UCLM – E.S. de Informática

Ingeniería en Informática

GESTIÓN Y ADMINISTRACIÓN DE REDES

DESPLIEGUE Y ADMINISTRACIÓN DE UNA IT

BASADA EN CMS

Autores:

Felipe Segovia Friginal

Diego Dorado Galán

Andrés Castellanos Cantos

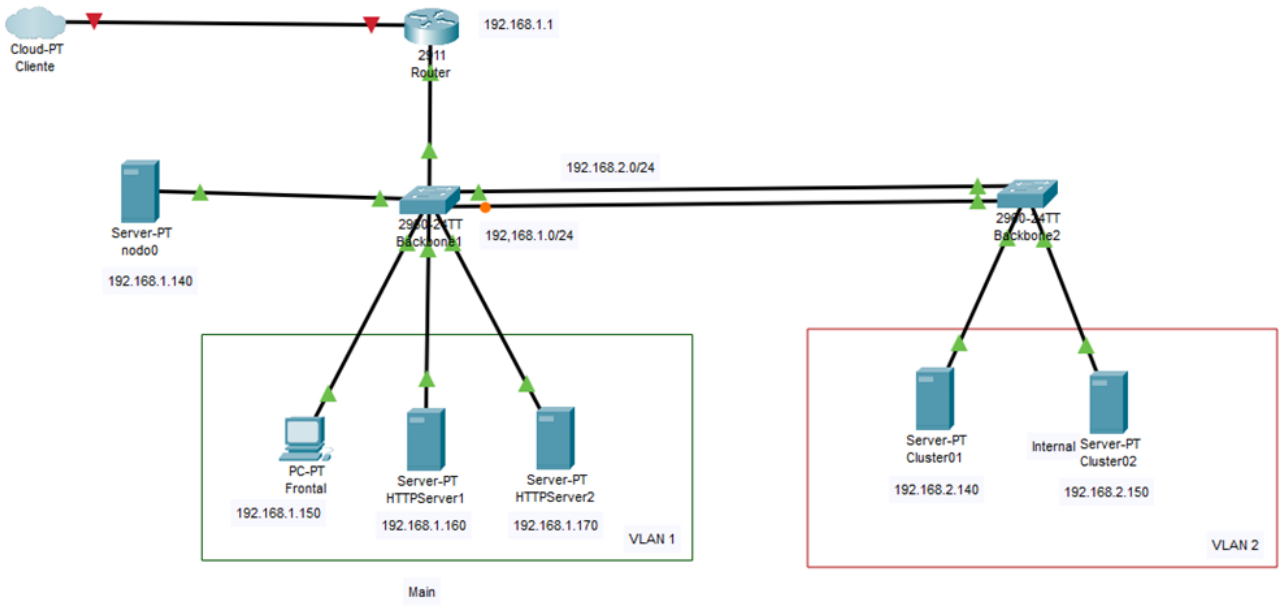
ÍNDICE

1. INTRODUCCIÓN

En el siguiente documento se tratará de explicar la resolución y elaboración de las prácticas de la asignatura Gestión y Administración de Red, así como cumplir con los requisitos especificados en la rúbrica de evaluación de la misma.

Se explicarán brevemente los servicios utilizados así como el procedimiento para desplegarlos en la infraestructura deseada, añadiendo algunos apuntes, posibles fallos o recomendaciones.

2. DIAGRAMA DE RED



3. TABLA CON VARIABLES Y VALORES POR DEFECTO

Oracle VM VirtualBox Versión 7.0.8 r156879 (Qt5.15.2)

NODO	IP	MAC	SISTEMA	ADAPTADOR RED	ALMAC
Nodo0	192.168.1.40 192.168.2.1	080027E156C4 080027C73F36	2048 MB RAM 2 CPU	1 - Ad.Puente 2 - Red Interna	25 GB
Frontera	192.168.1.150	080027BF6CCB	> 2048 MB RAM 1 CPU	1 - Ad.Puente	15 GB
Http1	192.168.1.160	08002762EF04	> 2048 MB RAM 1 CPU	1 - Ad.Puente	15 GB

Http2	192.168.1.170	080027CD7DAC	> 2048 MB RAM 1 CPU	1 - Ad.Puente	15 GB
Cluster 1	192.168.2.150	0800279734F1	> 4096 MB RAM 4 CPU	2 - Red Interna	15 GB
Cluster 2	192.168.2.160	080027B59C96	> 4096 MB RAM 4 CPU	2 - Red Interna	15 GB

Todos los nodos, excepto nodo0, tendrán ‘Orden de arranque por Red’ activado y prioritario.

Todos los nodos tendrán habilitado EFI y es opcional Secure Boot.

Los nodo con Adaptador Puente utilizan ‘Intel PRO/1000 MT Desktop (82540EM)’ como tipo de adaptador.

