Asignatura	Datos del alumno	Fecha
Experto Universitario en DevOps & Cloud	Apellidos: Arenas Morante	14/07/2022
	Nombre: Iván	

Propuesta de solución

Caso práctico 2

URL de repositorio solución de GitHub: https://github.com/aremox/caso_pracico_2

Asignatura	Datos del alumno	Fecha
Experto Universitario en DevOps & Cloud	Apellidos: Arenas Morante	14/07/2022
	Nombre: Iván	

Caso práctico 2. Arquitectura

Diagramas mostrando los elementos desplegados y su rol. En el caso del PaaS además se deberán indicar todos sus componentes (workers, masters …). Estos diagramas deberán ser realizados por el alumno y no puede reutilizar los utilizados en clase, disponibles en internet…

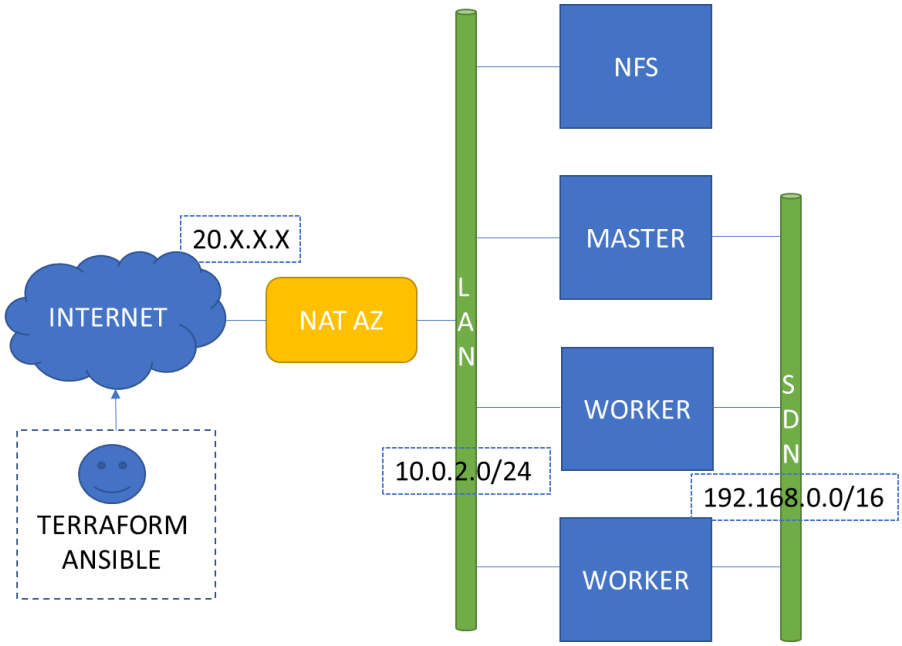


Ilustración 1Infraestructura montada

Caso práctico 2. Descripción del entorno

Descripción del entorno de PaaS desplegado. Numero de instancias, rol de cada una, características de cada una (memoria, ram …).

Nuestra infraestructura se compone de cuatro nodos. El primero es un NFS donde se compartirá un filesystem para almacenar el contenido de los POD de K8S. El segundo nodo es un master de K8S para gestionarlo.

El tercero y cuarto son los worker de K8S donde se desplegará los POD de la aplicación.

Disponemos de tres direccionamientos diferentes:

- Direccionamiento privado: 10.0.2.0/24 direccionamiento privado de las maquinas
- Direccionamiento público: 20.X.X.X direccionamiento público de Azure. Son las ips que nos asigna azure para poder llegar a nuestras maquinas desde el exterior y por SSH. Funciona como un nateo de IP entre las privadas y las publicas

Direccionamiento K8S: 192.168.0.0/16 Es el direccionamiento SDN (red definida por software) controlada por CALICO.

Por otro lado, tenemos el terraform y ansible instalado en un PC para administrar la infraestructura y no tener guardado en esta los ficheros de configuración

Se la idea de usar dos worker es para poder probar la réplica de los POD y su escalado en diferentes maquinas compartiendo almacenamiento por NFS

DIMENSIONAMIENTO

Nodo	Tipo	CPU	RAM	DISCOS
NFS	Standard_B1ls	1	512 mb	4 + 10
Master	Standard_A2_v2	2	4 Gb	20
Worker	Standard_B1ms	1	2 Gb	4
Worker	Standard_B1ms	1	2 Gb	4

El disco adicional solo se añade a la máquina del NFS, al resto no es necesario. Si lo necesitásemos seria tan sencillo como modificar el bucle de creación de discos y poner la variable de numero de máquinas.

