Tabla de Contenidos

Descripción

El despliegue de la solución está conformado por dos componentes fundamentales, la Infraestructura y la Aplicación.

La Infraestructura nos provee de los recursos necesarios para que puedan ejecutarse todos los servicios computacionales.

La Aplicación nos brinda la solución informática del requerimiento.

En este documento detallaremos cómo están compuestos cada uno de ellos y cómo se realiza el despliegue para poder tener una solución funcional.

Infraestructura

Los requerimientos solicitados para esta solución debían componerse de:

- 1 Nodo Master
- 2 Nodos Worker
- 1 Nodo NFS
- 1 Disco para NFS
- IP pública para acceder a los servicios
- 1 Ingress-Controller para acceder a los servicios

Debido a las limitaciones propuestas por la capa de estudiante que Azure propone los requerimientos anteriormente mencionados no pudieron satisfacerse, sin embargo se presenta la siguiente solución para poder cumplimentar con el funcionamiento de la aplicación (Ver diagrama 1):

- 1 Nodo Master
- 2 Nodos Worker
- Servicio NFS corriendo en uno de los Nodos Worker
- 1 Disco para NFS
- IP pública para acceder a los servicios

- Ingress-Controller para acceder a los servicios

Aparte de los requerimientos se han creado recursos extra para poder brindar un mejor manejo de las contraseñas, la seguridad de los servidores y el resguardo de la información.

Azure Key Vault	Utilizado para el resguardo de información confidencial, como contraseñas, certificados y otros. En el caso de la solución propuesta se utiliza para guardar los datos de contraseña del usuario administrador de la base de datos MySQL y el usuario y contraseñas de la base de datos de la aplicación que se ha seleccionado para desplegar: Ghost
Seguridad de acceso a los servidores	Se ha decidido optar por dotar de acceso SSH sólo al Nodo Master, restringiendo dicho acceso sólo a la máquina desde donde se realizará el despliegue. Para poder acceder a los Nodos Worker, utilizaremos las instrucciones Proxy de SSH, llegando a las máquinas virtuales a través del Nodo Master.
Alias de DNS	Se ha configurado un alias de DNS para poder acceder a los servicios mediante una URL.
Certificados SSL	Para asegurar la comunicación cliente-servidor se ha provisto una solución de generación de certificados Letsencrypt, los que automáticamente se generan y se actualizan gracias a una solución implementada en Kubernetes.

Aplicación

La solución elegida para esta demostración ha sido Ghost. Es una plataforma de Blogging similar a WordPress. La aplicación se despliega en un ambiente de Kubernetes, brindándose la solución de NFS para leer/escribir datos persistentes y MySQL para alojar las bases de datos.

Se ha seleccionado esta aplicación porque nos permite la utilización de NFS para guardar ficheros de imágenes de los distintos blogs, la comunicación con otros servicios de Kubrernetes (mysql-svc) y la utilización de varios Pods del servicio con un Ingress-Controller que maneja sesiones.

Kubernetes es desplegado en 3 Nodos, con las siguientes características técnicas:

- 1 Nodo Master – 2 Cores, 4 Gb de RAM, 1 HD con el SO
- 2 Nodos Worker – 1 Core, 2 Gb de RAM, 1 HD con el SO

Aparte de Kubernetes se despliega un servidor de NFS, que se aloja en uno de los Nodos Worker, donde se monta un HD extrerno de 10 Gb en el que se guardarán los datos de la aplicación Ghost y los ficheros de configuración de la misma.

El despliegue final de la aplicación será formado por los siguientes componentes (diagrama 2, diagrama 3):

- Docker
- Kubernetes
    - 1 Nodo Master
    - 2 Nodos Worker
- Ingress-Controller
- NFS
- MySQL
- Cert-Manager
- Ghost
    - 1 Servicio
    - 1 Definición de Deployment con 3 Réplicas

- 1 Definición de Ingress
- 1 Configmap
- 1 Secret con la contraseña de Ghost
- 1 TLS issuer

La aplicación se accede mediante un alias de DNS que nos facilita Azure al momento de crear la dirección IP estática, utilizada para acceder al Nodo Master. Cuando nuestra petición llega al Nodo Master un Ingress-Controller (Ingress-Nginx-Controller) atiende la solicitud, sirviendo una conexión segura con TLS via utilizando el protocolo HTTPS y conectando al servicio de Ghost que atiende en el puerto 2368 utilizando el protocolo HTTP. Todas las conexiones entrantes son servidas mediante el protocolo HTTPS.

Luego de que el proyecto haya sido desplegado podremos ingresar a la URL proporcionada por la salida de Terraform en "*ghost_http_service_url*" o "*ghost_https_service_url*".

Para configurar la aplicación, definir usuario administrador y poder comenzar a crear blogs debemos ingresar a "*ghost_admin_service_url*"

Despliegue:

El despliegue se realiza de una manera automática, pero para lograr que el mismo arranque hay ciertos pasos a seguir y ciertas recomendaciones a tener en cuenta que aquí se detallan.

