

UNIR AZURE – TERRAFORM y ANSIBLE

Alumno: Diego Malvasi

INTRODUCCION.....	2
OBJETIVO.....	2
ESTRUCTURA DE LA SOLUCION.....	3
ARQUITECTURA.....	4
DESPLIEGUE AZURE	4
Pasos Despliegue	5
Configurando la aplicación	10
Ejemplo uso de la aplicación	13
Problemas encontrados.....	17
Azure	17
Local	19
DESPLIEGUE KSM - LOCAL.....	20
IMPLEMENACION LOCAL.....	20

INTRODUCCION

En este TP se desplegará una arquitectura en Azure y luego se instalará un cluster de Kubernetes en está utilizando las herramientas **Terraform** y **Ansible**.

Como parte del aprendizaje, se trató de utilizar diferentes métodos para ejecutar las mismas tareas, para analizar cómo funcionaba. En un nivel productivo, capaz no es tan claro la implementación, pero se realizó de esta manera poder entender mejor el funcionamiento.

A continuación, se detallan algunos ejemplos:

En terraform en el entorno local para la creación de los workers se utilizaron un loop de diferentes listas para los nombres, ips, macs, etc. Para el caso de Azure, se utilizó un loop de una lista del tipo **objeto**. Además, para la creación de **reglas de la nic**, se utilizó un **for_each**.

En ansible, se utilizaron variable a nivel global, a nivel de rol, a nivel de tarea y pasando variables al llamar un rol, etc. Además, se utilizó los defaults para las variables que no sean seteadas.

Otro ejemplo es para el caso de **dnf**, en algún caso se itero sobre una lista de app a instalar, otro caso se pasó el array de aplicaciones y también se utilizó un Sting separados por coma, para realizar un Split del mismo y luego iterarlo.

Para el caso del despliegue de la aplicación, a los nombres de los componentes se les dio el nombre **jenkins_nombreComponente** para poder entender con más claridad cómo funciona kubernetes evitando poner a todos por ejemplo **Jenkins** y no poder diferenciar cada componente.

En lo posible se intentó utilizar librerías de ansible para los comandos que no se encontró se utilizó **command**.

OBJETIVO

Se requiere desplegar una cluster de **Kubernete** vanilla en **Azure** , desplegando una aplicación que en este caso se seleccionó **Jenkins** con la siguiente arquitectura.

HOST	Requerido		
	Cant	CPU	Memoria
MASTER	1	4	8
NFS	1	2	4
WORKERS	2	2	4

Debido a la limitación de la cuenta de Azure de estudiante, para esta solución solo se desplegará un **worker** en lugar de los dos requeridos. Además, la capacidad de CPU del equipo master también se vera afectado pudiendo utilizar solamente 2 Cores en lugar de 4.

A continuación, se visualiza una tabla comparativa.

HOST	Requerido			Desplegado				
	Cant	CPU	Memoria	Cant	Nombre Tamaño	CPU	Memoria	IP Privada
MASTER	1	4	8	1	Standard D2 v3	2	8	10.0.1.30
NFS	1	2	4	1	Standard A2 v2	2	4	10.0.1.20
WORKERS	2	2	4	1	Standard A2 v2	2	4	10.0.1.50

ESTRUCTURA DE LA SOLUCION

Para el caso de **terraform** se utilizó el concepto de módulos para la creación de las Nic e Ips y otro para las reglas de seguridad.

Para el caso de los workers, se utilizaron loops para crear los diferentes workers. No se realizó así para todos otros los hosts ya que se entiende que son independientes conceptualmente, aunque sean parecidos.

La única diferencia, es que para el host **nfs** se agrego un segundo disco con el formato adecuado.

Para el caso de **ansible** se utilizaron roles los cuales se pensaron para hacer reutilizados para los diferentes tipos de **hosts**.

A continuación, se puede ver un esquemático de los roles y como se fueron ejecutando dependiendo del tipo de nodos. La estrella indica que ese rol se ejecuto para ese tipo de host.

ROLES	NFS	master	workers
Prepare_host	★	★	★
Prepare_nodes	★	★	★
install_master		★	
install_workers			★
install_nfs	★		
install_jenkins		★	

Estructura de directorio

A continuación, se hace una breve descripción del directorio de la solución. Se trata de plasmar los archivos o estructuras mas importantes. Muchas estructuras son repetidas, por ejemplo, en ansible tenemos la estructura de carpeta para los roles.

NOTA: No se detallan todos los archivos.