Caso práctico 2. Proceso de despliegue

Deberás describir todo el proceso de despliegue, tanto de los elementos de infraestructura (instancia CentOS, PaaS y la aplicación). Se hará referencia al código en el repositorio para explicar el despliegue. Esta descripción deberá ser a alto nivel y los detalles técnicos deberán estar en los comentarios del código. No se debe

explicar cómo funciona el código (esa parte tendrá que estar presente dentro del código con comentarios).

1. Laboratorio local:

Para la validación de los scripts y no consumir recursos de azure se monto un laboratorio en virtual box desplegado con vagrant.

En laboratorio, compuesto por cuatro maquinas, se decidió que una de estas actuara de terminal que luego nos ha servido para ejecutar los scripts contra azure. Esto era necesario para facilitar la ejecución de la practica ya que nos encontrábamos sobre un entorno Windows.

Tambien cabe destacar la complicación que nos encontramos a la hora de ejecurar los scripts contra maquinas creadas por vagrant ya que este crea una tarjeta eth0 como NAT complicando la comunicación entre las maquinas

Lo resolvimos ejecutando desde ansible la desactivación de la tarjeta

```
- name: Quitar interface 0
  shell: echo "DEVICE=eth0" > /etc/sysconfig/network-scripts/ifcfg-eth0
&& service network restart
```

2. Instalacion de infraestructura en azure:

Lo primero que hemos hecho para crear la infraestructura con terraform, ha sido indicar en main.tf el provider que necesitábamos para comunicarnos con azure.

```
    azurerm = {
        source = "hashicorp/azurerm"
        version = "~> 3.10.0"
    }
```

Posteriormente, hemos definiendo en el fichero correccion-vars.tf las variables necesarias o que podíamos cambiar como son el número de máquinas, el tamaño de los discos adicionales o los modelos de máquinas a utilizar

```
variable "num_maquinas" {
  default = 4
}
```

```
variable "availability-size" {
  type = list(string)
  default = [
    "Standard_B1ls",
    "Standard_A2_v2",
    "Standard_B1ms",
    "Standard_B1ms"
  ]
}
variable "disco" {
  default = 10
}
```

En el fichero recursos.tf hemos ido definiendo los recursos de azure y asociándolos a las máquinas.

Por último, se creó el fichero outputs.tf que usamos para definir las salidas de la ejecución. En este fichero también hemos utilizado la función templatefile para crear los ficheros de inventario y de variables globales de ansible y así automatizar el proceso completo de instalación.

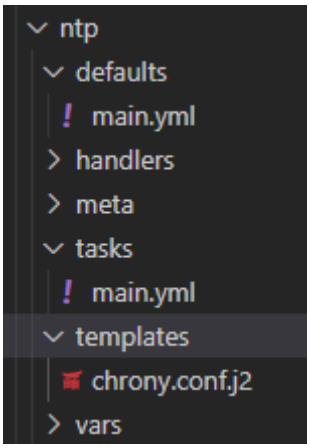
```
resource "local_file" "ansible_inventory" {
  content = templatefile("inventory.tmpl",
    {
     pubip = "${azurerm_public_ip.publicip.*.ip_address}"
     priip = "${azurerm_network_interface.nic.*.private_ip_address}"
     usuario = "${ var.usuario }"
     path_rsa = "${var.path_rsa}"
    }
  )
  filename = "/etc/ansible/inventory"
}

resource "local_file" "ansible_variables" {
  content = templatefile("var_main.yml.tmpl",
    {
     pubip = "${azurerm_public_ip.publicip.*.ip_address}"
     priip = "${azurerm_network_interface.nic.*.private_ip_address}"
     nombre = "${azurerm_linux_virtual_machine.vm.*.name}"
     usuario = "${ var.usuario }"
     path_rsa = "${var.path_rsa}"
```

```
    }
  )
  filename = "/etc/ansible/playbook/roles/vars/main.yml"
}
```

3. Prerrequisitos de instalación:

Ya entrando en el área de ansible, lo primero que hemos creado es el rol de ntp (chrony) para tener todas las maquinas sincronizadas en hora. Para esto el script de ansible instala, pone la zona horaria, copia el fichero de configuración con las direcciones del ROA (https://www2.roa.es/hora/) y forzamos la sincronización.

