Automatización de despliegues en entornos Cloud

1. Objetivos de la actividad

Esta actividad pretende conseguir que te familiarices con las tecnologías de automatización y Cloud.

Implementar una solución IaS Automatizada utilizando Terraform y Ansible

Crear un entorno en Kubernetes

Desplegar una aplicación.

1. Introducción

Esta practica consiste en el despliegue de una aplicación utilizando contenedores a través de la tecnología de Kubernetes.

Para el desarrollo de esta actividad se hace uso de tecnologías como Terraform para el despliegue del entorno en Azure, Ansible para la instalación de la infraestructura de kubernetes y el servidor de NFS.

En el desarrollo se utilizó imágenes predefinidas en dockerhub a modo de demostrar que la aplicación funciona.

1. **Diagramas:**

* **Infraestructura en Azure (Terraform):**

Se desarrollo un programa escrito en HCL usando terraform para desplegar 3 maquinas virtuales que se configuraron como un cluster de kubernetes. Cada VM cuenta con una IP privada y una IP publica para comunicarse con el exterior y se utilizo una imagen gratuita de CENTOS para su sistema operativo. Cada maquina esta asociada a un security group el cual tiene permitido el acceso al puerto 22 para poder acceder vía SSH y al puerto 30000 para que la aplicación sea expuesta hacia el exterior. Toda la instalación se realizo dentro de un resource group que permite agrupar a cada uno de los elementos.



* **Ansible:**

Se ha utilizado ansible para el despliegue automatico de Kubernetes. Los ficheros utilizados para esta práctica están estructurados de la siguiente manera:

**ansible/**

├── 01-common.yml

├── 02-nfs.yml

├── 03-master-worker.yml

├── 03-master.yml

├── 04-master.yml

├── 05-workers.yml

**├── group\_vars**

│   ├── app.yaml

│   ├── common.yaml

│   ├── master\_workers.yaml

│   ├── master.yaml

│   ├── nfs.yaml

│   └── workers.yaml

├── hosts

├── implementacion-aplicacion.yml

├── implementacion-kubernetes.yml

**└── roles**

**├── app**

**│   ├── files**

│   │   ├── Paso1\_redis-master.yaml

│   │   ├── Paso2\_redis-master-service.yaml

│   │   ├── Paso3\_redis-slave.yaml

│   │   ├── Paso4\_redis-slave-service.yaml

│   │   ├── Paso5\_frontend-service.yaml

│   │   └── Paso6\_frontend.yaml

**│   └── tasks**

│   └── main.yaml

**├── common**

**│   └── tasks**

│   └── main.yaml

**├── master**

**│   ├── files**

│   │   └── custom-resources.yaml

**│   └── tasks**

│   └── main.yaml

**├── master\_workers**

**│   └── tasks**

│   └── main.yaml

**├── nfs**

**│   └── tasks**

│   └── main.yaml

**└── workers**

**└── tasks**

└── main.yaml

Cada archivo yml de la rama principal (ansible) llama a un rol del directorio roles para ejecutar todas las acciones, y a su vez estos roles llaman a las variables que se encuentran en la carpeta group\_vars.

Así, por ejemplo, si se ejecuta el fichero 01-common.yml, este llamara al role common que contiene un task main.yaml adentro. Este task a su vez utilizara las variables que se encuentran en el archivo common.yaml del directorio group\_vars.

* **Kubernetes:**

Para la instalación y el despliegue de Kubernetes se utilizó Ansible. Esta herramienta nos permite automatizar la instalación y configuración de todo el cluster de kubernetes, además de hacer el despliegue automático de la aplicación. Para este ejercicio se cuenta con un nodo Master y un nodo Worker que serán los encargados de contener toda la infraestructura de kubernetes (pods, servicios, deployments, etc) y un servidor NFS que se utilizara para almacenamiento compartido en la red. Debido a las limitaciones de la cuenta de Azure, solo fue posible desplegar 3 maquinas virtuales y no 4 como sugería el enunciado debido a que hay limitantes en el número de vCPUs que se pueden configurar.



