqgt   1/1    Running   0          6m24s  10.42.1.8   kubeworker1
alertmanager-stable-kube-prometheus-sta-alertmanager-0  2/2    Running   0          5m52s  10.42.1.11  kubeworker1
prometheus-stable-kube-prometheus-sta-prometheus-0   2/2    Running   0          5m49s  10.42.2.9   kubeworker2
stable-grafana-567c9b5995-vtfkb   3/3    Running   0          6m24s  10.42.2.8   kubeworker2
stable-prometheus-node-exporter-spkds  1/1    Running   0          6m25s  192.168.10.12  kubeworker1
root@kubemaster1:/# kubectl get svc -n prometheus
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
stable-grafana ClusterIP  10.43.187.118  <none>        80/TCP          7m24s
stable-kube-state-metrics ClusterIP  10.43.59.107  <none>        8080/TCP       7m24s
stable-prometheus-node-exporter ClusterIP  10.43.24.218  <none>        9100/TCP       7m24s
stable-kube-prometheus-sta-alertmanager ClusterIP  10.43.229.56  <none>        9093/TCP,8080/TCP  7m24s
stable-kube-prometheus-sta-prometheus ClusterIP  10.43.237.135  <none>        9090/TCP,8080/TCP  7m24s
stable-kube-prometheus-sta-operator ClusterIP  10.43.105.48  <none>        443/TCP         7m24s
alertmanager-operated ClusterIP  None           <none>        9093/TCP,9094/TCP,9094/UDP  6m50s
prometheus-operated ClusterIP  None           <none>        9090/TCP       6m48s
```



4.1.5.3.2. Habilitando el acceso externo a nuestra infraestructura.

Cuando se crea un clúster de Kubernetes, generalmente se crea una IP de clúster para cada nodo, como hemos visto en la captura anterior. Sin embargo, para poder habilitar el acceso externo, necesitamos tener un servicio de **LoadBalancer** o **NodePort**, ya que los servicios de tipo **ClusterIP** sólo exponen el contenedor en la red interna de Kubernetes.

Para eso, necesitaremos editar el servicio de Prometheus y Grafana ejecutando el siguiente comando:

```
kubectl edit svc stable-kube-prometheus-sta-prometheus -n prometheus  
kubectl edit svc stable-grafana -n prometheus
```

Al final de la configuración del servicio debemos modificar el siguiente parámetro:

```
selector:  
  app.kubernetes.io/name: prometheus  
  operator.prometheus.io/name: stable-kube-prometheus-sta-prometheus  
  sessionAffinity: None  
  type: LoadBalancer  
status:  
  loadBalancer: {}  
  
root@kubemaster1:/# kubectl edit svc stable-kube-prometheus-sta-prometheus -n prometheus  
service/stable-kube-prometheus-sta-prometheus edited  
root@kubemaster1:/# kubectl edit svc stable-kube-prometheus-sta-prometheus -n prometheus  
service/stable-grafana edited  
root@kubemaster1:/# █
```

Para comprobar si se han ejecutado los cambios en los servicios correctamente, lanzaremos el siguiente comando:

```
kubectl get svc -n prometheus
```

Durante la ejecución de las prácticas, se identifica que el servicio de Grafana no consigue adquirir una IP externa:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
stable-kube-state-metrics	ClusterIP	10.43.59.107	<none>	8080/TCP	22h
stable-prometheus-node-exporter	ClusterIP	10.43.24.218	<none>	9100/TCP	22h
stable-kube-prometheus-sta-alertmanager	ClusterIP	10.43.229.56	<none>	9093/TCP,8080/TCP	22h
stable-kube-prometheus-sta-operator	ClusterIP	10.43.105.48	<none>	443/TCP	22h
alertmanager-operated	ClusterIP	None	<none>	9093/TCP,9094/TCP,9094/UDP	22h
prometheus-operated	ClusterIP	None	<none>	9090/TCP	22h
stable-kube-prometheus-sta-prometheus	LoadBalancer	10.43.237.135	192.168.10.11,192.168.10.12,192.168.10.13	9090:30666/TCP,8080:32417/TCP	22h
stable-grafana	LoadBalancer	10.43.187.118	<pending>	80:32721/TCP	22h

Buscando información sobre el problema, éste problema puede deberse al servicio de balanceo de k3s basado en *traefik*. Al comprobar el estado de este servicio, se ve que no está ejecutándose:



```
kubectl -n kube-system get daemonset
```

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
svclb-traefik-7fa54ef6	3	3	3	3	3	<none>	22d
svclb-stable-kube-prometheus-sta-prometheus-c414daa8	3	3	3	3	3	<none>	9m18s
svclb-stable-grafana-805a2279	3	3	0	3	0	<none>	7m17s

El problema se resuelve **reiniciando el daemon de traefik** con el siguiente comando:

```
kubectl -n kube-system rollout restart daemonset svclb-traefik-7fa54ef6
```

```
root@kubemaster1:/# kubectl -n kube-system rollout restart daemonset svclb-traefik-7fa54ef6
daemonset.apps/svclb-traefik-7fa54ef6 restarted
```

Al hacer esto, se consigue que los dos servicios, Grafana y Prometheus, tengan tres IP externas, una por nodo, por lo que ya podríamos acceder desde el exterior.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
stable-kube-state-metrics	ClusterIP	10.43.59.107	<none>	8080/TCP	22h
stable-prometheus-node-exporter	ClusterIP	10.43.24.218	<none>	9100/TCP	22h
stable-kube-prometheus-sta-alertmanager	ClusterIP	10.43.229.56	<none>	9093/TCP, 8080/TCP	22h
stable-kube-prometheus-sta-operator	ClusterIP	10.43.105.48	<none>	443/TCP	22h
alertmanager-operated	ClusterIP	None	<none>	9093/TCP, 9094/TCP, 9094/UDP	22h
prometheus-operated	ClusterIP	None	<none>	9090/TCP	22h
stable-kube-prometheus-sta-prometheus	LoadBalancer	10.43.237.135	192.168.10.11, 192.168.10.12, 192.168.10.13	9090:30666/TCP, 8080:32417/TCP	22h
stable-grafana	LoadBalancer	10.43.187.118	192.168.10.11, 192.168.10.12, 192.168.10.13	80:32721/TCP	22h

Antes de acceder a Grafana, es necesario conocer los datos de usuario y contraseña de acceso, para ello debemos ejecutar lo siguiente:

```
kubectl get secrets -n prometheus
```

NAME	TYPE	DATA	AGE
stable-kube-prometheus-sta-admission	Opaque	3	2d21h
alertmanager-stable-kube-prometheus-sta-alertmanager	Opaque	1	2d21h
stable-kube-prometheus-sta-prometheus	Opaque	0	2d21h
stable-grafana	Opaque	3	2d21h
alertmanager-stable-kube-prometheus-sta-alertmanager-generated	Opaque	1	2d21h
alertmanager-stable-kube-prometheus-sta-alertmanager-tls-assets-0	Opaque	0	2d21h
alertmanager-stable-kube-prometheus-sta-alertmanager-web-config	Opaque	1	2d21h
prometheus-stable-kube-prometheus-sta-prometheus	Opaque	1	2d21h
prometheus-stable-kube-prometheus-sta-prometheus-tls-assets-0	Opaque	1	2d21h
prometheus-stable-kube-prometheus-sta-prometheus-web-config	Opaque	1	2d21h
sh.helm.release.v1.stable.v1	helm.sh/release.v1	1	2d21h

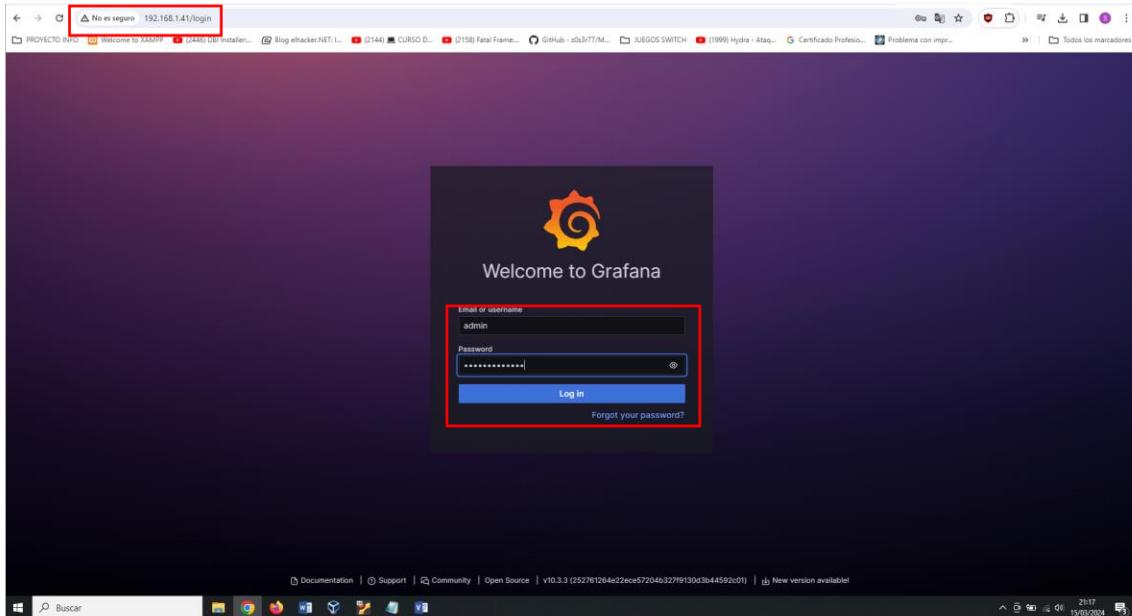
Las contraseñas, por defecto, están ocultas. Lanzando el siguiente comando conseguimos que decodifique la contraseña en *base64* y nos la muestre en pantalla con el *echo*. El usuario por defecto es “*admin*”.

Esta sería la contraseña de acceso del usuario *admin*.