Los servidores de tiempo los hemos definido en el fichero main del directorio default y la estructura del fichero está en templates



4. Instalación de NFS:

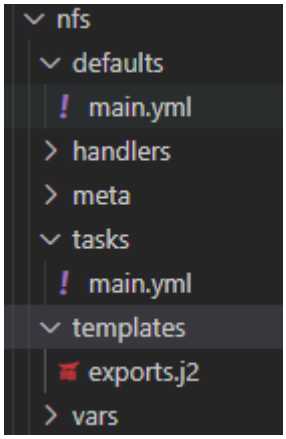
Lo primero que hemos acometido, por su sencillez, ha sido montar el servidor de NFS y los clientes de este. Esto ha supuesto la creación de dos roles, uno para el servidor, lanzado en hosts identificado como NFS y otro rol para los clientes, que se ha lanzado en el resto.

Al igual que hemos hecho con el NTP, hemos sacado a variables lo que hemos considerado necesario como rutas a exportar o los servicios a abrir en el firewall.

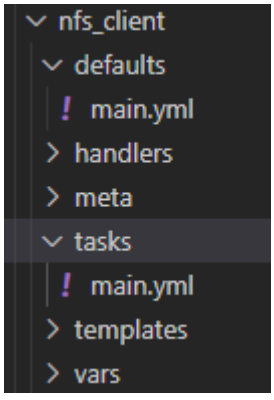
Hemos creado tres rutas a exportar dependiendo de la aplicación que despleguemos:

- /nfs_k8s: Para desplegar NGINX, es donde dejaremos un index.html para probar que funciona
- /nfs_k8s/mysql: Para desplegar un MaríaDB y usarlo en un wordpress
- /nfs_k8s/wordpress: para desplegar el contenido del wordpress en el apache

En la ejecución de las tareas podemos destacar la creación de los directorios, la instalación de los paquetes, la copia de los templates con la sustitución de las variables y la configuración del firewall



En la parte del cliente, las acciones a destacar son la instalación de los paquetes y la configuración de firewall. En esta parte creamos una tarea (se encuentra comentada) que nos montaba el filesystem nfs para poder comprobar que funcionaba correctamente.



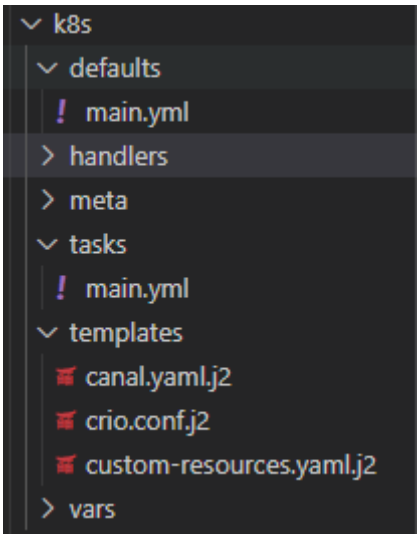
5. Instalación de K8s Master:

La instalación de K8S consiste en los siguientes pasos:

1- Desactivar el selinux, actualizar la maquina y reiniciar
2- Instalar el wget
3- Configurar parámetros del kernel y desactivar swap
4- Configurar repositorios he instalar paquetes CRI-O, kubernetes y iproute-tc
5- Configurar firewall
6- Copiar ficheros he inicializar k8s con los parámetros de la máquina. Destacamos que usamos ansible_facts para obtener IP y nombre de la maquina
7- Instalar Tigera, calico y HAProxy
8- Crear un fichero con el comando de unión al cluster

Es importante indicar que tuvimos que meter tiempos de pausa para que diese tiempo a arrancar los PODs

Al igual que hemos hecho en roles anteriores, tanto los puertos como otros parámetros necesarios los tenemos definidos como variables, bien globales o default, para luego utilizarlos en los templates o en las tareas



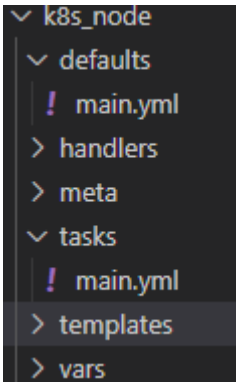
Como mejora de futuro y siguiendo esta estructura, simplificaríamos el script, sacando a otros ficheros cada uno de estos bloques e importándolos en el main.