Se aconseja para evitar problemas en la configuración de ubuntu por red tener más de 3000 MB de Ram.

Imagen SSOO(Nodo0): ubuntu-20.04.6-desktop-amd64.iso

Imagen SSOO(Nodos remotos): ubuntu-22.04.2-live-server-amd64.iso

Todos los nodos tienen como usuario ‘ubuntu’ y password ‘server’ al igual que en el certificado ssh.

NODO	**SERVICIO**	**VERSIÓN**
Nodo0	isc-dhcp-server apache2 tftp-hpa ansible python	v4.4.1-2.1 v2.4.41 v0.17-22 v2.9.6 v3.8.10
Frontera	haproxy	v2.4.22
Http1	mediawiki apache2	v1.40.0 v2.4.52
Http2	mediawiki apache2	v1.40.0 v2.4.52
Cluster1	kubectl mysql mariadb-client	v1.27.3 v8.0.33 v1:10.6.12
Cluster2	kubectl mysql mariadb-client	v1.27.3 v8.0.33 v1:10.6.12

PARÁMETRO	VALOR POR DEFECTO	SIGNIFICADO	OBSERVACIONES
Clave SSH	none	Especificar la clave SSH para acceder a un nodo remoto	Intentará usar la clave por defecto almacenada en ~/.ssh/id_rsa.pub
Inventario	Inventory.ini	Ruta al archivo de inventario de los nodos a controlar	Es aquí dónde debemos indicar las direcciones de red con las que trabajar
playbook	-	Sirve para indicar el archivo que queremos ejecutar en el host remoto	ansible-playbook <archivo.yaml>
Clave root	False	Si necesitamos permisos de root para controlar un nodo remoto, Ansible nos pedirá introducir su clave	Se puede introducir como parámetro al ejecutar el comando mediante '-k' o 'K', o bien usar '-KK' para que nos solicite introducirla.
Variables adicionales	-	-	'—extra-vars' o '-e'
Archivo de configuración	ansible.cfg	Archivo dónde podemos introducir los parámetros anteriores si queremos modificar su valor por defecto y evitar así introducirlo en el propio comando	En nuestro caso hemos añadido la clave SSH a utilizar así como el inventario de hosts.

4. CONFIGURACIÓN NODO 0

Para configurar automáticamente la configuración del servicio de PXE en nuestro repositorio de Github tenemos en la carpeta ‘pxe’ todos los archivos necesarios para la configuración, podemos clonar el repositorio y en la carpeta ‘pxe/bin’ tenemos un script llamado ‘setup’ que nos hace automáticamente la instalación de los servicios isc-dhcp-server, tftp-hpa y apache2, y la autoconfiguración de estos.

4.1 DHCP

```
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.149 192.168.1.200;
    option routers 192.168.1.1;
    option subnet-mask 255.255.255.0;
    option domain-name-servers 8.8.8.8, 172.20.32.3, 172.20.32.4;
}

subnet 192.168.2.0 netmask 255.255.255.0 {
    range 192.168.2.149 192.168.2.200;
    option routers 192.168.2.1;
    option subnet-mask 255.255.255.0;
    option domain-name-servers 8.8.8.8, 172.20.32.3, 172.20.32.4;
}
```

En ‘/etc/dhcp/dhcpd.conf’ donde configuramos las dos subredes con un rango de direcciones que daremos a las máquinas, la puerta de enlace que será nuestro option router, en el caso de la subred 192.168.2.0 como es internal la puerta de enlace será el propio nodo0, y por último las direcciones de dns.

```
host serverHTTP2 {
    hardware ethernet 08:00:27:CD:7D:AC;
    fixed-address 192.168.1.170;
    next-server 192.168.1.140;
    filename "efi/shimx64.efi";
}

host cluster1 {
    hardware ethernet 08:00:27:97:34:F1;
    fixed-address 192.168.2.150;
    next-server 192.168.1.140;
    filename "efi/shimx64.efi";
}
```

Después configuramos las diferentes máquinas según su mac en la que configuraremos si ip (siempre dentro del rango de ip según la subred), el next-server que será la dirección del server tftp y filename que es el archivo de configuración efi.

4.2 TFTP

El archivo de configuración ‘cloud-init’ es un archivo de configuración de ubuntu básico en el que lo más importante es:

```
realname: ubuntu
username: ubuntu
password: $6$eMKpPA05XXI.Q0mM$sjyi3P9RmYqnQI4eGcAfEjlqmk2A6x3o/UzlwTjO.cmSfGSRYV5focuMfxlOcgYpmVn1Qkx\
```

Creamos el usuario ubuntu y con la contraseña cifrada que es ‘server’.

```
network:
  version: 2
  ethernets:
    enp0s3:
      critical: true
      dhcp-identifier: mac
      dhcp4: true
```

Creamos la configuración de red creando una interfaz enp0s3 que estará configurada según nuestro servidor dhcp identificado por la mac de la máquina.

```
ssh:
  allow-pw: true
  authorized-keys: [ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIP67Q8PurVNGTS9nc287Z/HK5w7jgaDXj
  install-server: true
```

Instalamos también el servidor ssh y agregamos la clave ssh del nodo0 que utilizaremos más adelante para la comunicación ssh entre el nodo0 y la máquina remota con Ansible.