```
root@kubemaster1:/home/master1# kubectl -n prometheus get secret stable-grafana -o jsonpath=".data.admin-password" | base64 --decode ; echo  
prom-operator  
root@kubemaster1:/home/master1#
```

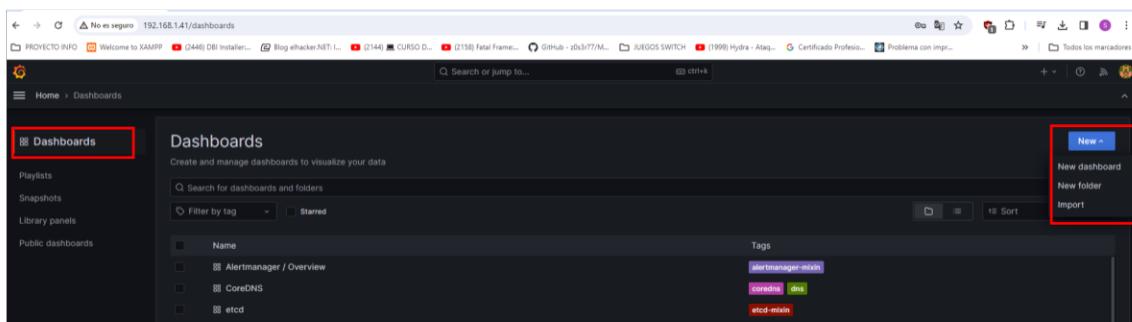
En mi máquina local, accedemos con cualquiera de las IP externas a través del navegador:
<http://192.168.1.40>, 41 o 42.

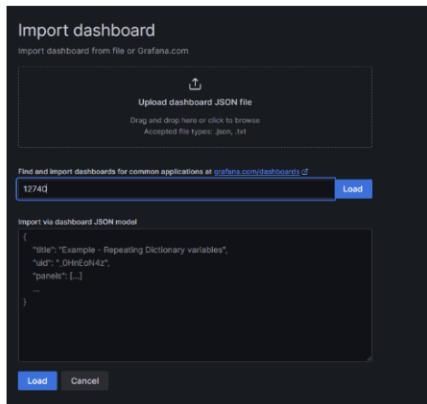


Introduciendo las credenciales anteriormente recuperadas se accede a la página principal de Grafana.

- Importar Dashboard.

En el menú general, iremos a Dashboard y después pulsaremos en “New”.





Puede que nos remita un “*Error Bad Gateway*” al intentar importar el Dashboard.

```
logger=grafana.update.checker t=2024-03-14T18:02:06.008877031Z level=error msg="Update check failed" error="failed to get stable version from grafana.com: Get <https://grafana.com/api/grafana/versions/stable>: dial tcp: lookup grafana.com on 10.42.1.34: timeout" duration=10.001827973ms referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:08:52.861933163Z level=info msg="Request Completed" method=GET path=/api/live/ws status=-1 remote_addr=10.42.1.34 time_ms=0 duration=9.38291ms size=0 referer="grafana.com"
```

```
logger=handlers t=2024-03-14T18:08:52.861933163Z level=error msg="Internal server error" error="*[plugin.downstreamError] client: failed to call resources: error querying resource" referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=9.38291ms referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:08:52.861933163Z level=error msg="Request Completed" method=POST path=/api/datasources/uid/prometheus/resources/api/v1/series status=500 remote_addr=10.42.1.34 time_ms=10043 referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=10.0430007ms size=16 referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:02.833023056Z level=info msg="Initialized channel handler" channel=grafana/dashboard/uid=d581e46e4ec7ba40a07646395f7b2 address=grafana/dashboard/uid=d581e46e4ec7ba40a07646395f7b23 referer="grafana.com"
```

```
logger=grafana.update.checker t=2024-03-14T18:09:02.874880213Z level=error msg="Internal server error" error="*[plugin.downstreamError] client: failed to call resources: error querying resource" referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=10.0430007ms size=16 referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:02.874880213Z level=error msg="Request Completed" method=POST path=/api/datasources/uid/prometheus/resources/api/v1/series status=500 remote_addr=10.42.1.34 time_ms=10043 referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=10.0430007ms size=16 referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:02.881551912Z level=error msg="Internal server error" error="*[plugin.downstreamError] client: failed to call resources: error querying resource" referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=10.0430007ms size=16 referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:02.883311682Z level=error msg="Request Completed" method=POST path=/api/datasources/uid/prometheus/resources/api/v1/series status=500 remote_addr=10.42.1.34 time_ms=10114 referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=10.0430007ms size=16 referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:02.884020232Z level=error msg="Request Completed" method=POST path=/api/datasources/uid/prometheus/resources/api/v1/series status=500 remote_addr=10.42.1.34 time_ms=10089 referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=10.0430007ms size=16 referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:02.885202382Z level=error msg="Internal server error" error="*[plugin.downstreamError] client: failed to call resources: error querying resource" referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=10.0430007ms size=16 referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:02.884466879Z level=error msg="Request Completed" method=POST path=/api/datasources/uid/prometheus/resources/api/v1/series status=500 remote_addr=10.42.1.34 time_ms=10106 referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=10.0430007ms size=16 referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:02.885536395Z level=error msg="Internal server error" error="*[plugin.downstreamError] client: failed to call resources: error querying resource" referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=10.0430007ms size=16 referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:02.885936957Z level=error msg="Request Completed" method=POST path=/api/datasources/uid/prometheus/resources/api/v1/series status=500 remote_addr=10.42.1.34 time_ms=10097 referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=10.0430007ms size=16 referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:12.824045682Z level=info msg="Request Completed" method=POST path=/api/datasources/uid/prometheus/resources/api/v1/series status=500 remote_addr=10.42.1.34 time_ms=9328 duration=9.3286ms referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=9.3286ms referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:12.824045682Z level=error msg="Request Completed" method=POST path=/api/datasources/uid/prometheus/resources/api/v1/series status=400 remote_addr=10.42.1.34 time_ms=9310 duration=9.3109ms referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=9.3109ms referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:12.830378192Z level=info msg="Request Completed" method=POST path=/api/datasources/uid/prometheus/resources/api/v1/series status=400 remote_addr=10.42.1.34 time_ms=9329 duration=9.3296ms referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=9.3296ms referer="grafana.com"
```

```
logger=context userId=1 orgId=1 uname=admin t=2024-03-14T18:09:12.942288952Z level=error msg="Internal server error" error="*[plugin.downstreamError] client: failed to call resources: error querying resource" referer="grafana.com"
```

```
logger=Post <http://stable-kube-prometheus-sta-prometheus:9090/api/v1/series>: dial tcp: lookup stable-kube-prometheus-sta-prometheus on 10.42.1.34: timeout" duration=10.046035451ms size=16 referer="grafana.com"
```

Para diagnosticar y solucionar el problema comprobamos los *logs* de Grafana con siguiente comando:

```
kubectl -n prometheus logs stable-grafana-567c9b5995-5jj5w
```

Podemos ver que se trata de un problema resolviendo direcciones. Al comprobar todos los nodos del *namespace* de Prometheus hay varios que no levantan. En concreto el pod *coredns-6799fbcd5-cvcd4*, es el que está manejando las direcciones DNS y nos retorna “ErrImagePull”, que indica que Kubernetes no puede descargar la imagen de contenedor especificada para el pod.



NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	svclb-traefik-7fa54ef6-96bm4	0/2	Pending	0	3h27m
kube-system	helm-install-traefik-crd-2kshh	0/1	Completed	0	24d
kube-system	svclb-traefik-7fa54ef6-8cn9n	0/2	Pending	0	3h27m
kube-system	svclb-traefik-7fa54ef6-vdpb2	0/2	Pending	0	3h19m
kube-system	helm-install-traefik-bxzww	0/1	Completed	1	24d
prometheus	stable-prometheus-node-exporter-5n6rm	1/1	Running	20 (66m ago)	2d22h
kube-system	svclb-stable-kube-prometheus-sta-prometheus-c414daa8-7lkr5	2/2	Running	22 (66m ago)	47h
kube-system	svclb-stable-grafana-805a279-qtgsw	1/1	Running	11 (66m ago)	47h
prometheus	stable-grafana-567c9b5995-5jj5w	1/1	Running	19 (66m ago)	24d
kube-system	local-path-provisioner-84db5d44d9-m62wd	3/3	Running	17 (66m ago)	5h22m
prometheus	stable-prometheus-node-exporter-spkd5	1/1	Running	38 (65m ago)	24d
prometheus	stable-kube-prometheus-sta-operator-697547bc9d-5mtgr	1/1	Running	6 (65m ago)	2d22h
kube-system	svclb-stable-grafana-805a279-h9wh4	1/1	Running	5 (65m ago)	47h
kube-system	svclb-stable-kube-prometheus-sta-prometheus-c414daa8-ktfgk	2/2	Running	10 (65m ago)	47h
prometheus	alertmanager-stable-kube-prometheus-sta-alertmanager-0	2/2	Running	4 (65m ago)	80m
kube-system	svclb-stable-kube-prometheus-sta-prometheus-c414daa8-9ts28	2/2	Running	11 (65m ago)	47h
kube-system	metrics-server-67c658944b-p4jws	1/1	Running	40 (65m ago)	24d
prometheus	stable-kube-state-metrics-5988bf67d5-49pvr	1/1	Running	11 (65m ago)	5h21m
prometheus	prometheus-stable-kube-prometheus-sta-prometheus-0	2/2	Running	3 (65m ago)	77m
prometheus	stable-prometheus-node-exporter-ngst7	1/1	Running	7 (65m ago)	2d22h
Default	ubuntu	1/1	Running	0	57m
kube-system	coredns-6799fbcd5-cvc4	0/1	ErrImagePull	0	9s

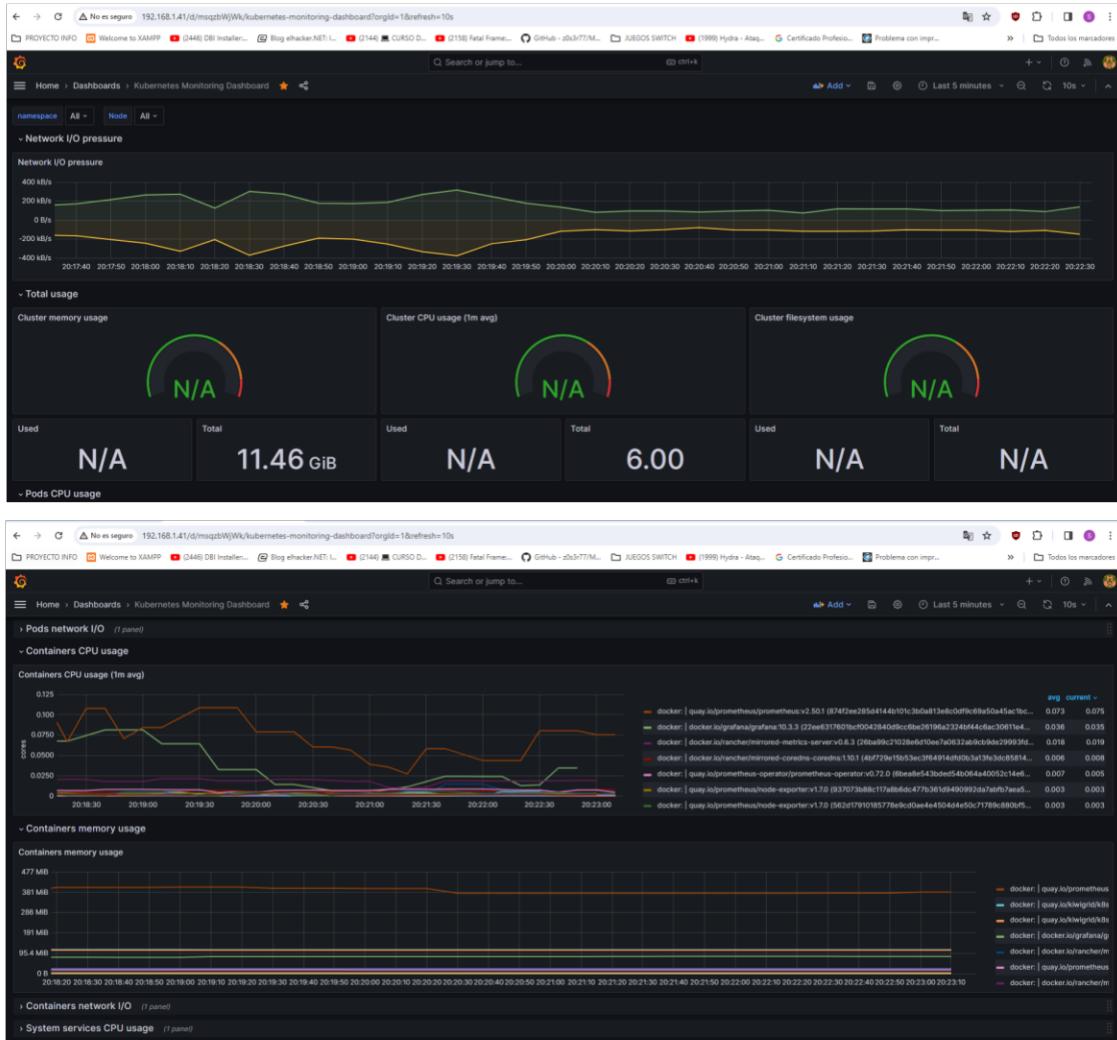
Se ejecuta el siguiente comando para obtener más información de dicho pod:

```
describe pod coredns-6799fbcd5-cvc4
```

```
Events:
Type Reason Age From Message
Normal Scheduled 44s default-scheduler successfully assigned kube-system/coredns-6799fbcd5-cvc4 to kubewerker1
Normal Pulling 246 (x2 over 43s) kubelet Pulling image "rancher/mirrored-coredns:coredns:1.10.1"
Warning Failed 19s (x2 over 38s) kubelet failed to pull image "rancher/mirrored-coredns:coredns:1.10.1": failed to pull and unpack image "docker.io/rancher/mirrored-coredns:coredns:1.10.1" from docker.io: failed to resolve reference "docker.io/rancher/mirrored-coredns:coredns:1.10.1": failed to do request: Head "https://registry-1.docker.io/v2/rancher/mirrored-coredns:coredns/manifests/1.10.1": dial tcp: lookup registry-1.docker.io on 127.0.0.5:5631: connect: connection refused
Warning Failed 19s (x2 over 38s) kubelet Error: ErrImagePull
Normal Backoff 4s (x2 over 38s) kubelet Back-off pulling image "rancher/mirrored-coredns:coredns:1.10.1"
Warning Failed 4s (x2 over 38s) kubelet Error: ImagePullBackoff
```

Todo apunta a que Netplan interfiere en los *nameserver* en el *resolv.conf*, por lo que es necesario asignar los *nameserver* de google (8.8.8.8, 8.8.4.4) en *.yaml* de configuración de Netplan. Hecho esto, se elimina el pod *coredns-6799fbcd5-cvc4* y de manera automática se levanta otro, pudiendo así importar el Dashboard sin problema.

En las siguientes capturas podemos observar que ya podemos movernos por Grafana, importar Dashboards y comprobar su funcionamiento pudiendo verificar que ya ha comenzado a monitorear los recursos de nuestro clúster.



4.1.5.3.3. Instalación Wordpress

En este punto vamos a comprobar una manera muy sencilla la instalación de otro aplicativo, en este caso Wordpress, a través de Helm.

Primero buscaremos el paquete para ver cuál es la última versión en los repositorios añadidos:

```
helm search repo wordpress
```

```
2. Master1 3. Worker1 4. Worker2
root@kubemaster1:/home/master1# helm search repo wordpress
NAME        CHART VERSION  APP VERSION  DESCRIPTION
artifact-hub/wordpress  9.0.3      5.3.2       DEPRECATED Web publishing platform for building...
bitnami/wordpress   20.1.2      6.4.3       WordPress is the world's most popular blogging ...
bitnami/wordpress-intel 2.1.31    6.1.1       DEPRECATED WordPress for intel is the most popu...
stable/wordpress    9.0.3      5.3.2       DEPRECATED Web publishing platform for building...
```



Recuperados los datos de repositorio, nombre de chart y versión, procedemos a la instalación con el siguiente comando:

```
helm install bitnami/wordpress --generate-name
```

Se nos da la información de la versión instalada, cómo averiguar usuario y contraseña, URL de acceso externo a Wordpress, etc.

```
root@kubemaster1:/home/master1# helm install bitnami/wordpress --generate-name
NAME: wordpress-1710777568
LAST DEPLOYED: Mon Mar 18 15:59:35 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
  ✓ Deployment NAME: wordpress
  ✓ CHART VERSION: 20.1.2
  ✓ APP VERSION: 6.4.3
  ** Please be patient while the chart is being deployed **
Your WordPress site can be accessed through the following DNS name from within your cluster:
  wordpress-1710777568.default.svc.cluster.local (port 80)
To access your WordPress site from outside the cluster follow the steps below:
1. Get the WordPress URL by running these commands:
  NOTE: It may take a few minutes for the LoadBalancer IP to be available.
  Watch the status with: `kubectl get svc --namespace default -w wordpress-1710777568`
  export SERVICE_IP=$(kubectl get svc --namespace default -w wordpress-1710777568 -o template --template="{{ range (index .status.loadBalancer.ingress 0) }}{{ . }}{{ end }}")
  echo "WordPress Admin URL: http://$SERVICE_IP/admin"
  echo "WordPress Admin URL: http://$SERVICE_IP/admin"
2. Open a browser and access WordPress using the obtained URL.
3. Login with the following credentials below to see your blog:
  echo Username: user
  echo Password: $(kubectl get secret --namespace default wordpress-1710777568 -o jsonpath='{.data.wordpress-password}' | base64 -d)
WARNING: There are "resources" sections in the chart not set. Using "resourcesPreset" is not recommended for production. For production installations, please set the following values according to your workload requirements
+info https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
root@kubemaster1:/home/master1#
```

Al ejecutar el comando `kubectl get svc --namespace default -w wordpress-1710777568` para ver el estado del servicio, compruebo que no me está dando una IP externa y el **LoadBalancer** no se está ejecutando, pero MariaDB y el Wordpress están levantados.

```
root@kubemaster1:/home/master1# kubectl get svc --namespace default -w wordpress-1710777568
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
wordpress-1710777568   LoadBalancer   10.43.55.150  <pending>     80:31276/TCP,443:30804/TCP   4m37s

root@kubemaster1:/home/master1# kubectl get all
NAME                           READY   STATUS    RESTARTS   AGE
pod/ubuntu                      0/1     Unknown   0          6d
pod/wordpress-1710777568-mariadb-0 1/1     Running   1 (76m ago)  2d1h
pod/wordpress-1710777568-65b5b7bcd4-jw87k 1/1     Running   4 (73m ago)  2d1h

NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/kubernetes   ClusterIP  10.43.0.1    <none>        443/TCP         30d
service/wordpress-1710777568-mariadb  ClusterIP  10.43.131.179 <none>        3306/TCP       2d1h
service/wordpress-1710777568   LoadBalancer  10.43.55.150  <pending>     80:31276/TCP,443:30804/TCP  2d1h

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/wordpress-1710777568  1/1      1           1          2d1h

NAME           DESIRED   CURRENT   READY   AGE
replicaset.apps/wordpress-1710777568-65b5b7bcd4  1         1         1         2d1h

NAME           READY   AGE
statefulset.apps/wordpress-1710777568-mariadb  1/1      2d1h
root@kubemaster1:/home/master1#
```

El balanceo debería ser automático, pero puede darse este error, que el service/wordpress-1710777568 EXTERNAL-IP se quede `<pending>`.

Nuevamente aplicamos el mismo workaround descrito en el caso de Prometheus y Grafana, y podremos acceder a wordpress. (Página 21)



4.2. CREACIÓN Y CONFIGURACIÓN DE CLÚSTER DE KUBERNETES EN AZURE CON AKS, HELM Y TERRAFORM.

4.2.1. ¿Qué es Azure y AKS?



Azure es una plataforma de servicios en la nube ofrecida por Microsoft. Proporciona una amplia gama de servicios de computación, almacenamiento, redes, bases de datos, inteligencia artificial, Internet de las cosas (IoT), etc., que permite a las empresas desarrollar, implementar y administrar aplicaciones y servicios a través de la infraestructura de Microsoft distribuida globalmente.



Azure Kubernetes Service (AKS) es un servicio de Azure que simplifica la implementación, la administración y la escalabilidad de los clústeres de Kubernetes en la nube de Azure.

AKS ofrece características como la orquestación de contenedores, la escalabilidad automática, la integración con herramientas de desarrollo y DevOps, la monitorización y la seguridad integrada sin tener que preocuparse por la complejidad de administrar la infraestructura, lo que permite desarrollar, implementar y escalar aplicaciones de manera más ágil y eficiente.

4.2.2. Requisitos previos

En este caso, basta con tener una cuenta de estudiante para disponer de los recursos básicos y logarnos con ella. **Azure for Students** ofrece créditos gratuitos para estudiantes que se pueden usar para explorar y aprender sobre servicios de Azure durante un período específico.

4.2.2.1. Instalación en local de CLI de Azure en Windows

Descargamos el ejecutable correspondiente a nuestro S.O a través de la siguiente dirección:
<https://learn.microsoft.com/es-es/cli/azure/install-azure-cli-windows?tabs=azure-cli>

Verificamos la instalación desde PowerShell ejecutando el siguiente comando, que nos mostrará la versión instalada: `az --version`



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\MSI> az --version
azure-cli          2.59.0
core               2.59.0
telemetry          1.1.0

Dependencies:
msal                1.27.0
azure-mgmt-resource 23.1.0b2
```