6. Instalación de K8s Worker:

La instalación de los worker la podemos dividir en dos fases. La primera es la instalación de paquetes y la segunda la unión al cluster ejecutando el fichero creado en la instalación del master

Todo ello lo securizamos levantando el firewall y abriendo los puestos adecuados

Al igual que hemos hecho en roles anteriores, tanto los puertos como otros parámetros necesarios los tenemos definidos como variables, bien globales o default



7. Despliegue de aplicación:

Aunque en el enunciado de la práctica se indica que solo hace falta desplegar una aplicación, hemos decidido desplegar dos aplicaciones ya que la primera, nginx, se nos quedaba algo simple y aumentamos la dificultad montando un wordpress con su BBDD y su frontal

7.1. Despliegue de aplicación NGINX:

Lo primero que hemos hecho en el script de ansible ha sido montar el nfs y copiar un index.html para comprobar que la aplicación funciona. Luego desmontamos el nfs.

Para poder lanzar desde ansible comandos k8s, necesitamos instalar el cliente Python de openshift

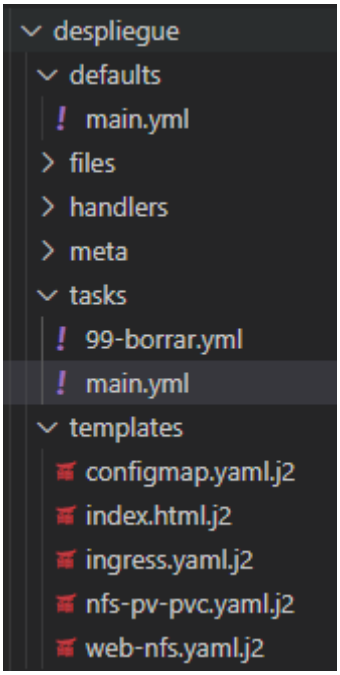
```
- name: Instalar cli py de openshift
  become: true
  pip:
    name: openshift
```

Con el cliente instalado en el Master, procedemos a ejecutar el despliegue de la aplicación.

1. Creamos el namespace
2. Creamos el almacenamiento (PV y PVC)
3. Creamos el POD y el servicio
4. Creamos el ingress
5. Configuramos el HAProxy
6. Obtenemos el puerto de nateo, la Ip del nodo HAProxy
7. Añadimos al /etc/hosts del master el nombre dado al host de la publicación de la aplicación y la ip del HAProxy
8. Lanzamos un CURL contra el HAProxy para ver si ha ido todo bien

```
TASK [despliegue : Obtener web] ************************************
ok: [20.77.71.187]

TASK [despliegue : Imprimir web] ***********************************
ok: [20.77.71.187] => {
    "msg": {
        "accept_ranges": "bytes",
        "changed": false,
        "connection": "close",
        "content_length": "97",
        "content_type": "text/html",
        "cookies": {},
        "cookies_string": "",
        "date": "Sat, 16 Jul 2022 10:07:27 GMT",
        "elapsed": 0,
        "etag": "\"62d28cfd-61\"",
        "failed": false,
        "last_modified": "Sat, 16 Jul 2022 10:03:41 GMT",
        "msg": "OK (97 bytes)",
        "redirected": false,
        "server": "nginx/1.23.0",
        "status": 200,
        "url": "http://practica:31407/practica"
    }
}
```

9. Desmontamos la aplicación (opcional mediante variable)



Para la creación de los recursos de K8S hemos usado templates con variables

```
- name: Crear almacenamiento (PV y PVC)
  kubernetes.core.k8s:
    state: present
    template: 'nfs-pv-pvc.yaml.j2'
```

Al igual que nos ha pasado en otros scripts, nos hemos visto obligados a meter pausas para dar tiempo a que los servicios arrancases