4.3 GRUD, ARRANQUE UEFI

```
menuentry "Boot from Local Disk" {
    insmod chain
    search --set=root --file /EFI/ubuntu/grubx64.efi
    chainloader /EFI/ubuntu/grubx64.efi
}

menuentry "Install Ubuntu 22.04 Server - Standard Configuration" {
    linux /ubuntu-22.04/vmlinuz root=/dev/ram0 ip=dhcp url=http://${pxe_default_server}/ubuntu-22.04
    initrd /ubuntu-22.04/initrd
}

menuentry "Install Ubuntu 22.04 Server - Manual Installation" {
    linux /ubuntu-22.04/vmlinuz root=/dev/ram0 ip=dhcp url=http://${pxe_default_server}/ubuntu-22.04
    initrd /ubuntu-22.04/initrd
}
```

Configuramos un archivo grub que será el encargado de configurar nuestra interfaz de arranque en modo seguro uefi. El menú Standard Configuration nos proporciona una

instalación de ubuntu automático teniendo como configuración nuestra archivo cloud-config y la .iso que tenemos en nuestro servidor apache.
En cambio el menú Manual Installation realizará una instalación de ubuntu manual.

4.4 ANSIBLE

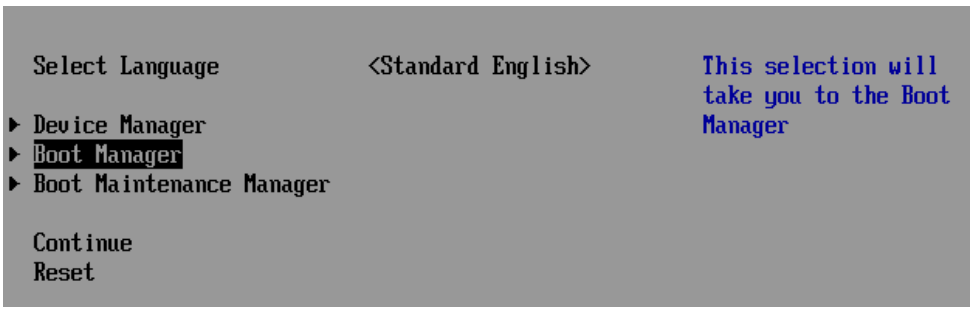
Ansible es un software de automatización del aprovisionamiento de software, gestión de configuración y despliegue de aplicaciones. Básicamente, es una herramienta de orquestación que permite gestionar servidores, configuraciones y aplicaciones de forma sencilla, robusta y paralela.

Su elemento principal es el playbook, conjunto de tareas organizadas con un fin y que se ejecutan de forma secuencial. A partir de un fichero de inventario, dónde indicamos los nodos y grupos de nodos, determinamos un estado deseado para esas máquinas, mediante las tareas definidas previamente en el playbook.

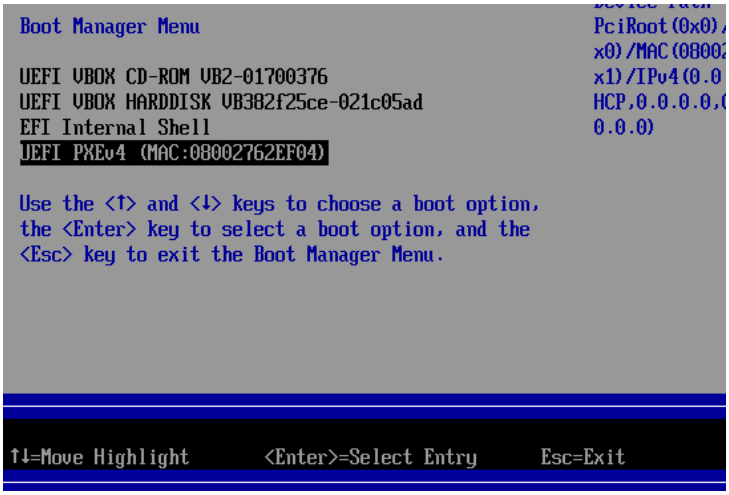
5. DESPLIEGUE

5.1 ARRANQUE POR RED. PXE

Una vez que tenemos la máquina creada y configurada su mac, uefi habilitado y arranque por red, podemos iniciar la máquina.
Nos aparecerá el siguiente mensaje indicándonos que no ha encontrado ningun CD-ROM y debemos pulsar cualquier tecla para entrar en el Boot Manager Menú.