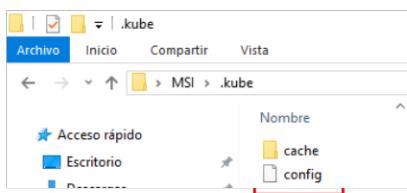
Debemos comprobar que se realiza la conexión remota con Azure con el siguiente comando: `az login` y si todo es correcto podemos visualizar una pantalla como la siguiente:

```
Windows PowerShell
PS C:\Users\MSI> az login
A web browser has been opened at https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize. Please continue the login in the web browser. If no web browser is available or if the web browser fails to open, use device code flow with `az login --use-device-code`.
[{"cloudName": "AzureCloud", "homeTenantId": "e2d47e54-663d-49d0-9b6c-1a4146c56a08", "id": "3214bfc2-f1af-44ba-bf7a-42513fc23d14", "isDefault": true, "managedByTenants": [], "name": "Azure for Students", "state": "Enabled", "tenantId": "e2d47e54-663d-49d0-9b6c-1a4146c56a08", "user": {"name": "festichan2611@linkiafp.online", "type": "user"}}]
```

4.2.2.2. Instalación de Kubectl en Windows



kubectl es una herramienta de línea de comandos que se utiliza para interactuar con clústeres de Kubernetes. En Windows, kubectl puede ser instalado y utilizado de manera similar a como se utiliza en otros sistemas operativos como Linux o macOS. Con kubectl se pueden gestionar aplicaciones, recursos y realizar tareas de administración y monitoreo en tu entorno de Kubernetes directamente desde la línea de comandos sin necesidad de usar los CLI de Azure, AWS o Google Cloud.



En la carpeta de usuario se tenemos a su vez la carpeta `.kube`, donde se almacenará un fichero `config`, el cual configura la conexión al clúster de Kubernetes, ya sea local o, como en este caso, desde Azure.

<https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/>



4.2.2.3. Instalación de Chocolatey



Chocolatey es una plataforma de gestión de paquetes para Windows que facilita la instalación, actualización y administración de software. Funciona de manera similar a los gestores de paquetes en sistemas operativos como Linux, como *apt* en Ubuntu o *yum* en CentOS.

Los paquetes de software en Chocolatey son scripts de **PowerShell** que automatizan el proceso de instalación y configuración de aplicaciones. Esto permite instalar software sin necesidad de interactuar con instaladores gráficos o descargar archivos de instalación manualmente.

Además, Chocolatey ofrece características como la gestión de versiones, la instalación silenciosa, la integración con herramientas de administración de sistemas y la capacidad de crear paquetes personalizados.

Abrimos Power Shell y lazamos el siguiente comando para la instalación:

```
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

Debemos también configurar la variable de entorno PATH:

```
C:\ProgramData\chocolatey\choco.exe;C:\ProgramData\chocolatey\bin
```

Probamos si está instalado con el comando: **choco**

```
Administrator: Windows PowerShell
PS C:\Windows\system32> choco
Chocolatey v2.2.2
Please run 'choco -?' or 'choco <command> -?' for help menu.
PS C:\Windows\system32>
```

4.2.3. Creación de clúster Kubernetes con AKS

Accedemos a la web de Azure utilizando las credenciales de estudiante. Creamos el clúster siguiendo los pasos de la configuración guiada que proporciona Microsoft:

<https://learn.microsoft.com/es-es/azure/aks/learn/quick-kubernetes-deploy-portal?tabs=azure-cli#create-an-aks-cluster>



Nota: Durante la instalación he decidido **no implementar Grafana directamente como ofrece el propio Azure**, ya que más adelante se mostará la forma de implementar aplicaciones sin necesidad del provider, aplicando exactamente los mismos pasos que en la instalación sobre el cluster de K3s.

Una vez finalice la configuración guiada podremos acceder a la vista general y las propiedades de nuestro clúster.

The screenshot shows the Azure portal interface for the deployment '891607389 | Información general'. It displays a green checkmark indicating 'Se completó la implementación'. Key details include the subscription ('Azure for Students'), resource group ('AKS-grupo-recursos'), start time ('23/4/2024, 19:00:54'), and correlation ID ('0069b39f-a400-4522-8226-6856ab496ac8'). On the right, there are promotional cards for 'Cost Management' and 'Microsoft Defender for Cloud'.

Vista general del clúster:

The screenshot shows the Azure portal interface for the Kubernetes cluster 'AKS-cluster-proyecto'. The left sidebar lists various management options like 'Información general', 'Control de acceso (IAM)', and 'Recursos de Kubernetes'. The main panel displays cluster details such as state ('En ejecución (running)'), subscription ('Azure for Students'), location ('West Europe'), and node group ('1 grupo de nodos'). It also shows network settings ('Azure CNI'), service endpoints ('aks-cluster-proyecto-dns-nauuanhk.cnp.westeurope.azurek8s.io'), and configuration parameters. A 'Comenzar' button is visible at the top.



4.2.3.1. Obtención de credenciales para acceder desde el exterior sin usar Cloud Shell de Azure

Accedo al Cloud Shell en formato Bash y se ejecuta el siguiente comando para acceder a las credenciales que necesitaremos:

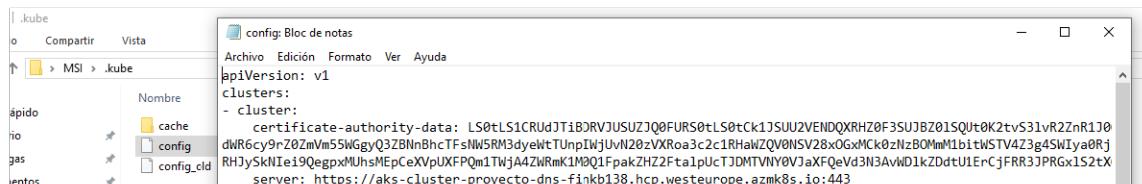
```
az aks get-credentials --resource-group AKS-grupo-recursos --name AKS-cluster-proyecto --admin
```

Para mostrar la información que contiene en fichero *config* ubicado en /home/susana/.kube/config, que después copiaremos en nuestro .kube/config en local, usaremos el siguiente comando:

```
cat /home/susana/.kube/config
```

```
susana [ ~ ]$ az aks get-credentials --resource-group AKS-grupo-recursos --name AKS-cluster-proyecto --admin
Merged "AKS-cluster-proyecto-admin" as current context in /home/susana/.kube/config
susana [ ~ ]$ cat /home/susana/.kube/config
apiVersion: v1
clusters:
- cluster:
  - certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUU2VENDQXRHZ0F3SUJBZ01SQUt0K2tvS31vR2ZnR1J00
```

Con la información del *config* de kubernetes desplegado en AKS Azure, copiamos todo y nos dirigimos a la carpeta ./kube de nuestro local, creamos un *config_old* por seguridad, y creamos un nuevo fichero *config* con toda la información que copiamos anteriormente.



4.2.3.2. Verificación de acceso desde nuestro equipo local

Ejecutamos PowerShell, nos logamos de nuevo y establecemos la cuenta con los comandos:

```
az login
az account set --subscription="3214bfc2-f1af-44ba-bf7a-42513fc23d14"
```



Ejecutamos el siguiente comando desde nuestro Power Shell local y comprobamos que tenemos acceso a nuestros nodos de AKS desde el exterior:

```
Windows PowerShell
PS C:\Users\MSI> kubectl get nodes
NAME           STATUS  ROLES   AGE    VERSION
aks-agentpool-34063058-vmss000000  Ready   agent   23m   v1.28.5
aks-agentpool-34063058-vmss000001  Ready   agent   23m   v1.28.5
```

4.2.4. Instalación de Helm en Windows

Una vez instalado Chocolatey en nuestro equipo local, para instalar Helm sólo debemos introducir el siguiente comando:

```
choco install kubernetes-helm
```

Finalizada la instalación, podemos comprobar que todo está correcto con un comando de ejemplo: `helm ls`. Podemos comprobar que se ha creado la estructura.