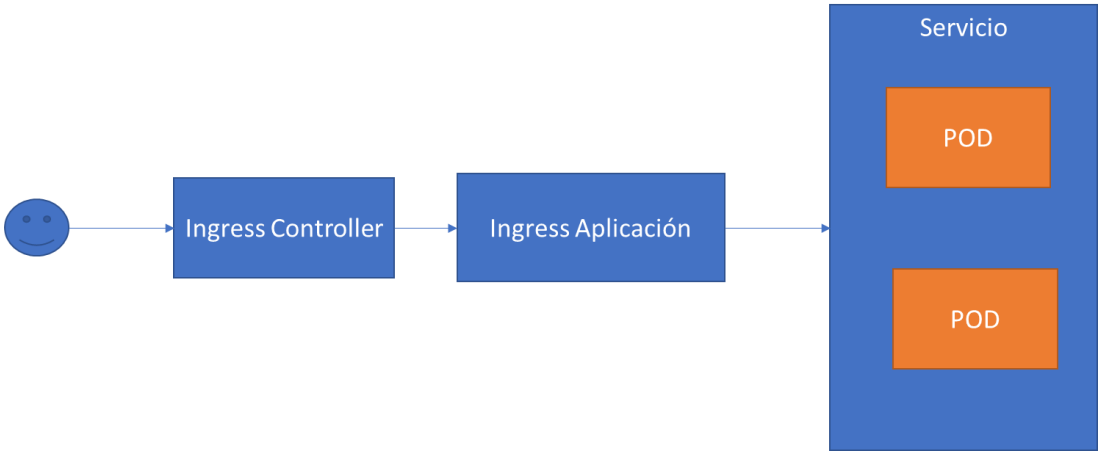
7.2. Despliegue de aplicación Wordpress:

La instalación de wordpress es similar a la nginx, compartiendo gran parte del código. Centrándonos solo en las diferencias, podemos observar que los PVC son dos en lugar de uno ya que necesitamos el de la BBDD y el del apache.

Por otro lado, tenemos los servicios que también son dos, el de la BBDD y el de los apaches.

La parte de publicación mediante ingress solo se hace en los apaches al igual que hicimos con el nginx

7.3. Funcionamiento de ingress:



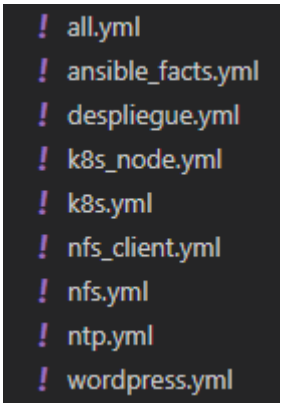
Cuando lanzamos el comando curl a la url, estamos invocando al servicio de ingress controller que tiene nateado el puerto 80, este a su vez encamina el tráfico al ingress de la aplicación que estamos publicando por el puerto 80.

Continuando el camino de la petición, llegaremos al servicio de la aplicación que a su vez contiene los pod desplegados.

8. Otros temas a destacar:

Queríamos destacar el tema de lanzar todo desde un solo fichero llamado caso2.sh donde están las invocaciones a terraform y a ansible. Como comentamos en puntos anteriores terraform, mediante los templates, crea el inventario y las variables globales.

Otro punto a tener en cuenta es la creación de playbook independientes para cada rol, para temas de pruebas y de ejecuciones controladas. El playbook all.yml es el que lanza todos los demás playbook y el invocado por caso2.sh



Por último, indicar que el usuario que se usa para ejecutar los scripts de ansible son en el terminal vagrant y en azure azureuser.

El azureuser esta añadido el fichero sudores para ejecutar todos los comandos sin necesidad de poner la password, con lo que no hemos necesitado añadir otro a este fichero.

Caso práctico 2. Parametrización del despliegue

Sera necesario indicar que variables o datos son necesarios cambiar dependiendo del entorno y como se debe ejecutar el plan de terraform y los playbooks de ansible.

Como comentábamos en el apartado anterior, la mayoría de la parametrización de ansible la envia terraform mediante templates.

Si nos viésemos obligasos a modificar la parametrización, vamos a indicar los ficheros a modificar.