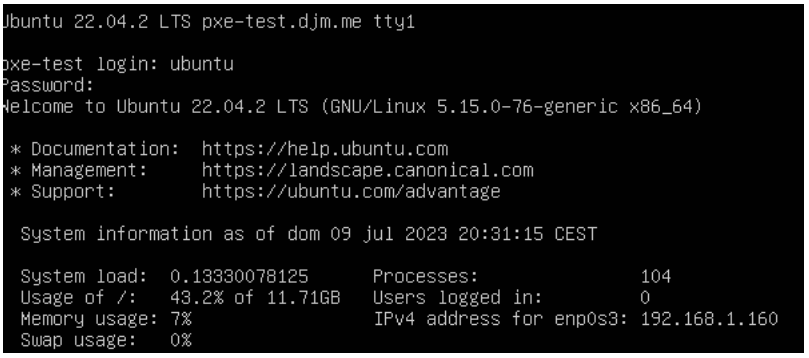
Una vez dentro del Boot Manager Menú entramos en la opción Boot Manager.



Nos aparecerá el arranque UEFI PXEv4 con nuestra MAC donde entraremos. Después dejaremos que carle el servidor dhcp y nos proporcione una IP y nos aparecerá el siguiente menú.

Menú Grub donde entraremos dentro de la opción Standard Configuration que nosubu realizará la instalación de Ubuntu 22.04 Server automáticamente con la configuración de nuestro cloud-init.

Una vez terminada la configuración del Cloud-Init, todos los nodos tienen como login ‘Ubuntu’ y Password ‘server’.



5.2 AUTOMATIZACIÓN DE CONFIGURACIÓN DE SISTEMAS. ANSIBLE

Ansible nos permite automatizar la instalación y puesta en marcha del software y configuración de los nodos haciendo uso del protocolo SSH en el puerto 22. Para ello, debemos copiar la clave creada en el nodo de control en el resto de nodos remotos.

Tal y como se ha mencionado anteriormente, se ha creado también un archivo de configuración de Ansible para predefinir algunas variables como el archivo de inventario, clave ssh a utilizar, etc.

El comando básico que se ejecutará será:

ansible-playbook main.yaml -KK

A partir de aquí trabajamos con funcionalidades que nos ofrece Ansible como es el uso de roles para distinguir, por ejemplo, entre nodos maestros y workers en la red Internal, que ejercen como control-plane y data-plane del clúster de Kubernetes.

Para que el despliegue sea lo más automático posible, incorporamos toda la lógica de Ansible sobre el archivo “main.yaml”, dónde se indica la secuencia para la configuración de los sistemas de todos los nodos en función de su rol: balanceador, servidor HTTP y nodos del clúster de Kubernetes.

DESPLIEGUE DE UN CLÚSTER DE KUBERNETES

Para el despliegue del clúster de Kubernetes sobre el que instanciar la BBDD, hemos considerado que es mejor práctica desplegar un clúster que no sea HA. Esto es debido a que hacemos uso únicamente de dos nodos en la red Internal, por lo que para desplegar un clúster de alta disponibilidad necesitaríamos ocupar ambos nodos para el control-plane y no tener data-plane, lo que no sería recomendable para instanciar en él una base de datos (aunque sería posible haciendo uso de taints, aunque sería limitar el uso de pods del control-plane para usarlos en la base de datos). Por ello, preferimos un clúster funcional dónde poder trabajar a un clúster de alta disponibilidad.

Para el despliegue haremos uso de dos roles: un rol master y un rol worker, dónde almacenamos sus respectivas lógicas de ejecución en el archivo tasks/main.yaml, y otros ficheros necesarios.

En primer lugar, debemos instalar las herramientas necesarias en los respectivos nodos, como son Docker.io, iproute2, apt-transport-https o curl, con las que necesitaremos trabajar. Añadimos la clave APT de Kubernetes, instalamos Kubelet, Kubeadm y Kubectl (los tres componentes deben coincidir en versión, en nuestro caso hemos usado 1.27.3).
Debemos hacer un swapoff -a e iniciar los servicios de Docker y del Kubelet para después, desde el nodo master hacer:

kubeadm init --pod-network-cidr={{ pod_cidr_network }} --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem

Este comando se encarga de iniciar el clúster de Kubernetes. Cuando se realiza, debemos crear un directorio en el nodo remoto llamado .kube, que albergará el archivo config (contexto). Debemos copiarlo a este directorio ya que por defecto se almacena en /etc/kubernetes/admin.conf.
Se debe instalar también el plugin de red que sirve para que los pods y nodos se comuniquen entre ellos. Aquí tenemos muchas opciones con las que trabajar, como Calico, Cri-O o Flannel, el que hemos decidido utilizar. La ejecución de kubeadm init devuelve un token que debemos almacenar para poder añadir nodos a dicho clúster.