```
Administrator: Windows PowerShell
PS C:\Windows\system32> choco
Chocolatey v2.2.2
Please run 'choco -?' or 'choco <command> -?' for help menu.
PS C:\Windows\system32> choco install kubernetes-helm
Chocolatey v2.2.2
Installing the following packages:
kubernetes-helm
By installing, you accept licenses for the packages.
Progress: Downloading kubernetes-helm 3.14.3... 100%
kubernetes-helm v3.14.3 [Approved]
kubernetes-helm package files install completed. Performing other installation steps.
The package kubernetes-helm wants to run 'chocolateyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N)o/[P]rint): y
Downloading kubernetes-helm 64 bit
  from 'https://get.helm.sh/helm-v3.14.3-windows-amd64.zip'
Progress: 100% - Completed download of C:\Users\MSI\AppData\Local\Temp\chocolatey\kubernetes-helm\3.14.3\helm-v3.14.3-windows-amd64.zip (15.76 MB).
Download of helm-v3.14.3-windows-amd64.zip (15.76 MB) completed.
Hashes match.
Extracting C:\Users\MSI\AppData\Local\Temp\chocolatey\kubernetes-helm\3.14.3\helm-v3.14.3-windows-amd64.zip to C:\ProgramData\chocolatey\lib\kubernetes-helm\tools...
C:\ProgramData\chocolatey\lib\kubernetes-helm\tools
Environment Vars (like PATH) have changed. Close/reopen your shell to see the changes (or in powershell/cmd.exe just type `refreshenv`).
ShimGen has successfully created a shim for helm.exe.
The install of kubernetes-helm was successful.
Software installed to 'C:\ProgramData\chocolatey\lib\kubernetes-helm\tools'

Chocolatey installed 1/1 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
PS C:\Windows\system32> helm ls
NAME      NAMESPACE   REVISION      UPDATED STATUS  CHART      APP VERSION
PS C:\Windows\system32>
```



4.2.4.1. Instalación de Prometheus-Grafana con Helm

Para realizar la instalación de Prometheus-Grafana desde nuestro Power Shell, repetiré los pasos del apartado 4.1.5.3.1. *Instalación de Prometheus – Grafana (página 19)*, el resumen de los comandos lanzados sería el siguiente:

```
kubectl create namespace prometheus
helm search repo prometheus-community/kube-prometheus-stack
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm install stable prometheus-community/kube-prometheus-stack --version 57.0.1 -n
prometheus
```

Podemos comprobar con los siguientes comandos, que se han creado todos los pods, pero deberemos modificar el **TYPE** en los *yaml* **stable-kube-prometheus-sta-prometheus** y **stable-grafana** para indicar tipo **Loadbalancer** y que genere una IP con la que podamos acceder a la aplicación desde fuera de la red interna de nuestro clúster en Azure.

The screenshot shows two windows side-by-side. The left window is a Windows PowerShell session with the command `kubectl get svc -n prometheus` running. The output lists several services, including 'stable-grafana' and 'stable-kube-prometheus-sta-prometheus', both of which have their 'type' field set to 'ClusterIP'. The right window is a text editor showing the YAML configuration for the 'stable-kube-prometheus-sta-prometheus' service. A red box highlights the 'spec' section, specifically the 'clusterIP' and 'loadBalancer' fields. Another red box highlights the 'type' field, which is currently set to 'LoadBalancer'.

```
PS C:\Users\MSI> kubectl get svc -n prometheus
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP
alertmanager-operated ClusterIP  None          <none>
prometheus-operated ClusterIP  None          <none>
stable-grafana   ClusterIP  10.0.144.195  <none>
stable-kube-prometheus-sta-alertmanager ClusterIP  10.0.135.230  <none>
stable-kube-prometheus-sta-operator    ClusterIP  10.0.36.164   <none>
stable-kube-prometheus-sta-prometheus LoadBalancer 10.0.27.79   172.21.195.88
                                         9090:TCP,8080/TCP 4m8s
                                         443/TCP           4m8s
stable-kube-state-metrics ClusterIP  10.0.72.20   <none>
stable-prometheus-node-exporter   ClusterIP  10.0.253.224 <none>
9100/TCP           4m8s
PS C:\Users\MSI> kubectl edit svc stable-grafana -n prometheus
```

```
namespace: prometheus
resourceVersion: "6334"
uid: 3777953c-89c0-4e70-9d52-e03efce506ed
spec:
  clusterIP: 10.0.144.195
  clusterIPs:
  - 10.0.144.195
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http-web
    port: 80
    protocol: TCP
    targetPort: 3000
  selector:
    app.kubernetes.io/instance: stable
    app.kubernetes.io/name: grafana
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer: {}
```

Una vez hecho este cambio, si lanzamos de nuevo el comando `kubectl get svc -n prometheus`, podemos ver que ya nos realiza el balaceo y podemos comprobar si accedemos.



NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
alertmanager-operated	ClusterIP	None	<none>	9093/TCP,9094/TCP,9094/UDP	4m2s
prometheus-operated	ClusterIP	None	<none>	9090/TCP	4m2s
stable-grafana	ClusterIP	10.0.144.195	<none>	80/TCP	4m8s
stable-kube-prometheus-sta-alertmanager	ClusterIP	10.0.135.230	<none>	9093/TCP,8080/TCP	4m8s
stable-kube-prometheus-sta-operator	ClusterIP	10.0.36.164	<none>	443/TCP	4m8s
stable-kube-prometheus-sta-prometheus	LoadBalancer	10.0.27.79	172.211.195.88	9090:30928/TCP,8080:31861/TCP	4m8s
stable-kube-state-metrics	ClusterIP	10.0.72.20	<none>	8080/TCP	4m8s
stable-prometheus-node-exporter	ClusterIP	10.0.253.224	<none>	9100/TCP	4m8s

Prueba de acceso a Prometheus:

The screenshot shows the Prometheus web interface. The URL in the address bar is `172.211.195.88:9090/graph?g0.expr=&g0.tab=1&g0.display_mode=lines&g0.show_exemplars=0&g0.range_input=1h`. The interface includes a navigation bar with links like 'PROJECTO INFO', 'Microsoft Certified...', 'Curso AZ-900T00...', 'Education - Micros...', '(2899) Virtualized I...', 'Welcome to XAMPP', '(2719) 010 Instalac...', and '(2719) 010 Instalac...'. Below the bar, there's a search bar with placeholder text 'Expression (press Shift+Enter for newlines)', several checkboxes for configuration options, and tabs for 'Table' and 'Graph'. A date range selector shows 'Evaluation time' with arrows for navigation. The main area displays the message 'No data queried yet'. At the bottom left is a blue button labeled 'Add Panel'.

Prueba de acceso a Grafana:

The screenshot shows the Grafana login screen. The URL in the address bar is `172.211.196.247/login`. The page features the Grafana logo at the top, followed by the text 'Welcome to Grafana'. It has two input fields: 'Email or username' containing 'jemail or username' and 'Password' containing 'password'. To the right of the password field is a visibility toggle icon. Below the inputs is a large blue 'Log in' button. At the bottom right of the form is a link 'Forgot your password?'



4.3. EJEMPLOS DE USO DE TERRAFORM

4.3.1. ¿Qué es Terraform?



Terraform es una herramienta de **infraestructura como código (IaC)** creada por HashiCorp. Permite definir y administrar la infraestructura de manera automatizada utilizando archivos de configuración.

Terraform es independiente de cualquier proveedor de nube específico, lo que significa que puede ser utilizado para gestionar la infraestructura en variedad de entornos, incluyendo AWS, Azure, Google Cloud Platform, y otros proveedores de nube, así como infraestructuras locales y servicios de terceros.

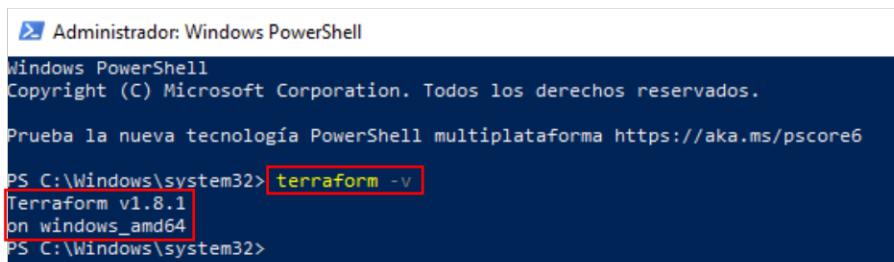
Terraform utiliza archivos de texto para describir la infraestructura y establecer variables. El lenguaje de los ficheros de configuración de Terraform se llama **HashiCorp Configuration Language (HCL)**. Los ficheros se deberán crear con la extensión ".tf".

4.3.2. Instalación de Terraform en Windows

Podemos descargar el software y las instrucciones de configuración de la siguiente URL:
<https://www.terraform.io/downloads.html>

Debemos añadir a la variable PATH la ruta donde hayamos descomprimido el fichero descargado.

Podemos verificar si todo está correcto y la versión instalada con el siguiente comando en PowerShell: `terraform -v`



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

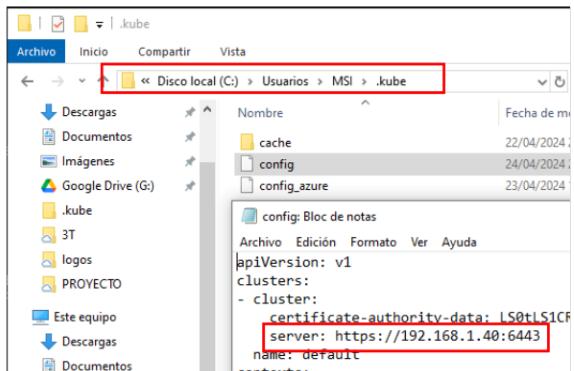
PS C:\Windows\system32> terraform -v
Terraform v1.8.1
on windows_amd64
PS C:\Windows\system32>
```



4.3.3. Uso de Terraform con un clúster local virtualizado (K3s)

Una vez configurado Terraform en nuestro S.O local, se realizará una prueba de uso con el clúster virtualizado anteriormente.

El formato de los archivos de configuración puede estar en dos formatos: formato Terraform y JSON. El formato de Terraform es más legible, admite comentarios y es el formato generalmente recomendado para la mayoría de los archivos de Terraform.



Previamente se configuró **kubectl** en Windows, con lo que ya disponemos del directorio con el fichero de **.kube/config**. Debemos cambiar del fichero config de Azure al que se generó en la creación del cluster local con K3s.

4.3.3.1. Creación de directorio y ficheros de Terraform

Se creará una carpeta con el nombre “terraform-tfg” que contendrá los siguientes archivos .tf:

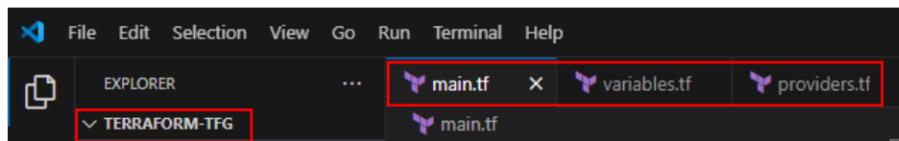
- **main.tf**: Este fichero contiene la configuración principal de la infraestructura que se quiere gestionar. Aquí es donde se definen los recursos, como instancias de máquinas virtuales, bases de datos, redes, etc. También es donde se especificará cómo estos recursos deben configurarse: tamaño, región, etiquetas, etc.
- **variables.tf**: En este fichero, se definen las variables que se utilizarán en la configuración principal (main.tf). Las variables pueden ser utilizadas para parametrizar la configuración, lo que hace que sea más fácil de mantener y reutilizar. Por ejemplo, se pueden definir variables para el nombre de la región, el tamaño de la instancia, las credenciales de acceso, etc.
- **providers.tf**: Este fichero se utiliza para especificar los proveedores de infraestructura que Terraform utilizará para crear y gestionar los recursos. Un proveedor en Terraform es un complemento que permite interactuar con un servicio en la nube específico o



infraestructura local. En este fichero se configuran los proveedores que se utilizarán en tu configuración, así como cualquier configuración global que necesiten, como las credenciales de acceso.