Todos los datos se guardan en un repositorio git para poder versionar la infraestructura como código (IaC). De esta manera siempre podremos revertir los cambios que realicemos y volver a una versión pasada sin inconvenientes.

Como todo repositorio genérico, existen datos que deben ser configurados para poder adaptarse a las diferentes necesidades de cada entorno, es por ello que la solución permite configurar el entorno por medio de variables (véase "Variables de despliegue").

Vista general:

El despliegue se realiza desde cualquier máquina cliente que tenga Docker instalado. De esta manera ejecutaremos un contenedor que contiene todas las herramientas necesarias para que podamos utilizar Terraform, Ansible y Azure independientemente del SO del cliente. Dentro del repositorio se encuentra el Dockerfile que se utilizó para crear la imagen (https://hub.docker.com/r/mvilla/casopractico2)

Primeramente se configura el entorno que desplegaremos por medio de variables, luego se aplican los cambios y Terraform se encarga de Crear el entorno y de ejecutar Ansible para configurar Kubernetes y la Aplicación de Ghost seleccionada.

El único punto de acceso a los servidores es por medio de la IP pública asociada al servidor Master, es por ello que se ha generado un script (bin/run.sh) que te dará la posibilidad de conectarte a cualquiera de los servidores de manera sencilla.

Descripción de pasos:

El proceso general de despliegue se divide en 10 pasos que se detallan a continuación

Paso 1º. Clonación del repositorio:

Parece lógico ya que esto lo estamos leyendo desde el mismo repositorio o en nuestra máquina local, pero el primer paso para comenzar es clonar el repositorio a nuestra máquina local, desde la que desplegaremos la Infraestructura y la Aplicación.

Para clonar el repositorio ejecutaremos:

```
git clone https://github.com/mvicha/k8s-azure.git
```

Esto descargará una copia del repositorio en el directorio k8s-azure dentro del directorio desde el que ejecutamos el comando anterior

Paso 2º. Preparación del entorno de ejecución:

Para facilitar la ejecución de la solución y no tener que instalar ningún tipo de requerimientos, así como para evitar la colisión de versiones de paquetes y los fallos he optado por incluir un contenedor de Docker desde el que se realizarán todos los pasos. Para ello debemos seguir ciertos procedimientos.

```
mkdir -p k8s-azure-data k8s-azure-data/.azure k8s-azure-data/.kube k8s-azure-data/.ssh
```

Con la línea anterior estamos creando una nueva estructura de directorios en donde se guardarán los datos desde nuestro entorno local para utilizar dentro del contenedor de Docker:

```
k8s-azure-data/
k8s-azure-data/.azure/
k8s-azure-data/.kube/
k8s-azure-data/.ssh/
```

Paso 3º. Inicio del entorno de ejecución:

Una vez completados los pasos anteriores, procederemos a dar inicio a nuestro entorno de ejecución. Para ello montaremos los directorios que acabamos de crear, y el directorio del repositorio en nuestro contenedor y le daremos inicio

docker container run --name casopractico2 -it --rm -v ${PWD}/k8s-azure-data/.azure:/root/.azure -v ${PWD}/k8s-azure-data/.kube:/root/.kube -v ${PWD}/k8s-azure-data/.ssh:/root/.ssh -v ${PWD}/k8s-azure:/SAFE_VOLUME mvilla/casopractico2:latest /bin/bash

Con la línea anterior estamos diciéndole a Docker que inicie un nuevo contenedor en una terminal de bash basado en la imagen "*mvilla/casopractico2*" con nombre "*casopractico2*", que nos de una "*terminal interactiva*", que elimine el contenedor al finalizar la ejecución, que monte el directorio "*k8s-azure-data/.azure*" dentro de "*/root/.azure*", el directorio "*k8s-azure-data/.kube*" dentro de "*/root/.kube*", el directorio "*k8s-azure-data/.ssh*" dentro de "*/root/.ssh*" y que finalmente monte el directorio "*k8s-azure*" dentro de "*/SAFE_VOLUME*".

Paso 4º. Creación de la cuenta de almacenamiento para guardar el estado de Terraform:

Aunque no es requerido, se recomienda seguir los pasos para guardar el estado de Terraform de manera remota. Esto nos dará la posibilidad de poder tener un resguardo del fichero en el que se guardan todos los objetos que han sido desplegados. Me ha sucedido en ocasiones que he eliminado el directorio completo de trabajo, ya sea por accidente o porque quería desechar los cambios y con él he eliminado el fichero de estado de Terraform, ya que no quedan guardados en ningún repositorio.