Desde el nodo worker, debemos realizar toda la instalación del software necesario y posteriormente utilizar el token almacenado previamente para hacer:

kubeadm join <control-plane-host>:<control-plane-port> --token <token> --discovery-token-ca-cert-hash <hash>

De tal forma, ya quedaría configurado el clúster de Kubernetes.

```
kdt23@kdt23:~$ sudo kubectl cluster-info
Kubernetes control plane is running at https://192.168.1.157:6443
CoreDNS is running at https://192.168.1.157:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

```
kdt23@kdt23:~$ sudo kubectl get nodes
NAME                STATUS   ROLES           AGE     VERSION
kubernetes-master   Ready    control-plane   4m10s   v1.27.3
```

DESPLIEGUE DE LA BASE DE DATOS EN EL CLÚSTER DE KUBERNETES

Para el instanciar la base de datos en el clúster de Kubernetes seguiremos el procedimiento que se documenta en el siguiente repositorio:
https://github.com/geerlingguy/mariadb-operator

Haremos uso de un operador de MariaDB para simplificar la instalación.

6. REGLAS IPTABLES

En el nodo0 para poder instalar Kubernetes y MariaDB en los nodos internos que no tienen conexión a internet debemos aplicar las siguientes reglas para redireccionar las peticiones de la red internal a la red main, teniendo así acceso a internet, y despues de la ejecución del playbook mediante Ansible, eliminar estas reglas.

```
sudo sysctl net.ipv4.ip_forward=1
sudo iptables -A FORWARD -i enpos8 -o enpos3 -j ACCEPT
sudo iptables -A FORWARD -i enpos3 -o enpos8 -m state --state RELATED,ESTABLISHED -j ACCEPT
sudo iptables -t nat -A POSTROUTING -o enpos3 -j MASQUERADE
```

En el nodo Frontera que es el único que tiene acceso a internet y que debe reenviar peticiones a los nodos http, además de permitir acceso ssh del nodo0:

```
// Permitimos trafico saliente y de reenvio
sudo iptables -P OUTPUT ACCEPT
sudo iptables -P FORWARD ACCEPT

// Permitimos trafico entrante por el puerto ssh proveniente del nodoo
sudo iptables -A INPUT -s 192.168.1.140 -p tcp --dport 22 -j ACCEPT

//Aceptamos trafico entrante al puerto 443(https) y 80(http) dentro de la red main
sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT

//Permitir el tráfico entrante relacionado y establecido:
sudo iptables -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT

// Denegamos trafico entrante
sudo iptables -P INPUT DROP
```

En los nodos http deben permitir conexión ssh del nodo0 y peticiones http y https del nodo Frontera, lo demás lo denegamos:

```
// Permitimos trafico saliente  pero no de reenvio
sudo iptables -P OUTPUT ACCEPT
sudo iptables -P FORWARD DROP

// Permitimos trafico entrante por el puerto ssh proveniente del nodoo
sudo iptables -A INPUT -s 192.168.1.140 -p tcp --dport 22 -j ACCEPT

//Aceptamos trafico entrante al puerto 443(https) y 80(http) proveniente del balanceador
sudo iptables -A INPUT -p tcp -s 192.168.1.150 --dport 80 -j ACCEPT
sudo iptables -A INPUT -p tcp -s 192.168.1.150 --dport 443 -j ACCEPT

// Denegamos peticiones entrantes
sudo iptables -P INPUT DROP
```

7. HAPROXY Y SERVICIO MEDIAWIKI

Para configurar las máquinas http, ejecutaremos el playbook main.yml en el que los nodos con role http instalaremos el servidor apache y los módulos php, además de copiar un .tar con los servicios de mediawiki, en los nodos remotos para después descomprimirlo. También con la ejecución del playbook configuraremos las iptables mencionadas anteriormente. Y ya tendríamos funcionando el servicio mediawiki.

Para el nodo Frontera que se encargará de balancear las peticiones http y https a los nodos http vamos a instalar haproxy también con la ejecución del playbook main.yml. En el que instalaremos haproxy y copiaremos un archivo de configuración para el control y reenvío de las peticiones http. Reiniciamos el servicio y copiamos las reglas iptables del nodo Frontera y estaría listo el servicio haproxy.

Para comprobarlo abrimos una máquina fuera de la red main e internal y mandamos la siguiente petición al nodo Frontera:

```
curl http://192.168.1.150/mediawiki-1.40.0
```

Obteniendo como respuesta la siguiente ya que no tenemos conexión con la base de datos:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://192.168.1.150/mediawiki-1.40.0/">here</a>.</p>
<hr>
<address>Apache/2.4.52 (Ubuntu) Server at 192.168.1.150 Port 80</address>
</body></html>
```