1. **Terraform:**

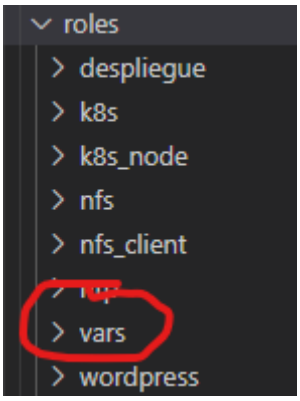
El fichero de parametrización es correccion-vars.tf y los parámetros configurados son los siguientes:

- **resource_group_name**: Grupo de azure
- **location_name**: lokalizacion de las maquinas en azure
- **network_name**: nombre de la red
- **subnet_name**: nombre de la subred
- **num_maquinas**: número de maquinas
- **usuario**: usuario que tendrá el sudo sin password para ser usado por ansible

- **path_rsa**: ruta donde tenemos las claves de ssh
- **disco**: tamaño del disco adicional
- **availability-size**: tipos de máquinas a montar

2. Ansible:

Por cada uno de los roles tenemos un fichero en la carpeta default, pero como decíamos anteriormente, terraform crea uno global, que se importa en todos los roles, en la carpeta var.



Desglosando por roles las variables tendríamos lo siguiente:

2.1. NTP:

- **ntp_timezone:** Zona horaria
- **ntp_servers:** servidores ntp

2.2. NFS:

- **nfs_exports:** Rutas y parámetros de los filesystem a exportar
- **nfs_directorio:** directorio local donde montar el nfs
- **service_firewall:** servicios de firewall a abrir

2.3. NFS_CLIENT

- **nfs_directorio:** directorio local donde montar el nfs
- **service_firewall:** servicios de firewall a abrir

2.4. K8S:

- **version:** Version de cri-o
- **os:** sistema operativo (lo podemos coger de ansible_facts) para ruta de cri-o

- **port_firewall:** Puertos del firewall
- **red_pod:** red de los pod
- **usuario:** usuario que ejecutara kubectl
- **grupo:** grupo del usuario

2.5. K8S_NODE

- **version:** Version de cri-o
- **os:** sistema operativo (lo podemos coger de ansible_facts) para ruta de cri-o
- **port_firewall:** Puertos del firewall
- **nombre:** Nombre del servidor de k8s master
- **usuario:** usuario que ejecutara kubectl
- **grupo:** grupo del usuario

2.6. DESPLIEGUE

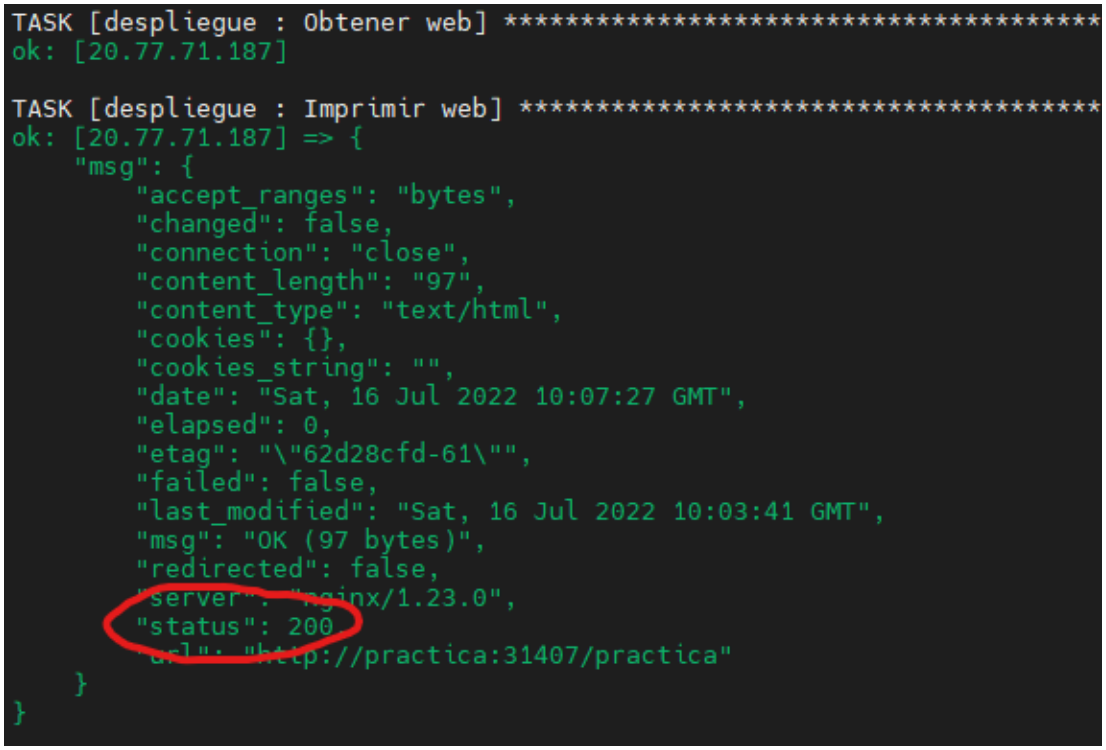
- **borrar:** activa el borrado de los pod si todo va bien
- **nfs_server:** Servidor nfs
- **nfs_path:** path NFS
- **capacidad:** tamaño del PV
- **namespace:** nombre del namespace
- **nombreapp:** nombre de la aplicación
- **maquina:** nombre que le vamos a dar a la exposición mediante ingress a la aplicación
- **apppath:** path de la aplicación
- **nfs_directorio:** Donde montamos el nfs para acceder a el