Para poder crear la cuenta de almacenamiento y configurar el fichero de estado procederemos a ejecutar los siguientes comandos:

a) Primero nos ubicaremos en el directorio de trabajo desde el que ejecutaremos los comandos

cd /SAFE_VOLUME

b) Luego iniciaremos sesión en la cuenta de Azure utilizando az-cli

az login

c) Creación del Grupo de Recursos, Cuenta de Almacenamiento y Contenedor (más información en la sección "*utilidades desarrolladas*")

utils/storage_account.sh

d) De la ejecución anterior obtenemos la salida y guardamos el resultado en el fichero terraform/state.tf

```
cat <<EOF > terraform/state.tf
terraform {
  backend "azurerm" {
    resource_group_name   = "tstate"
    storage_account_name  = "tstateXXXX"
    container_name        = "tstate"
    key                   = "terraform.tfstate"
    access_key                                                    =
"XXXXXXXXXXXXXXXXXXXYYYYYYYYYYYYZZZZZZZZZZZZZ"
  }
}
EOF
```

e) Desconexión de la cuenta de Azure

az logout

*Paso 5º. Opcional para nuevos desarrollos que no posean un "*Service Principal*" existente en Azure (Los profesores que utilizan su entorno existente no deberán seguir este paso nº 5):*

Para brindar acceso a los servicios de Azure por medio de un "*Service Principal*" se deberán realizar los siguientes pasos:

a) Inicio de sesión en la cuenta de Azure utilizando az-cli

```
az login
```

b) Definición de la subscripción que utilizaremos por defecto (Requerido en caso de que la cuenta tenga más de una subscripción)

```
az account -set --subscription "SubscriptionID"
```

c) Creación del "*Service Principal*" en el servicio "*Active Directory*" de Azure

```
az ad sp create-for-rbac --role "Contributor"
```

d) Con los valores obtenidos completaremos el fichero terraform/credentials.tf

```
cat <<EOF >terraform/credentials.tf
provider "azurerm" {
   subscription_id = "Valor del id de subscripción que utilizaremos"
   client_id       = "Valor de appId obtenido en el paso anterior"
   client_secret   = "Valor de password obtenido en el paso anterior"
   tenant_id       = "Valor de tenant obtenido en el paso anterior"

   features {}
 }
EOF
```

Paso 6º. Los profesores que utilizarán su entorno deberán seguir los siguientes pasos:

Para brindar acceso a los servicios de Azure por medio de un "*Service Principal*" se deberá utilizar el fichero credentials.tf que poseen los profesores. Este deberá copiarse dentro de terraform/credentials.tf

El fichero terraform/credentials.tf se verá de la siguiente manera:

```
provider "azurerm" {
    subscription_id = "Valor del id de subscripción que utilizaremos"
    client_id       = "Valor de appId obtenido en el paso anterior"
    client_secret   = "Valor de password obtenido en el paso anterior"
    tenant_id       = "Valor de tenant obtenido en el paso anterior"

    features {}
  }
```

Paso 7º. Configuración de las variables de desarrollo de Infraestructura y Aplicación:

Tanto la Infraestructura como la Aplicación son personalizables. Para ello deberemos crear el fichero terraform/correccion-vars.tf done parametrizaremos los valores requeridos para la aplicación. Para más detalle ver la sección "*variables de despliegue*".

Creación del fichero:

```
cat <<EOF >terraform/correccion-vars.tf
variable "location" {
  type        = string
  description = "Región de Azure donde crearemos la infraestructura"
  default     = "West Europe"
}
```

```
variable "storage_account" {
  type        = string
  description = "Nombre para la storage account"
  default     = "myuniquesa"
}

variable "ssh_user" {
  type        = string
  description = "Usuario para hacer ssh"
  default     = "mysshuser"
}

variable "prefix" {
  type        = string
  description = "Prefijo que se utilizará para crear nombre únicos"
  default     = "myuniqueprefix"
}

variable "ghost_dns_alias" {
  type        = string
  description = "Alias que se utilizará para acceder via URL al servidor de Ghost"
  default     = "myownghost"
}

variable "my_ip" {
  type        = string
  description = "Dirección IP de la máquina local. Necesaria para conectar a los servidores vía SSH"
  default     = "149.91.115.233/32"
}
```

```
# No se utiliza. La solución automáticamente crea una clave SSH para la conexión interna y externa.
variable "public_key_path" {
  type        = string
  description = "Ruta para la clave pública de acceso a las instancias"
  default     = ""
}

# No se utiliza. La solución automáticamente crea una clave SSH para la conexión interna y externa.
variable "private_key_path" {
  type        = string
  description = "Ruta para la clave privada de acceso a las instancias"
  default     = ""
}
```

Es importante cambiar los valores del fichero anteriormente creados con los que nos gusten, para más información consultar la sección "*variables de despliegue*".

Paso 8º. Despliegue de Infraestructura y Aplicación:

Habiendo configurado todos los pasos anteriores llega el momento de desplegar la Infraestructura y la Aplicación. Esto se puede realizar de dos maneras:

a) Ejecutando el script que se ha desarrollado (más información en la sección "*utilidades desarrolladas*")

```
bin/run.sh create
```

b) Ejecutando de forma manual:

```
cd terraform
terraform init
```

terraform plan -out plan.out
terraform apply plan.out

Cualquiera de las dos opciones dará el mismo resultado.