Estos ficheros forman la base de una configuración típica de Terraform y son fundamentales para definir la infraestructura como código.

Para trabajar con la edición de los archivos `.tf` usaré **Visual Studio Code**, ya que además nos permite ejecutar terminales.



4.3.3.2. Edición de los ficheros

Antes de todo, y para evitar fallos y/o conflictos en el despliegue automatizado, previamente hemos hecho un borrado de las aplicaciones desplegadas con anterioridad en nuestro clúster.

Vamos a repetir la instalación de Prometheus y Wordpress, pero esta vez de manera automatizada y totalmente desatendida. En las siguientes capturas aparece la configuración mínima de cada uno de los ficheros `.tf`.

Inicialmente sólo se configuran los ficheros para lanzar Prometheus, ya que posteriormente se añadirá Wordpress para dar un ejemplo de cómo funcionaría Terraform si vamos añadiendo más aplicaciones.

En el `main.tf` especificamos el recurso:

```
1 resource "helm_release" "prometheus" {
2   name      = var.prometheus_helm_name
3   repository = var.prometheus_helm_repository
4   chart      = var.prometheus_helm_release
5   version    = var.prometheus_chart_version
6   namespace  = var.prometheus_namespace
7   create_namespace = true
8   timeout    = 2000
9 }
```

A screenshot of the Visual Studio Code interface showing the 'main.tf' file in the editor. The file contains Terraform code defining a 'helm_release' resource for Prometheus. A specific block of code is highlighted with a red box, starting with 'resource "helm_release" "prometheus" {'. The code defines the resource's name, repository, chart, version, namespace, and timeout parameters, all using variable references like 'var.prometheus_helm_name' and 'var.prometheus_helm_repository'.

En el `providers.tf` configuramos la ruta de nuestro provider que, en este caso, será Helm, y la ruta de kubectl local de nuestro Windows.



```
provider "helm" {
  kubernetes {
    config_path = "C:/Users/MSI/.kube/config"
  }
}
```

En el *variables.tf* se definen las siguientes variables de configuración: namespace, helm name, helm release, helm repository y helm version.

```
variable "prometheus_namespace" {
  default = "prometheus"
}

variable "prometheus_helm_name" {
  default = "prometheus"
}

variable "prometheus_helm_release" {
  default = "kube-prometheus-stack"
}

variable "prometheus_helm_repository" {
  default = "https://prometheus-community.github.io/helm-charts"
}

variable "prometheus_chart_version" {
  default = "57.0.1"
}
```

4.3.3.3. Prueba de funcionamiento I

Como en *providers.tf* apuntamos al *./kube/config* modificado para este clúster, todos las operativas de Terraform se realizarán sobre el clúster de local.

```
PS C:\Users\MSI\terraform-tfg> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/helm...
- Installing hashicorp/helm v2.13.1...
- Installed hashicorp/helm v2.13.1 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\Users\MSI\terraform-tfg>
```



Ejecutando `terraform plan` nos mostrará los elementos que desplegará automáticamente. Con `terraform apply` se realizará el despliegue anteriormente descrito con el comando plan.

The screenshot shows the VS Code interface with the Terraform extension. On the left, the file tree shows a directory structure including `TERRAFORM-TFG`, `main.tf`, `providers.tf`, `README.md`, `terraform.tfstate`, and `variables.tf`. The `main.tf` file is open in the editor, containing a `resource "helm_release" "prometheus"` block. The terminal tab is active, displaying the command `PS C:\Users\MSI\terraform-tfg> terraform apply`. The terminal output shows the execution plan: "Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols: + create". It lists the creation of the `helm_release.prometheus` resource with various configuration parameters. A red box highlights this section of the terminal output. At the bottom of the terminal, it asks "Do you want to perform these actions? Terraform will perform the actions described above. Only 'yes' will be accepted to approve." The user types "Enter a value: yes" and presses Enter.

La instalación se realiza correctamente:

The terminal output shows the progress of the `helm_release.prometheus` creation, with status messages like "Creating...", "Still creating...", and finally "Creation complete after 2m14s [id=prometheus]". At the end, the terminal displays "Apply complete! Resources: 1 added, 0 changed, 0 destroyed." A red box highlights this final message.

Si consultamos directamente sobre nuestro clúster podremos ver que se encuentra desplegado.



```
root@kubemaster1:/home/master1# kubectl get pods -n prometheus
NAME                               READY   STATUS    RESTARTS   AGE
prometheus-prometheus-node-exporter-2mzwn   1/1     Running   0          9m52s
prometheus-prometheus-node-exporter-d9lnw    1/1     Running   0          9m52s
prometheus-prometheus-node-exporter-2zh8     1/1     Running   0          9m52s
prometheus-kube-prometheus-operator-5f475fd846-lhrbt  1/1     Running   0          9m52s
prometheus-kube-state-metrics-59b5d58f8f-56m4j   1/1     Running   0          9m52s
alertmanager-prometheus-kube-prometheus-alertmanager-0  2/2     Running   0          9m39s
prometheus-prometheus-kube-prometheus-prometheus-0  2/2     Running   0          9m38s
prometheus-grafana-5b4d6bf84-s8xrx      3/3     Running   0          9m52s
root@kubemaster1:/home/master1#
```

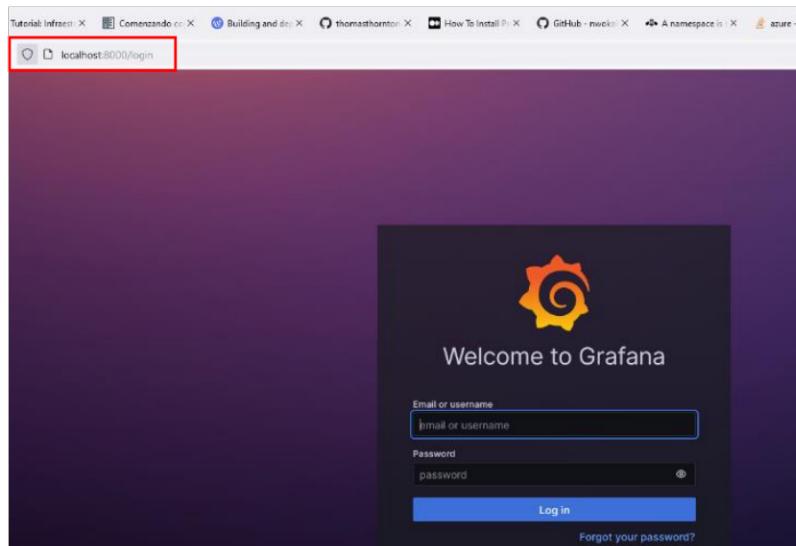
Lo mismo podemos realizar desde el Power Shell de nuestro equipo (ya que el *kubectl* tiene el archivo *config* en la carpeta *.kube*).

```
PS C:\Users\MSI> kubectl get pods -n prometheus
NAME                               READY   STATUS    RESTARTS   AGE
prometheus-prometheus-node-exporter-2mzwn   1/1     Running   0          9m27s
prometheus-prometheus-node-exporter-d9lnw    1/1     Running   0          9m27s
prometheus-prometheus-node-exporter-2zh8     1/1     Running   0          9m27s
prometheus-kube-prometheus-operator-5f475fd846-lhrbt  1/1     Running   0          9m27s
prometheus-kube-state-metrics-59b5d58f8f-56m4j   1/1     Running   0          9m27s
alertmanager-prometheus-kube-prometheus-alertmanager-0  2/2     Running   0          9m14s
prometheus-prometheus-kube-prometheus-prometheus-0  2/2     Running   0          9m13s
prometheus-grafana-5b4d6bf84-s8xrx      3/3     Running   0          9m27s
PS C:\Users\MSI>
```

Para acceder de forma sencilla al componente de Grafana se usará el comando ***kubectl port-forward***. Este comando se utiliza para crear un túnel seguro desde un puerto local hacia un puerto específico en un recurso dentro de tu clúster de Kubernetes. En este caso, *port-forward* se usa para **redirigir el tráfico desde el puerto local 8000 hacia el puerto 80 del servicio prometheus-grafana** en el espacio de nombres (namespace) prometheus.

```
kubectl port-forward -n prometheus svc/prometheus-grafana 8000:80
```

```
PS C:\Users\MSI\terraform-tfg> kubectl port-forward -n prometheus svc/prometheus-grafana 8000:80
Forwarding from 127.0.0.1:8000 -> 3000
Forwarding from [::1]:8000 -> 3000
```



La aplicación levanta sin errores a través del túnel creado con *kubectl port-forward*.