Caso práctico 2. Descripción de la aplicación

Describir el funcionamiento de la aplicación. Como se utilizaría para que un cliente pueda evaluarla.

Las dos aplicaciones que hemos montado (nginx y wordpress) son web y con lanzar un curl a la url, nos valdría para comprobar que esta funcionando

No obstante, el propio script de ansible lanza una llamada a la url para comprobar su funcionamiento, preparando el script para dar marcha atrás un despliegue si saliese mal.



Caso práctico 2. Problemas encontrados

Deberás describir los problemas que has encontrado, si los hubiera, cómo se han solucionado y referencias que se hayan utilizado para resolverlas (en formato APA). Si algún problema no se ha podido solucionar, se deberá analizar y proponer soluciones o siguientes pasos, el motivo por el que se cree que ha fallado…

Los diferentes problemas encontrados son los siguientes:

1- Tarjeta de red de vagrant: El problema fue que la tarjeta de red que crea vagran la asigna como eth0 y es de tipo NAT. Esto hace que k8s la coja por defecto y los pod no se vean.

La única solución que se me ocurrió fue borrar la tarjeta de red una vez creadas las máquinas. Esto no aplica a la practica en azure pero se ha dejado el código comentado a forma de recordatorio futuro.

2- Instalación de K8S con el modulo de ansible no permite excluyendo el paquete kubernetes.

No encontramos solución y optamos por ejecutar como comando Shell

3- Instalación de canal deja estos pod parados.

No encontramos solución y optamos por no instalarlo ya que solo con calico funciona perfectamente

Caso práctico 2. Licencia

Se deberá indicar la licencia utilizada e indicar las restricciones y el uso que permite la licencia

Hemos optado por una licencia MIT que “es corta, libre y permisiva, con condiciones que solo requieren la preservación de los avisos de derechos de autor y licencia. Los productos con licencia, las modificaciones y los productos más grandes, pueden distribuirse bajo diferentes términos y sin código fuente” (github, 2022).

Asignatura	Datos del alumno	Fecha
Experto Universitario en DevOps & Cloud	Apellidos: Arenas Morante	14/07/2022
	Nombre: Iván	

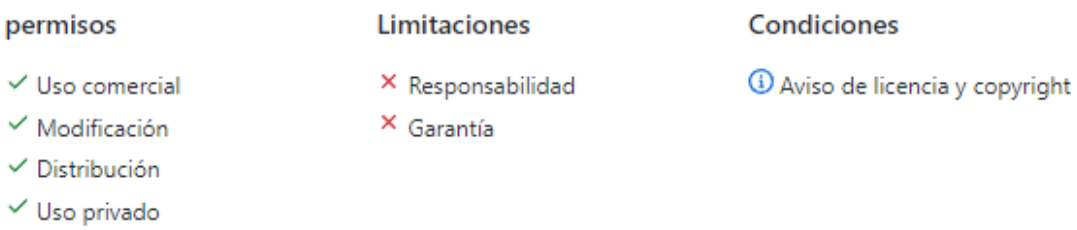


Ilustración 2Licencia MIT Captura de GITHUB (github, 2022)

Nota: para su entrega, dicho documento de plantilla se ha de exportar como PDF.

En la plantilla de la solución se ha de incorporar el enlace al repositorio de código del alumno con el código fuente como propuesta de la solución (en la portada de este presente documento).

Referencias

github. (16 de 07 de 2022). *GitHub.com*. Obtenido de https://github.com/aremox/caso_pracico_2/blob/main/LICENSE