El tiempo de ejecución estimado es de unos 27 minutos.

Con la finalización satisfactoria del proceso de creación ya tendremos nuestra Infraestructura y Aplicación corriendo en Azure. El resultado del proceso anterior debería mostrarnos por pantalla lo siguiente:

Apply complete! Resources: 37 added, 0 changed, 0 destroyed.

Outputs:

WARNING = ""
ansible_exec_command = "ansible-playbook -e jump_host='20.86.240.158' -e admin_user='azureuser' -e subnet_cidr_private='192.168.1.0/24' -e private_lan_master='192.168.1.100' -e private_lan_node01='192.168.1.101' -e private_lan_node02='192.168.1.102' -e internal_private_key_file='resources/internal.pem' -e external_private_key_file='resources/external.pem' -e ghost_url='ghosthope.westeurope.cloudapp.azure.com' -e prefix='mfvilla' ../ansible/playbooks/create.yml"
ghost_admin_service_url = "https://ghosthope.westeurope.cloudapp.azure.com/ghost"
ghost_http_service_url = "http://ghosthope.westeurope.cloudapp.azure.com"
ghost_https_service_url = "https://ghosthope.westeurope.cloudapp.azure.com"
master_private_address = "192.168.1.100"
master_public_address = "20.86.240.158"
node01_private_address = "192.168.1.101"
node02_private_address = "192.168.1.102"
ssh_admin_user = "azureuser"

ssh_master_connection = "ssh -i resources/external.pem azureuser@20.86.240.158"
ssh_node01_connection = "ssh -i resources/internal.pem -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null -o ProxyCommand='ssh -i resources/external.pem -W %h:%p -q azureuser@20.86.240.158' azureuser@192.168.1.101"
ssh_node02_connection = "ssh -i resources/internal.pem -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null -o ProxyCommand='ssh -i resources/external.pem -W %h:%p -q azureuser@20.86.240.158' azureuser@192.168.1.102"
subnet_cidr_private = tolist([
"192.168.1.0/24",
])
virtual_network_cidr = tolist([
"192.168.0.0/16",
])

Paso 9º. Prueba de la Aplicación:

Con la Infraestructura y la Aplicación desplegadas podremos abrir un navegador introduciendo la URL de "*ghost_http_service_url*" o "*ghost_https_service_url*" obtenidas en el output de la salida anterior. Cualquiera de las dos URLs que introduzcamos deberían terminar en una conexión HTTPS (diagrama 4).

En caso que hayamos perdido la salida anterior podemos obtenerla nuevamente de dos maneras:

a) Ejecutando el script que se ha desarrollado (más información en la sección "*utilidades desarrolladas*")

bin/run.sh info

b) Ejecutando de forma manual:

cd terraform
terraform output

Paso 10º. Acceso a los nodos Master y Worker de Kubernetes:

Por cuestiones de seguridad es el nodo Master el único que tiene una dirección IP pública asociada. Para acceder a este o a cualquiera de los nodos sólo nos bastaría con ejecutar:

bin/connect.sh <node>

Donde node tomaría la forma de alguno de los nodos:

- *master*
- node01
- *node02*

Ejemplo:

bin/connect.sh node01

Variables de despliegue

La siguiente es una lista que contiene las distintas variables que se utilizan en el despliegue, tanto de la Infraestructura como de la Aplicación. Se describe su objetivo, formato y ubicación:

Descripción de las variables:

virtual_network_cidr

Esta variable define el CIDR de nuestra red virtual. **Se recomienda no cambiarla**.

Tipo: String

Ubicación: terraform/variables.tf

subnet_cidr_private

Esta variable define el CIDR de nuestra red privada. **Se recomienda no cambiarla**.

Tipo: String

Ubicación: terraform/variables.tf

private_lan_master

Esta variable define la IP que utilizará el nodo master en la red privada. Debe estar en el rango de *subnet_cidr_private*.

Tipo: String

Ubicación: terraform/variables.tf

mysql_ghost_database

Esta variable define el nombre de la DB que se utilizará con la aplicación de blog.

Tipo: String

Ubicación: terraform/variables.tf

vm_size_master

Esta variable define el tipo de VM que se utilizará para el nodo Master. **Se recomienda no cambiarla**.

Tipo: String

Ubicación: terraform/variables.tf

workers

Esta variable define el nombre de cada uno de los nodos como objeto y conforma un array de parámetros para cada uno de ellos, definiendo la dirección IP privada de cada uno de los nodos y el tipo de VM que se utilizará. **Se recomienda no cambiarla**.

Tipo: Object

Ubicación: terraform/variables.tf

location

Esta variable define la ubicación donde se desplegará la plataforma.