4.3.3.4. Añadir nuevos recursos al despliegue de Terraform

Una vez creada nuestra estructura de ficheros básica, podemos ir añadiendo más aplicaciones según nuestras necesidades sin tener que eliminar nada de lo que tengamos previamente, puesto que Terraform detecta los recursos ya desplegados en los obviará de siguientes ejecuciones. Como seguimos trabajando con el mismo clúster sólo será necesario editar *main.tf* y *variables.tf* con los recursos que queremos añadir, en este caso un Wordpress.

```
main.tf
1 resource "helm_release" "prometheus" {
2   name      = var.prometheus_helm_name
3   repository = var.prometheus_helm_repository
4   chart      = var.prometheus_helm_release
5   version    = var.prometheus_chart_version
6   namespace  = var.prometheus_namespace
7   create_namespace = true
8   timeout    = 2000
9 }
10
11 resource "helm_release" "wordpress" [
12   name      = var.wordpress_helm_name
13   repository = var.wordpress_helm_repository
14   chart      = var.wordpress_helm_release
15   version    = var.wordpress_chart_version
16   namespace  = var.wordpress_namespace
17   create_namespace = true
18   timeout    = 2000
19 ]
20
21 set {
22   name = "service.type"
23   value = "ClusterIP"
24 }
25
26
```

```
variables.tf
22 variable "wordpress_namespace" {
23   default = "wordpress"
24 }
25
26 variable "wordpress_helm_name" {
27   default      = "wordpress"
28 }
29
30 variable "wordpress_helm_release" {
31   default      = "wordpress"
32 }
33
34 variable "wordpress_helm_repository" {
35   default      = "https://charts.bitnami.com/bitnami"
36 }
37
38 variable "wordpress_chart_version" [
39   default = "22.2.2"
40 ]
```



4.3.3.5. Prueba de funcionamiento II

Editados los ficheros, procedemos a repetir los mismos pasos que el punto 4.3.3.3. Prueba de funcionamiento I. Se puede ver que Terraform ha obviado el Prometheus ya desplegado y ha saltado directo al despliegue de Wordpress.

```
helm_release.wordpress: Still creating... [2m0s elapsed]
helm_release.wordpress: Still creating... [2m10s elapsed]
helm_release.wordpress: Still creating... [2m20s elapsed]
helm_release.wordpress: Still creating... [2m30s elapsed]
helm_release.wordpress: Still creating... [2m40s elapsed]
helm_release.wordpress: Creation complete after 2m42s [id=wordpress]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

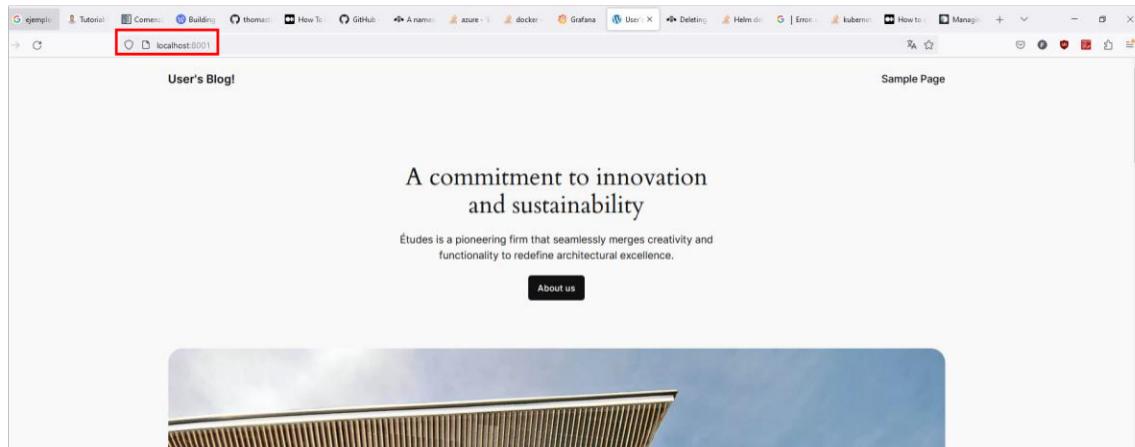
Si hacemos una comprobación sobre nuestro clúster, podemos ver que ya se encuentra desplegado y en estado READY.

```
PS C:\Users\MSI\terraform-tfg> kubectl get pods -n wordpress
NAME          READY   STATUS    RESTARTS   AGE
wordpress-mariadb-0   1/1     Running   0          10m
wordpress-8478c5795b-c4wc5   1/1     Running   0          10m
```

Ejecutamos de nuevo *kubectl port-forward* para hacer un túnel pero cambiado al puerto :8001 para no interferir con Grafana:

```
kubectl port-forward -n worpress svc/worpress 8001:80
```

Comprobamos que podemos acceder a Wordpress desde nuestro navegador.



Si hacemos de nuevo una consulta sobre nuestro clúster, podemos ver todos nuestros recursos desplegados y funcionando.



```
root@kubemaster1:/home/master1# kubectl get pods --all-namespaces -o wide
NAMESPACE     NAME                               READY   STATUS    RESTARTS   AGE      IP          NODE
kube-system   helm-install-traefik-crd-zrm8m   0/1    Completed  0          27h     <none>    kubeworker1
kube-system   coredns-7fd79bb5d-vd8sj          1/1    Running   2 (9h ago) 35d     10.42.1.84   kubeworker1
kube-system   local-path-provts-toner-84db5d44d9-m62wd 1/1    Running   68 (9h ago) 66d     10.42.0.253  kubemaster1
metallb-system speaker-nxtf6b                 1/1    Running   2 (9h ago) 35d     192.168.1.41  kubemaster1
metallb-system speaker-klrzp                  1/1    Running   6 (9h ago) 35d     192.168.1.40  kubemaster1
metallb-system speaker-tvwfj                  1/1    Running   1 (9h ago) 26h     192.168.1.42  kubeworker2
kube-system   metrics-server-67c658944b-p4wjs   1/1    Running   70 (9h ago) 66d     10.42.0.254  kubemaster1
metallb-system controller-5f56cd0f78-z5hxk     1/1    Running   6 (9h ago) 35d     10.42.0.252  kubemaster1
prometheus   prometheus-prometheus-node-exporter-cfbx7 1/1    Running   0         99m     192.168.1.42  kubeworker2
prometheus   prometheus-prometheus-node-exporter-55v74 1/1    Running   0         99m     192.168.1.41  kubeworker1
prometheus   prometheus-prometheus-node-exporter-2hvb 1/1    Running   0         99m     192.168.1.40  kubemaster1
prometheus   prometheus-kube-prometheus-operator-5f475fd846-wxkg5 1/1    Running   0         99m     10.42.0.10   kubemaster1
prometheus   prometheus-kube-state-metrics-59b5d50ff8f-wdxjq 1/1    Running   0         99m     10.42.2.100  kubeworker2
prometheus   alertmanager-prometheus-kube-prometheus-alertmanager-0 2/2    Running   0         99m     10.42.2.101  kubeworker2
prometheus   prometheus-grafana-5b4d6b9f84-b7zzz 3/3    Running   0         99m     10.42.1.95  kubeworker1
prometheus   prometheus-prometheus-kube-prometheus-prometheus-0 2/2    Running   0         99m     10.42.0.11   kubemaster1
wordpress   wordpress-mariadb-0                 1/1    Running   0         85m     10.42.2.105  kubeworker2
wordpress   wordpress-8478c5795b-c4wc5          1/1    Running   0         85m     10.42.1.98  kubemaster1
root@kubemaster1:/home/master1#
```

4.3.4. Uso de Terraform con AKS Azure

Aunque Terraform es compatible con Azure Kubernetes Service (AKS) y proporciona un conjunto de proveedores que permiten interactuar con los servicios de Azure, utilizaremos el mismo *provider* y solamente cambiaremos el archivo en *.kube/config*, ya que la API de Kubernetes es compatible, independientemente de encontrarse en local o en un servicio Cloud.

4.3.4.1. Creación de directorio y ficheros de Terraform

La ventaja que ofrece Terraform es que podemos reutilizar el código para implementar una estructura o un despliegue. En este caso será lo que vamos a realizar.

El fichero *.config_azure* contiene la información necesaria de conexión y datos de nuestro clúster.

```
config_azure: Bloc de notas
Archivo Edición Formato Ver Ayuda
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data: LS0tLS1CRUdjTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUU2VENDQXRhZ0F3SUJBZ01SUAvMWhMS1N1c2dZNXB4UmItS11RXPkaDBHbzxCZU5czl1MGVsNFM1dnZIAE1y0EFYOGRI2IRweHozU0dmT1VzMmlZWnx1SzA2UzNaCndoQ0ZaM0tXWGRcUszzYzcczPekYvRnEMm7VV2gr04ntTLdUYYXUlllJNXYT1NG8XRsTnE0hlpBnpSRVhN014M1E5U1RCAvQ301V3Mv7-MV0-CRwpvVHY2VHpDNC9vCjRFTWNHRU11Umoy
  server: https://aks-cluster-projecto-dns-h4tg5tg0.hcp.westeurope.azurek8s.io:443
  name: AKS-cluster-projecto
contexts:
- context:
  cluster: AKS-cluster-projecto
  user: clusterAdmin_AKS-grupo-recursos_AKS-cluster-projecto
  name: AKS-cluster-projecto-admin
current-context: AKS-cluster-projecto-admin
```

Vamos a reutilizar los ficheros *.tf* que creamos anteriormente para desplegar las mismas aplicaciones y lo único que tendremos que hacer es modificar el fichero *main.tf* indicando la ruta al *.config_azure* para que haga la conexión con el clúster de Azure.



4.3.4.2. Prueba de funcionamiento

Modificando el `providers.tf`, sólo tendríamos que repetir los comandos:

```
terraform init      terraform plan      terraform apply --auto-approve
```

En las siguientes capturas podemos comprobar que se han desplegado los dos recursos correctamente en nuestro clúster Azure.

The screenshot shows the Visual Studio Code interface with the Terraform extension open. The left sidebar shows the project structure with files like `main.tf`, `variables.tf`, `providers.tf`, and `wordpress-values.yaml`. The `providers.tf` file is selected and contains the following code:

```
provider "helm" {
  kubernetes {
    config_path = "C:/Users/MSI/.kube/config_azure"
  }
}
```

The right side shows the terminal output of the `terraform apply` command. It shows the creation of two resources: `helm_release.prometheus` and `helm_release.wordpress`. Both resources are shown as still creating for a long time (up to 3m39s). Finally, they both reach a completed state after approximately 3m39s. The terminal output ends with the message `Apply complete! Resources: 2 added, 0 changed, 0 destroyed.`