En cambio si lanzamos la petición directamente a los nodos http no obtendremos respuesta:

```
nodo0@nodo0-VirtualBox:~$ curl http://192.168.1.170/mediawiki-1.40.0
```

Para comprobar el funcionamiento del haproxy podemos ejecutar el siguiente comando en el nodo Frontera y comprobar que reenvía cada vez a una las peticiones que le llegan:

```
sudo tail -f /var/log/haproxy.log
```

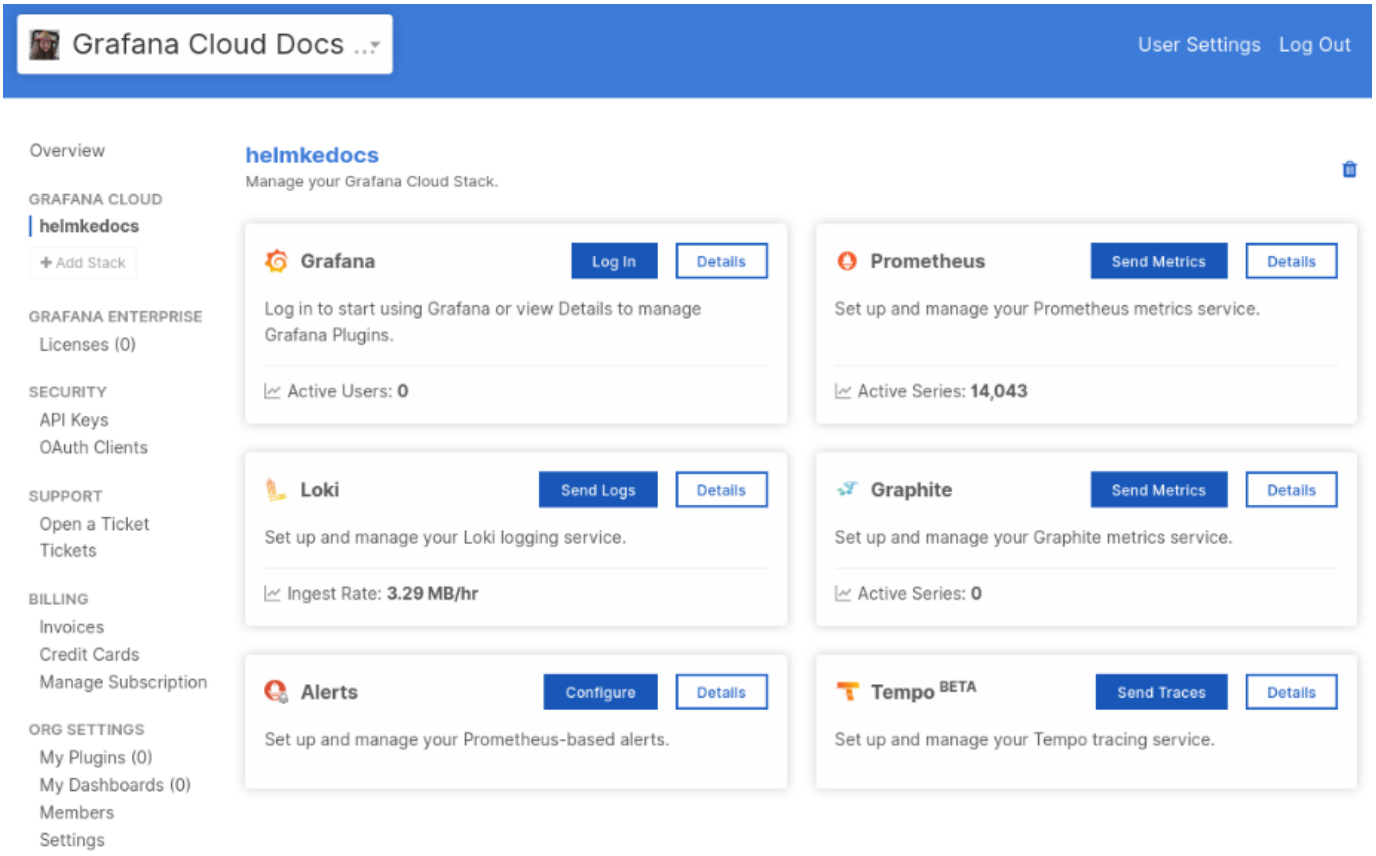
8. MONITORIZACIÓN

Este punto era del que se encargaba un compañero que finalmente ha decidido no participar, por lo que no nos ha dado tiempo a desarrollarlo. Sin embargo, no queremos dejar sin explicar la parte de monitoreo ya que consideramos que es fundamental para un funcionamiento correcto y detección de errores de cualquier sistema.
Debemos hacer uso de Prometheus y Grafana para monitorizar todos los nodos del sistema. Son herramientas que se utilizan generalmente en clústeres de Kubernetes, pero que podemos utilizar en nuestro proyecto para manejar métricas de los distintos hosts.
En primer lugar debemos instalar en los nodos remotos Node_exporter (escucha en HTTP: 9100). Esto puede hacerse mediante, por ejemplo el comando:

wget https://github.com/prometheus/node_exporter/releases/download/v/node_exporter-*.*-amd64.tar.gz*