Tipo: String

Ubicación: terraform/correccion-vars.tf

storage_account

Define el nombre de la “Cuenta de Almacenamiento” que se utilizará.

Tipo: String

Ubicación: terraform/correccion-vars.tf

ssh_user

Define el nombre de usuario que se creará dentro de las VMs de Azure para poder conectarse vía SSH con acceso privilegiado.

Tipo: String

Ubicación: terraform/correccion-vars.tf

prefix

Define un prefijo que se utilizará para identificar recursos de una manera única.

Tipo: String

Ubicación: terraform/correccion-vars.tf

ghost_dns_alias

Define un alias que se utilizará para en conjunto con el dominio de Azure para poder acceder a los servicios de la aplicación vía DNS.

Tipo: String

Ubicación: terraform/correccion-vars.tf

my_ip

Define la dirección IP pública de la máquina desde la que conectaremos a los servidores vía SSH.

Tipo: String

Ubicación: terraform/correccion-vars.tf

public_key_path

Define la ruta hacia la clave pública asociada a la llave de SSH que se utilizará para conectar a los servidores de Azure. *Si public_key_path o private_key_path contienen una cadena vacía Terraform creará las claves por nosotros*.

Tipo: String

Ubicación: terraform/correccion-vars.tf

private_key_path

Define la ruta hacia la clave privada de SSH que se utilizará para conectar a los servidores de Azure. *Si public_key_path o private_key_path contienen una cadena vacía Terraform creará las claves por nosotros*.

Tipo: String

Ubicación: terraform/correccion-vars.tf

Utilidades desarrolladas

La siguiente es una lista con la descripción de cada una de las utilidades desarrolladas y sus parámetros de utilización.

utils/storage_account.sh

Esta utilidad es utilizada para crear un "*Grupo de Recursos*" una "Cuenta de Almacenamiento" y un "Contenedor" de Azure para poder guardar el estado remoto de Terraform

Se ejecuta sin ningún parámetro y nos devuelve los valores que debemos ingresar en el fichero *terraform/state.tf* para guardar el estado de Azure de forma remota.

bin/run.sh

Esta utilidad se utiliza de acuerdo a las siguientes parámetros para distintas acciones

create:

Despliega la Infraestructura de Azure y la Aplicación de Kubernetes

Parámetro: Ninguno

Ejemplo: bin/run.sh create

deploy:

Despliega la Aplicación de Kubernetes

Parámetro: Ninguno

Ejemplo: bin/run.sh deploy

undeploy:

Elimina la Aplicación de Kubernetes de la Infraestructura

Parámetro: Ninguno

Ejemplo: bin/run.sh undeploy

destroy:

Elimina la Infraestructura y consigo la Aplicación de Kubernetes

Parámetro: Ninguno

Ejemplo: bin/run.sh destroy

info:

Muestra información del despliegue. Es lo mismo que terraform output

Parámetro: Ninguno

Ejemplo: bin/run.sh info

my_ip:

Muestra la dirección IP pública desde donde se ejecuta el script. Ideal para obtener nuestra dirección IP pública para utilizar en la configuración.

Parámetro: Ninguno

Ejemplo: bin/run.sh my_ip

bin/connect.sh

Esta utilidad se utiliza para conectar a los nodos vía SSH

Parámetro: Nombre del nodo a conectar (master, nodo01 o nodo02).

Ejemplo: bin/connect.sh node01

Diagramas

Los siguientes diagramas muestran gráficamente el resultado de la Infraestructura y la Aplicación desplegados por medio de Terraform y Ansible. Se incluye información de la relación de los roles asignados a cada nodo.