* **Aplicación:**

La aplicación seleccionada para la práctica es un guestbook, el cual permite a través de una interfaz web almacenar cadenas de texto en una base de datos REDIS. La estructura de la aplicación esta formada por los siguientes elementos:



En primer lugar, tenemos una Base de Datos Redis Master, la misma que va a contener de manera persistente la información. Por otro lado, tenemos 2 réplicas de nodos esclavos de Redis que funcionan de cache de los datos para mejorar la lectura de los datos. Para comunicar los pods es necesario crear services, en este caso al tener mas de 1 replica de los nodos esclavos se creo un servicio de tipo ClusterIP para comunicar a otros estos pods bajo una subred. Por otro lado, se generó otro servicio de tipo CLuster IP (Redis Master) para tener comunicación entre los nodos esclavos y el master. Se generaron 2 réplicas de los front-end con el fin de balancear la carga del trafico entrante y finalmente se genero un servicio que puede ser de tipo Nodeport o LoadBalancer para que se comunique con el exterior. Para esta practica y por la facilidad que presenta se uso un servicio del tipo Nodeport que mapea el puerto 80 del contenedor hacia el puerto 30000 del nodo.

1. **Manual de uso de la aplicación:**

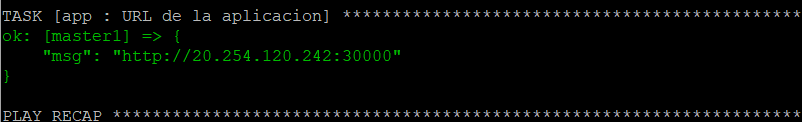
* **Prerrequisitos:**
  + Cuenta en Azure
  + Maquina con Linux donde se pueda correr un bash script
  + En caso de que se despliegue on-premise, configurar 3 maquinas virtuales como describe el ejercicio
* **Como desplegar la aplicación para azure:**
  + Clonar el repositorio de github en una maquina local

<https://github.com/jimpolbmx1/Caso_Practico2_Juan_Pablo_davalos.git>

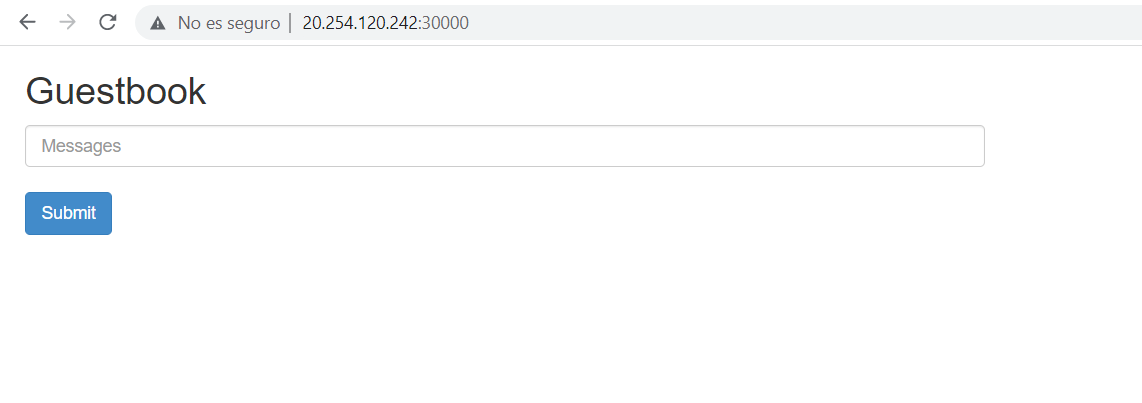
* + Ingresar al directorio terraform y abrir con un editor de text el archivo vars.tg
  + Modificar los valores de la suscripción con los propios de la cuenta:



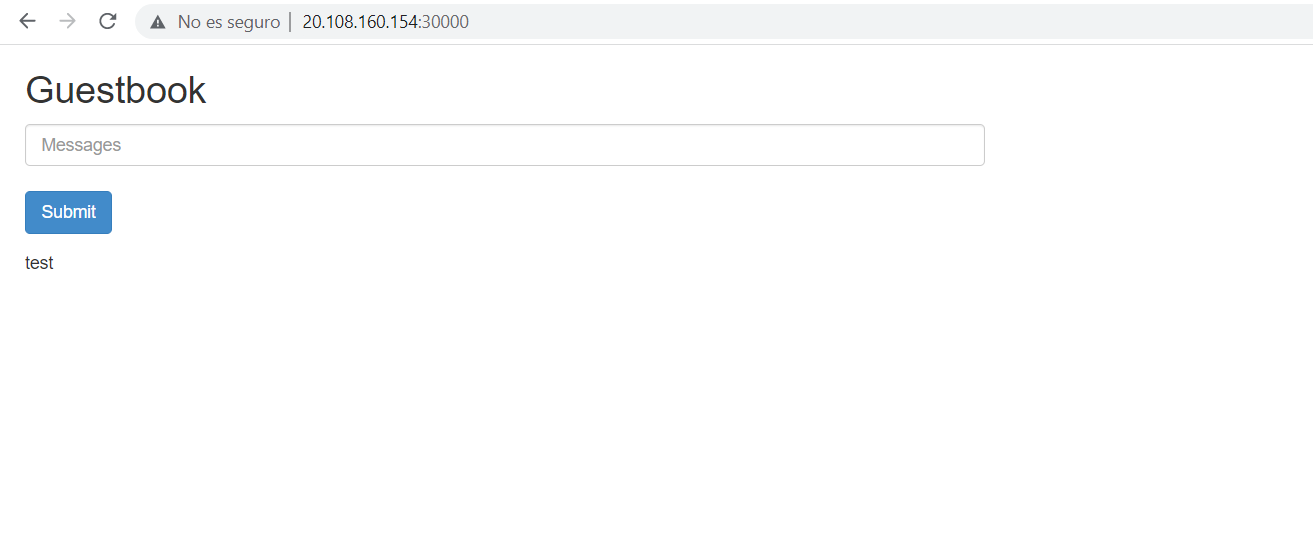
* + Ya con los valores cambiamos nos movemos al directorio raíz y ejecutamos el script “Deploy\_application.sh”
  + Al finalizar, el script nos mostrara la URL de la aplicación, que puede ser ejecutada desde cualquier browser:



* + Abrir la URL en el browser en ingresar cualquier cadena de caracteres



* + En la parte de abajo se puede verificar como las cadenas se guardan en la base de datos:



* Como desplegar la aplicación on-premise (Comprobado con VMWare):
  + Acceder al directorio ansible y abrir el archivo hosts.
  + Editar las IPs de los nodos Master, worker y nfs y agregar los de la infraestructura que ya han sido previamente generadas
  + Por defecto nuestra cluster en kubernetes corre sobre un SDN en flannel, ya que la practica exige que se corra la aplicación sobre Máquinas Virtuales en Azure, pero si se desea correr la aplicación usando el SDN de Calico, es necesario editar las variables que se encuentran en el siguiente directorio: ansible\group\_vars\master.yaml
  + Remplazar el valor de “flannel” por “calico”
  + Editar el archivo \group\_vars\master\_workers.yaml y remplazar la IP del nodo NFS:

private\_ip\_nfs: "<IP NFS>"

* + Editar el archivo \group\_vars\nfs.yaml y remplazar las IPs de los nodos Master y Worker:

private\_ip\_master: "<IP Master>"