Extraemos el binario, que es el que se encargará de recopilar las métricas de las máquinas y formatearlas de forma que sean legibles por Prometheus, proporcionando estas a un puerto interno para que Prometheus pueda extraerlas. Nos movemos al directorio creado tras descomprimir el archivo anterior y damos permisos de ejecución al binario para su posterior ejecución. Las métricas que expone podemos verlas si hacemos un curl http://localhost:9100/metric en cada nodo. Es importante recordar que estos pasos se tienen que ejecutar en todos los nodos, por lo que es fundamental el uso de Ansible.

Después de esto, debemos instalar Prometheus en el nodo 0. Si descomprimimos y accedemos al directorio descargado, podremos modificar el archivo de configuración de Prometheus para configurar el scraping de los endpoints de los nodos, y finalmente arrancar el servicio de Prometheus.
Debemos tener una cuenta de Grafana Cloud, ya que a este servidor es al que Prometheus transmitirá la información. Esta herramienta es ampliamente configurable y permite definir políticas de escrituras, de scraping, etc.



(https://grafana.com/docs/grafana-cloud/quickstart/noagent_linuxnode/)

Después de esto, tenemos que ejecutar el binario de Prometheus indicando el archivo de configuración que queremos utilizar usando el parámetro “--config.file”. Un ejemplo de configfile de Prometheus sería el siguiente:

```
global:
  scrape_interval: 60s

scrape_configs:
  - job_name: node
    static_configs:
      - targets: ['localhost:9100']

remote_write:
  - url: '<Your Metrics instance remote_write endpoint>'
    basic_auth:
```

```
username: 'your grafana username'
password: 'your Grafana API key'
```

Si entramos a Grafana Cloud veremos cómo se están tratando las métricas.
Ahora debemos configurar un dashboard para visualizar las métricas. Tenemos la opción de instalar una plantilla prediseñada o crearla nosotros mismos.

9. AÑADIR NODOS

SUSTITUCIÓN / AMPLIACIÓN DE NODOS CMS

Para actualizar los nodos pertenecientes a CMS debemos tener en cuenta varios factores.
En primer lugar, el nodo debe estar operativo, es decir, debemos haber provisionado previamente su sistema operativo y configuración de red mediante PXE, al igual que hicimos para crear los nodos originales. Una vez el nodo esté operativo, la lógica de ampliación y sustitución se realiza sobre Ansible.
Ansible trabaja con un nodo central que controla los nodos que figuran en el archivo de inventario. Es decir, la forma para ampliar los nodos sería únicamente añadir un nodo al grupo o host_group que tenemos asociado para el rol de CMS.
Ansible se basa en el uso de SSH para el control de los nodos remotos, por lo que es necesario crear una clave SSH en el nodo maestro y pasarla al nodo remoto para su control. Aquí tenemos varias opciones para ello. Por ejemplo, podemos pasarle la clave SSH al nodo en el momento de su aprovisionamiento mediante PXE, hacerlo manualmente en la terminal (haciendo uso de ssh-copy-id) o bien hacer uso de un script que nos facilite el trabajo.
Para este caso concreto, nos ha parecido buena idea desarrollar un playbook que automatice todo el proceso de adición de nodos al inventario y configuración de SSH.
Hemos desarrollado un playbook y un script que hacen uso de un archivo “nodos.yaml” dónde únicamente debemos indicar su dirección IP, dirección MAC y su rol. Toda la documentación relativa la encontramos en el repositorio de Github (Ansible/add_nodos).
Brevemente, para evitar ser repetitivos, hacemos uso de un playbook (add_nodes.yaml) que comprueba si los nodos que figuran en el archivo de “nodos.txt” ya están en el inventario o se deben añadir. Este playbook también hace uso de un script (sh_ssh_nodos.sh) para pasar la clave ssh a utilizar a todos los nodos, con el fin de conseguir un grado de automatización que sería imposible si se hiciese manualmente.
La sustitución de nodos seguiría un proceso similar a la adición de estos. En primer lugar, debemos aprovisionar y configurar el nodo para, después, modificar el archivo de inventario. Es recomendable realizar una transición gradual del tráfico o carga de trabajo de estos nodos, es decir, reducir paulatinamente el trabajo del nodo a sustituir e ir añadiendo carga de trabajo al nodo sustituto para así evitar tiempos de espera o inactividad. Una vez que el nodo sustituto ocupe la carga total de tráfico o trabajo, podremos eliminar el nodo original del archivo de inventario.
De forma resumida, para añadir nuevos nodos CMS lo que se debe hacer es modificar el archivo de inventario de Ansible, en concreto añadiendo al host group correspondiente.
Toda la documentación relativa al código de ejemplo se encuentra en el repositorio.