Diagrama 1: Despliegue

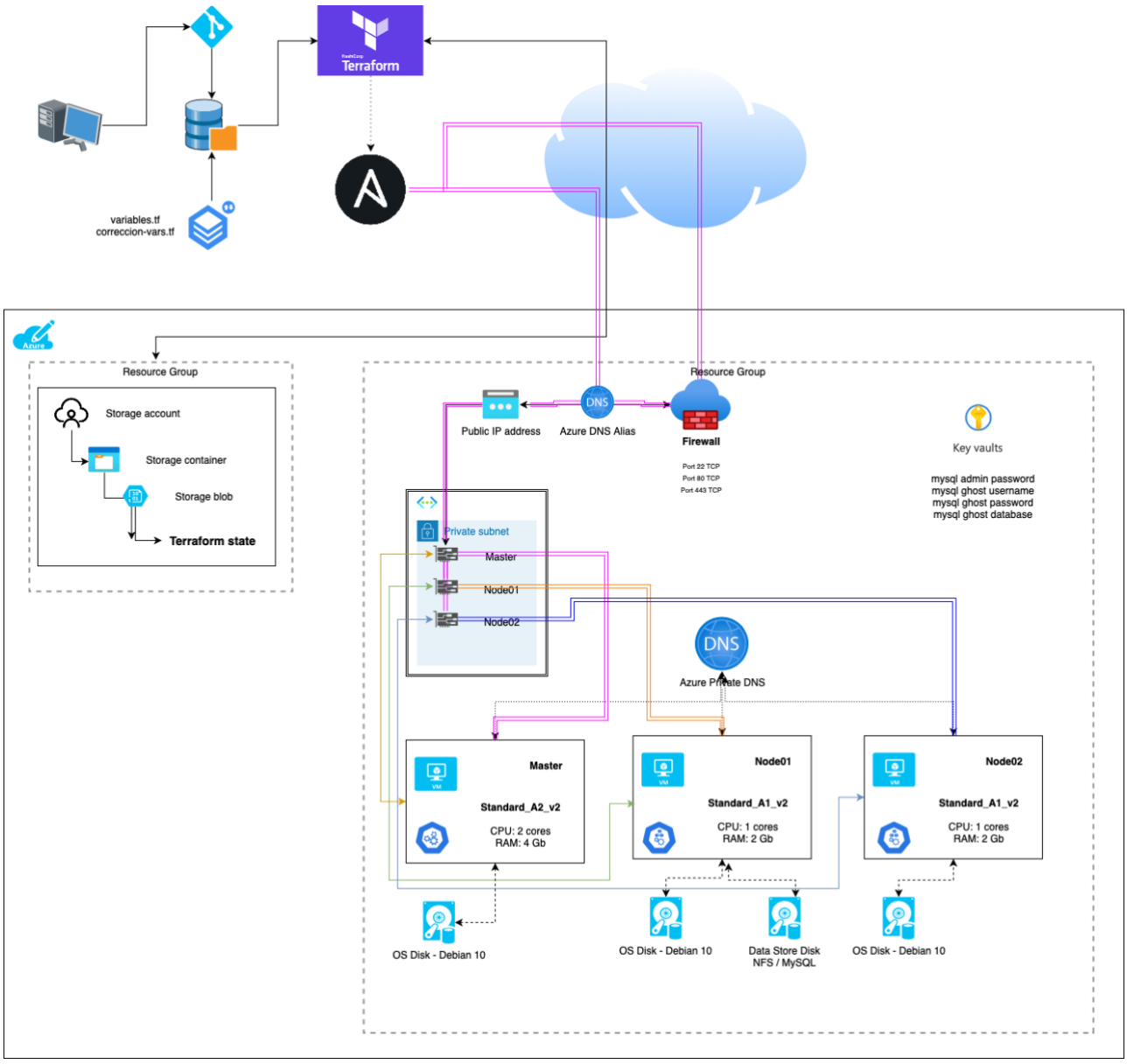


Diagrama 2: Relación de Roles Ansible con Nodos

Relación Playbooks de Ansible con Nodos

	Common	Nfs	Deployment-Tools	Docker	Kubernetes	Kubemasters	Kubenodes	Ingress-Nginx-Controller	Cert-Manager	MySQL	Ghost
Master	✔		✔	✔	✔	✔		✔	✔	✔	✔
Node01	✔	✔		✔	✔		✔				
Node02	✔			✔	✔		✔				