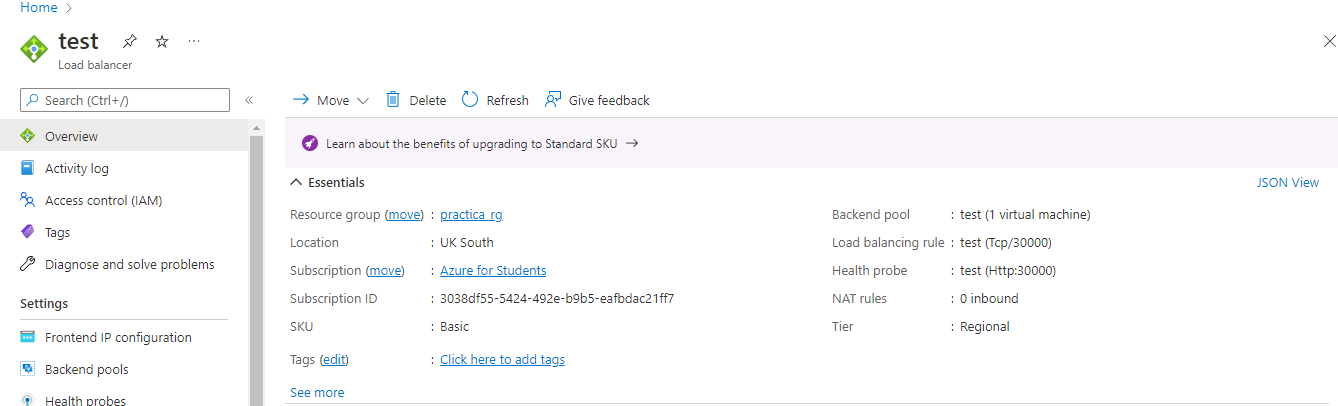
private\_ip\_worker: "<IP Worker>"

* + Correr los comandos siguientes desde el directorio ansible:
    - ansible-playbook -i hosts implementacion-kubernetes.yml
    - ansible-playbook -i hosts implementacion-aplicacion.yml
  + Al correr el ultimo playbook la aplicación mostrara la URL desde donde se puede acceder y probar la aplicación.

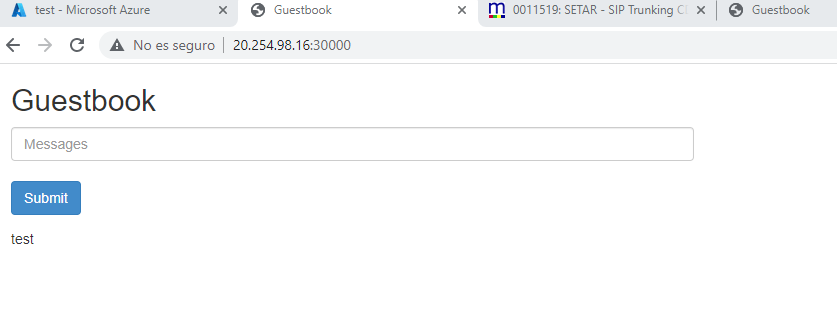
1. **Problemas encontrados en la solución:**

* Para una solución cloud, no es recomendable usar un Ingress controller como HA Proxy, en su lugar lo mejor es usar un servicio del tipo loadbalancer y tener un balanceador externo. En el caso de Azure se realizaron pruebas generando un loadbalancer externo y se genero un pool interno con las maquinas virtuales con lo cual cuando se hacia una consulta a la IP pública del balanceador de carga apuntaba a cualquier pod de cualquier Nodo. Por ejemplo:

Se genero un loadBalancer denominado test, que apunta al nodo master del cluster en el puerto 30000(donde escucha la aplicacion):



Al probar logramos el mismo resultado que apuntando a la IP pública del nodo master:



* La cuenta estudiantil de Azure es muy limitada y se tuvo que configurar el nodo master con diferentes características al nodo worker. Cuando se tiene un balanceador de carga en la nube, es necesario que las maquinas virtuales sean de similares características, pero en este caso al no poder generarlas igual, no se pudo probar correctamente el balanceador de carga.