AMPLIAR LOS NODOS DE LA BASE DE DATOS

Este caso es algo distinto pero la lógica es similar. La base de datos se instancia sobre el clúster, por lo que si queremos ampliar los nodos de la base de datos tenemos dos opciones: añadir un nodo como nodo maestro y que ejecute el control-plane (convirtiéndose en un clúster de alta disponibilidad una vez se tienen 3 nodos en el control-plane(HA)), o añadirlo como nodo "worker" y que ejecute el data-plane (o node-plane). Para ello, debemos añadir el nodo que queremos implementar al host-group worker o master.
En nuestro playbook, únicamente hemos añadido la opción de añadirlo al data-plane, cómo worker, ya que el procedimiento será igual que en el despliegue original. En cambio, si queremos añadirlo como master-node, no tenemos controlado si ya existe el clúster y debe unirse a este o debe crear uno nuevo, por lo que siempre intentará desplegar un nuevo clúster, lo que desencadena en un fallo ya que creará otro control-plane independiente.

Si queremos crear un clúster de alta disponibilidad utilizando distintos nodos para el control-plane debemos desplegar un clúster etcd. Después, el primer nodo master debe hacer un kubeadm init especificando los detalles del clúster etcd. Esto es básicamente en lo que se diferencia de nuestro diseño.
Para la adición de nodos al control-plane, el nodo en cuestión debería ejecutar un comando similar al siguiente:
``*kubeadm join --control-plane <control-plane-host>:<control-plane-port> --token <token> --discovery-token-ca-cert-hash <hash>*``

Y para añadir el nodo al data-plane (nodo worker):
``*kubeadm join <control-plane-host>:<control-plane-port> --token <token> --discovery-token-ca-cert-hash <hash>*``

En el segundo caso, es el mismo comando que para vincular el nodo worker original con el nodo master, por lo que para el resto de nodos worker a añadir el proceso sería exactamente igual y valdría con añadirlo al host-group "worker" en el archivo de inventario.

Ahora bien, desde el punto de vista lógico, si queremos redundar y garantizar disponibilidad en la base de datos, no es suficiente con esto, y de hecho, no es necesario. Para ello, es más práctico (y lo habitual) hacer uso de las ventajas que nos ofrece un orquestador de contenedores como es Kubernetes.
La opción más básica es instanciar la base de datos en un pod (unidad lógica mínima en Kubernetes), algo que no es recomendable ya que los pods son efímeros. Lo habitual es hacer uso de **Deployments** ya que garantizan la vida del contenedor hasta que deseemos destruirlo. Esto se debe gracias al componente que se despliega al hacer uso de un deployment llamado **ReplicaSet** (puede desplegarse también de forma independiente). Este componente se encarga de garantizar que el número de pods coincide con el especificado en el manifiesto YAML que define el propio Deployment (es decir, garantiza que el estado deseado coincide con el estado actual). Un ejemplo básico de deployment podría ser el siguiente:

El parámetro replicas del spec es dónde se define el número de réplicas que se debe tener en todo momento.
Sin embargo, el hecho de replicar una base de datos no es una buena idea, ya que el hecho de sincronizar las escrituras en distintas réplicas es, cuanto menos peligroso.
Por ello, es una opción hacer uso de **StatefulSet**.

También debemos incluir el manifiesto del **Servicio**, que se encargará de exponer la base de datos para que sea accesible desde los nodos de la red Main (nodos CMS). Este servicio debe ser de tipo NodePort ya que no tenemos una infraestructura cloud por detrás. Si fuese así, podríamos desplegarlo como LoadBalancer y utilizar la IP de este para trabajar, pero no aplica para este trabajo.

10. REFERENCIAS

https://www.palomargc.com/posts/Cluster-de-Kubernetes-con-Ansible/

https://blog.kubesimplify.com/kubernetes-126

https://github.com/geerlingguy/mariadb-operator/tree/master

https://kubernetes.io/docs/home/

d13r/pxe: PXE boot to install Ubuntu over the network (github.com)

Automated 20.04 Server Installation using PXE and live server image - Ask Ubuntu

Automated Server installer config file reference | Ubuntu

https://grafana.com/docs/grafana-cloud/quickstart/noagent_linuxnode/

https://developer.couchbase.com/tutorial-node-exporter-setup