Diagrama 3: Servicios por VM

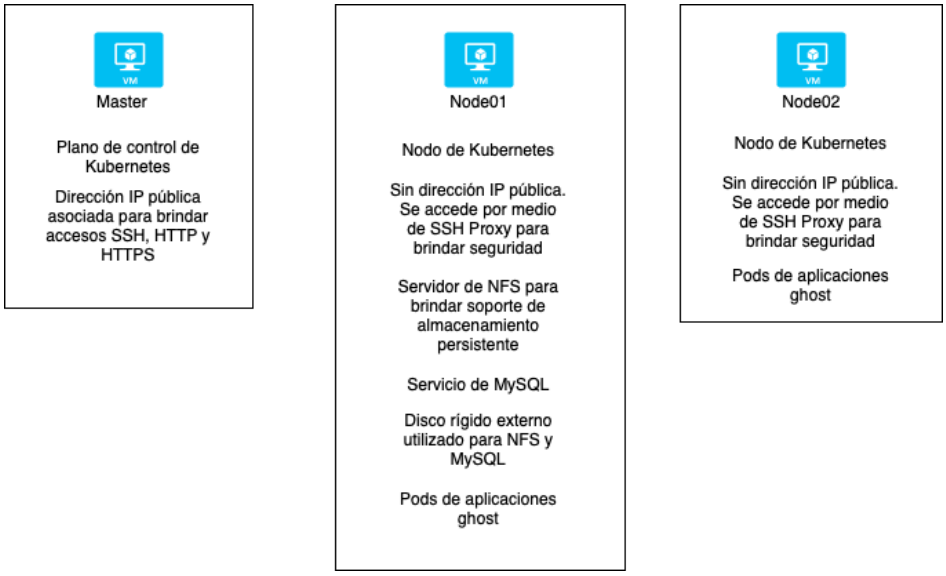
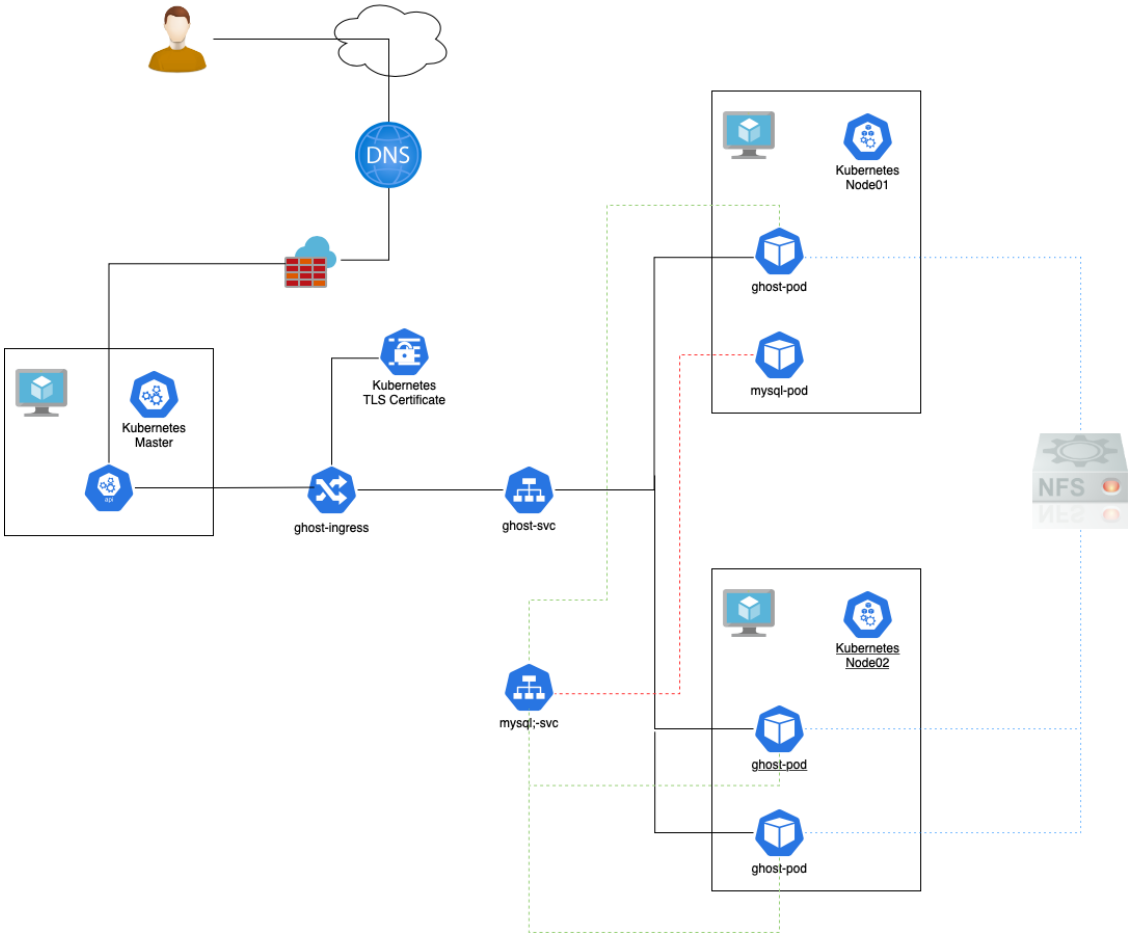


Diagrama 4: Kubernetes App



Descripción de Kubernetes

Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa. Tiene un ecosistema grande y en rápido crecimiento. El soporte, las herramientas y los servicios para Kubernetes están ampliamente disponibles.

Componentes

El siguiente es un detalle de los comoponentes de Kubernetes tomado de la página de Kubernetes dedicada a brindar información (https://kubernetes.io/es/docs/concepts/overview/components/)

Componentes del plano de control

Los componentes que forman el plano de control toman decisiones globales sobre el clúster (por ejemplo, la planificación) y detectan y responden a eventos del clúster, como la creación de un nuevo pod cuando la propiedad replicas de un controlador de replicación no se cumple.

Estos componentes pueden ejecutarse en cualquier nodo del clúster. Sin embargo para simplificar, los scripts de instalación típicamente se inician en el mismo nodo de forma exclusiva, sin que se ejecuten contenedores de los usuarios en esos nodos. El plano de control se ejecuta en varios nodos para garantizar la alta disponibilidad.

kube-apiserver

El servidor de la API es el componente del plano de control de Kubernetes que expone la API de Kubernetes. Se trata del frontend de Kubernetes, recibe las peticiones y actualiza acordemente el estado en etcd.