```
Plan: 2 to add, 0 to change, 0 to destroy.
helm_release.prometheus: Creating...
helm_release.prometheus: Still creating... [10s elapsed]
helm_release.wordpress: Creating...
helm_release.prometheus: Still creating... [20s elapsed]
helm_release.wordpress: Still creating... [10s elapsed]
helm_release.prometheus: Still creating... [30s elapsed]
helm_release.wordpress: Still creating... [20s elapsed]
helm_release.prometheus: Still creating... [40s elapsed]
helm_release.wordpress: Still creating... [30s elapsed]
helm_release.prometheus: Still creating... [50s elapsed]
helm_release.wordpress: Still creating... [40s elapsed]
helm_release.prometheus: Still creating... [1m0s elapsed]
helm_release.wordpress: Still creating... [50s elapsed]
helm_release.prometheus: Still creating... [1m10s elapsed]
helm_release.wordpress: Still creating... [1m0s elapsed]
helm_release.prometheus: Still creating... [1m20s elapsed]
helm_release.wordpress: Still creating... [1m10s elapsed]
helm_release.prometheus: Still creating... [1m30s elapsed]
helm_release.wordpress: Still creating... [1m20s elapsed]
helm_release.prometheus: Still creating... [1m40s elapsed]
helm_release.wordpress: Still creating... [1m30s elapsed]
helm_release.prometheus: Still creating... [2m0s elapsed]
helm_release.wordpress: Still creating... [1m50s elapsed]
helm_release.prometheus: Creation complete after 2m3s [id=helm_release.prometheus]
helm_release.wordpress: Still creating... [2m0s elapsed]
helm_release.wordpress: Still creating... [2m10s elapsed]
helm_release.prometheus: Still creating... [2m20s elapsed]
helm_release.wordpress: Still creating... [2m30s elapsed]
helm_release.prometheus: Still creating... [2m40s elapsed]
helm_release.wordpress: Still creating... [2m50s elapsed]
helm_release.prometheus: Still creating... [3m0s elapsed]
helm_release.wordpress: Still creating... [3m10s elapsed]
helm_release.prometheus: Still creating... [3m20s elapsed]
helm_release.wordpress: Still creating... [3m30s elapsed]
helm_release.wordpress: Creation complete after 3m39s [id=helm_release.wordpress]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

Si nos dirigimos al portal de Azure y ejecutamos una consulta en el Cloud CLI podemos ver que todo está levantado y funcionando.



READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
1/1	Running	0	67m	10.224.0.4	aks-agentpool-16462943-vms0000001	<none>	<none>
1/1	Running	0	10.224.0.113	aks-agentpool-16462943-vms0000000	<none>	<none>	<none>
1/1	Running	0	67m	10.224.0.4	aks-agentpool-16462943-vms0000001	<none>	<none>
1/1	Running	0	67m	10.224.0.113	aks-agentpool-16462943-vms0000000	<none>	<none>
1/1	Running	0	67m	10.224.0.35	aks-agentpool-16462943-vms0000001	<none>	<none>
1/1	Running	0	67m	10.224.0.102	aks-agentpool-16462943-vms0000001	<none>	<none>
1/1	Running	0	67m	10.224.0.105	aks-agentpool-16462943-vms0000001	<none>	<none>
3/3	Running	0	67m	10.224.0.4	aks-agentpool-16462943-vms0000001	<none>	<none>
3/3	Running	0	67m	10.224.0.113	aks-agentpool-16462943-vms0000000	<none>	<none>
3/3	Running	0	67m	10.224.0.4	aks-agentpool-16462943-vms0000001	<none>	<none>
3/3	Running	0	67m	10.224.0.113	aks-agentpool-16462943-vms0000000	<none>	<none>
1/1	Running	0	41m	10.224.0.109	aks-agentpool-16462943-vms0000001	<none>	<none>
1/1	Running	0	41m	10.224.0.132	aks-agentpool-16462943-vms0000000	<none>	<none>
1/1	Running	0	67m	10.224.0.113	aks-agentpool-16462943-vms0000000	<none>	<none>
1/1	Running	0	67m	10.224.0.4	aks-agentpool-16462943-vms0000001	<none>	<none>
2/2	Running	0	67m	10.224.0.47	aks-agentpool-16462943-vms0000001	<none>	<none>
2/2	Running	0	67m	10.224.0.84	aks-agentpool-16462943-vms0000001	<none>	<none>
2/2	Running	0	3m22s	10.224.0.173	aks-agentpool-16462943-vms0000000	<none>	<none>
3/3	Running	0	3m34s	10.224.0.108	aks-agentpool-16462943-vms0000001	<none>	<none>
1/1	Running	0	3m34s	10.224.0.25	aks-agentpool-16462943-vms0000001	<none>	<none>
1/1	Running	0	3m34s	10.224.0.7	aks-agentpool-16462943-vms0000001	<none>	<none>
2/2	Running	0	3m22s	10.224.0.179	aks-agentpool-16462943-vms0000000	<none>	<none>
1/1	Running	0	3m34s	10.224.0.4	aks-agentpool-16462943-vms0000001	<none>	<none>
1/1	Running	0	3m34s	10.224.0.113	aks-agentpool-16462943-vms0000000	<none>	<none>
1/1	Running	0	4m6s	10.224.0.170	aks-agentpool-16462943-vms0000000	<none>	<none>
1/1	Running	0	4m6s	10.224.0.191	aks-agentpool-16462943-vms0000000	<none>	<none>

5. CONCLUSIONES

Este proyecto ha sido mi primera toma de contacto con las tecnologías más extendidas en lo que se refiere a la gestión de infraestructuras IT.

He podido familiarizarme con la gestión de contenedores bajo Kubernetes – concretamente K3s – y utilizar herramientas de despliegue como Helm y Terraform, las cuales permiten la instalación y configuración de recursos IT como código (Infrastructure as Code).

He logrado crear y configurar clústeres de Kubernetes en diferentes entornos, desde un entorno local virtualizado hasta un entorno en la nube con Azure. He podido comprobar la capacidad de adaptación y escalabilidad de Kubernetes, así como la utilidad de herramientas como Helm o Terraform para simplificar la gestión de la infraestructura.

Con este proyecto he podido aplicar los conocimientos obtenidos a lo largo de este Ciclo Formativo de cara trabajar con tecnologías modernas de orquestación y automatización, siendo un punto de partida para continuar formándome en el ámbito de la computación en la nube y la gestión de contenedores.



6. REFERENCIAS BIBLIOGRÁFICAS

- **Contenedores**

<https://blogs.manageengine.com/espanol/2021/06/09/contenedores-nueva-era-oferta-infraestructura-ti.html>

<https://www.computerweekly.com/es/consejo/Cual-es-el-futuro-del-despliegue-de-contenedores>

<https://www.redhat.com/es/topics/automation/what-is-cloud-automation>

- **Kubernetes**

<https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/>

<https://ubuntu.com/kubernetes/install>

- **K3S**

<https://docs.k3s.io/>

<https://github.com/k3s-io/k3s/releases/tag/v1.29.2-rc1%2Bk3s1>

<https://es.linux-console.net/?p=20998>

- **Helm**

<https://helm.sh/docs/>

- **Artifacthub**

<https://artifacthub.io/>

- **Dev Community**

<https://dev.to/>

- **Grafana**

<https://grafana.com/docs/>

<https://grafana.com/docs/grafana/latest/dashboards/build-dashboards/import-dashboards/>

- **Deploying a Wordpress Website on a Kubernetes Cluster using Helm**

<https://www.digitalocean.com/community/developer-center/deploying-a-wordpress-website-on-a-kubernetes-cluster-using-helm>

- **charts/bitnami/wordpress/**

<https://github.com/bitnami/charts/tree/main/bitnami/wordpress/#installing-the-chart>

- **Consultas de errores durante el proyecto**



- <https://github.com/k3s-io/k3s/issues/1126>
- <https://github.com/rancher/k3os/issues/208>
- <https://github.com/helm/charts/issues/1282>
- <https://github.com/bitnami/charts/issues/8744>
- <https://stackoverflow.com/questions/51470059/kubernetes-service-showing-external-ip-pending-how-can-i-enable-it>
- <https://docs.k3s.io/networking>
- <https://kubernetes.io/docs/concepts/services-networking/service/#loadbalancer>
- <https://docs.bitnami.com/virtual-machine/apps/wordpress/troubleshooting/debug-errors/>
- **AKS Azure**
 - <https://learn.microsoft.com/es-es/azure/aks/learn/quick-kubernetes-deploy-portal?tabs=azure-cli#create-an-aks-cluster>
 - <https://k21academy.com/terraform-iac/create-manage-aks-cluster-using-terraform/>
 - <https://learn.microsoft.com/es-es/cli/azure/>
 - <https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/>
 - <https://learn.microsoft.com/es-es/azure/aks/control-kubeconfig-access#get-and-verify-the-configuration-information>
- **Helm for Windows**
 - <https://github.com/helm/helm/releases>
 - <https://phoenixnap.com/kb/install-helm>
- **Chocolatey**
 - <https://chocolatey.org/>
- **AKS LoadBalancer External-IP stuck on <pending>**
 - <https://stackoverflow.com/questions/75154385/aks-loadbalancer-external-ip-stuck-on-pending>
- **AKSCapacityError**
 - <https://learn.microsoft.com/es-es/troubleshoot/azure/azure-kubernetes/create-upgrade-delete/akscapacityerror>
- **Terraform**
 - https://developer.hashicorp.com/terraform?product_intent=terraform
 - <https://developer.hashicorp.com/terraform/tutorials>



<https://learn.microsoft.com/es-es/azure/developer/terraform/get-started-windows-bash?tabs=bash>
<https://terraform-infraestructura.readthedocs.io/es/latest/installacion/>
<https://terraform-infraestructura.readthedocs.io/es/latest/sintaxis/index.html>
<https://www.paradigmadigital.com/dev/terraform-la-navaja-suiza-dominar-todos-los-iaas/>
<https://blog.ichasco.com/terraform-desplegar-un-cluster-de-kubernetes-utilizando-modulos/>
<https://imagineinformacion.com/tutoriales/introduccion-a-terraform>
<https://www.terraform.io/docs/providers/index.html>
<https://stackoverflow.com/questions/67373856/unable-to-access-prometheus-dashboard-port-forwarding-doesnt-work>

7. AGRADECIMIENTOS

A Mar Millán,

Gracias por haberme animado a empezar este ciclo, por creer en mí, por apoyarme todo este tiempo, por tus ideas, por lo que me has enseñado, por tu paciencia conmigo, por no dejarme tirar la toalla cuando me ofusco. Sin ti, ni hubiera empezado ni hubiera acabado.

8. ANEXOS

8.1. Anexo – I: Enlace a repositorio en GitHub

En el siguiente enlace se puede consultar el repositorio del proyecto con los comandos utilizados en cada paso, ficheros y archivos readme.md



<https://github.com/svglmm?tab=projects>



8.2. Anexo – II: Diagrama de la infraestructura