La principal implementación de un servidor de la API de Kubernetes es kube-apiserver. Es una implementación preparada para ejecutarse en alta disponiblidad y que puede escalar horizontalmente para balancear la carga entre varias instancias.

etcd

Almacén de datos persistente, consistente y distribuido de clave-valor utilizado para almacenar toda a la información del clúster de Kubernetes.

Si tu clúster utiliza etcd como sistema de almacenamiento, échale un vistazo a la documentación sobre estrategias de backup.

Puedes encontrar información detallada sobre etcd en su documentación oficial.

kube-scheduler

Componente del plano de control que está pendiente de los Pods que no tienen ningún nodo asignado y seleciona uno donde ejecutarlo.

Para decidir en qué nodo se ejecutará el pod, se tienen en cuenta diversos factores: requisitos de recursos, restricciones de hardware/software/políticas, afinidad y anti-afinidad, localización de datos dependientes, entre otros.

kube-controller-manager

Componente del plano de control que ejecuta los controladores de Kubernetes. Lógicamente cada controlador es un proceso independiente, pero para reducir la complejidad, todos se compilan en un único binario y se ejecuta en un mismo proceso.

Estos controladores incluyen:

Controlador de nodos: es el responsable de detectar y responder cuándo un nodo deja de funcionar

Controlador de replicación: es el responsable de mantener el número correcto de pods para cada controlador de replicación del sistema

Controlador de endpoints: construye el objeto Endpoints, es decir, hace una unión entre los Services y los Pods

Controladores de tokens y cuentas de servicio: crean cuentas y tokens de acceso a la API por defecto para los nuevos Namespaces.

Componentes de nodo

Los componentes de nodo corren en cada nodo, manteniendo a los pods en funcionamiento y proporcionando el entorno de ejecución de Kubernetes.

kubelet

Agente que se ejecuta en cada nodo de un clúster. Se asegura de que los contenedores estén corriendo en un pod.

El agente kubelet toma un conjunto de especificaciones de Pod, llamados PodSpecs, que han sido creados por Kubernetes y garantiza que los contenedores descritos en ellos estén funcionando y en buen estado.

kube-proxy

kube-proxy permite abstraer un servicio en Kubernetes manteniendo las reglas de red en el anfitrión y haciendo reenvío de conexiones.

Problemas encontrados y soluciones aplicadas

Durante el desarrollo de la solución me vi enfrentado a varias situaciones que demandaron una atención diferencial con respecto al resto de situaciones. Estas fueron en particular 2 (dos) situaciones que cabe destacar.

Limitación de Azure

Descripción:

El primer inconveniente al comenzar con el desarrollo fue la limitación impuesta por Azure a las cuentas educacionales o de capa gratuita. Esta limitación impidió que asignáramos la cantidad de máquinas previstas para el desarrollo planteado

Solución:

Para conservar la funcionalidad de Kubernetes y poder poner a prueba lo que planteaba el ejercicio, decidí continuar con la cantidad de máquinas asignadas a Kubernetes (1 Nodo Master + 2 Nodos Workers) y utilizar uno de los Nodos Worker para compartir el disco por NFS.

Esta solución a la vez favoreció a que utilizáramos el mismo disco NFS para los datos de la base de datos MySQL. Por lo tanto, dentro de Node01 se gestiona tanto NFS como servicio de sistema y MySQL como aplicación de Kubernetes.

Timeout

Descripción:

Dado que hemos restringido el acceso vía SSH lo máximo posible, y debemos utilizar el Nodo Master como Nodo de salto (Bastion) hacia los otros nodos, algunos problemas intermitentes de Timeout sucedían durante la creación, despliegue o destrucción de la Infraestructura y/o Aplicación. Ansible se quedaba ejecutando indefinidamente algunos de los procesos dando errores de ejecución.

Solución:

Luego de buscar distintas soluciones, los cambios necesarios para que esto no suceda han sido los siguientes.

Playbooks:

strategy: free

Explicación:

- Task execution is as fast as possible per batch as defined by `serial` (default all). Ansible will not wait for other hosts to finish the current task before queuing more tasks for other hosts. All hosts are still attempted for the current task, but it prevents blocking new tasks for hosts that have already finished.
- With the free strategy, unlike the default linear strategy, a host that is slow or stuck on a specific task won't hold up the rest of the hosts and tasks.

Referencia:

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/free_strategy.html

ansible.cfg:

[defaults]
host_key_checking = False
timeout=20

[persistent_connection]
network_cli_retries = 5
buffer_read_timeout = 2

Referencia:

https://docs.ansible.com/ansible/latest/collections/ansible/netcommon/network_cli_connection.html

https://docs.ansible.com/ansible/latest/reference_appendices/config.html#default-timeout

Explicación:

network_cli_retries: Number of attempts to connect to remote host.

buffer_read_timeout: Configures, in seconds, the amount of time to wait for the data to be read from Paramiko channel after the command prompt is matched. This timeout value ensures that command prompt matched is correct and there is no more data left to be received from remote host.

timeout: Default timeout for connection plugins.