unir-azure-tp2

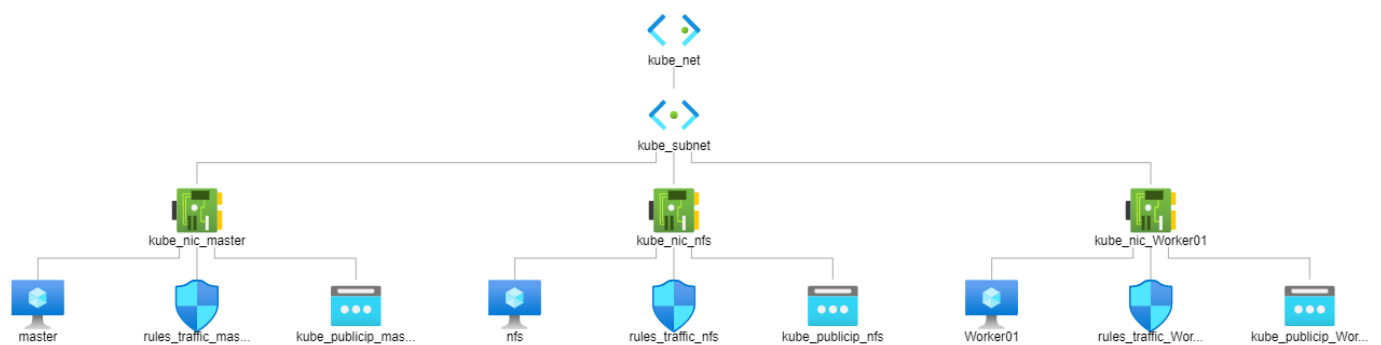
terraform	[Todos los archivos relacionados con Terraform]
modules	
az_network	[Creación de nic e ip]
az_security	[Creación de grupo de seguridad y reglas]
output_file.tf	[Crea los archivos Host e Public Ip NFS para ansible]
template	[Almacenas los templates de archivos para output_file.tf]
credenciales.tf	[Almacena las credenciales para conectarse a Azure]
vars_globals.tf	[Variable globales para modificar comunes, como so]
vars_workers.tf	[Variables referidas al workers, se puede modificar para crear mas de uno]
vm_workers.tf	[Crea la vm y recursos asociados del worker. Este esquema se repita para los otros tipos de host]
ansible	[Todos los archivos relacionados con Ansible]
hosts	[Donde se define el inventario. Este es generado por Terraform]
hosts.ok	[Se deja por si falla terraform, hay que renombrar y quitar ok]
group_vars	
kube.yaml	[Variable generales a todos los roles]
kube_ip_nfs.yaml	[Este archivo lo crea Terraform con la Ip publica del NFS. Se deja kube_ip_nfs.yaml.ok por si se quiere desplegar individualmente ya que terraform lo elimina si se realiza un destroy]
kube_workers.yaml	[Variable relacionadas con el worker. Este esquema se repite para cada tipo de rol]

```

├── Roles (Nomnbre Rol)
│   ├── Defaults      [ Se almacenan los defaults de variables si no se asigna valor]
│   ├── Tasks         [ Ejecuta las tareas del Role. Llama a main.tf y este a las otras]
│   ├── Files         [ Almacena los archivos que son copiados al host destino]
│   ├── Vars          [ Define variable a nivel de roles]
│   └── Roles (Nomnbre Rol)
│       └── install_workers [ Rol instala workers]
│           └── File
│               └── kube_token.conf [Archivo que genera el rol Master para ser utilizado en la
│                                       vinculación de los nodos]
├── deploy_kubernete_azure.sh [ Despliega toda la solución]
└── destroy_kubernete_azure.sh [ Elimina toda la solución]

```

ARQUITECTURA



DESPLIEGUE AZURE

A continuación, se detallan los pasos para el despliegue de la arquitectura en **Azure** utilizando **Terraform**, para luego desplegar un cluster de **Kubernetes** con **Ansible**. Para finalizar, se desplegará **Jenkins** en el cluster de **Kubernetes**.

Para realizar el despliegue, se requiere tener instalado Terraform, Ansible, Python y Git en la maquina donde se ejecutará el mismo.

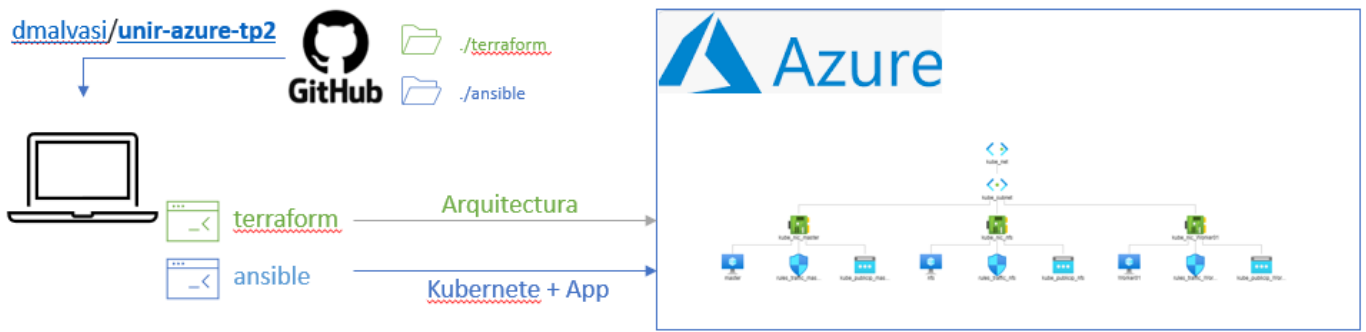
Por último, es necesario tener creada una clave publica **id_rsa.pub** en el directorio **~/ssh/**.

Las versiones de las aplicaciones deben ser compatibles con las siguientes versiones utilizada:

Componente	Versión
Terraform	terraform v1.1.7
Ansible	core 2.12.2
python	version 3.8.10
Git	version 2.25.1

NOTA: Las pruebas se ejecutaron en **wsl (Windos 10)** con una imagen de **Ubuntu 20.04.3 LTS**

Esquema de despliegue



Pasos Despliegue

El despliegue se realiza de manera automática, al ejecutar el comando despliega la arquitectura, el cluster de kubernetes, la aplicación Jenkins y la vinculación de los workers con el master.

Bajar reposito desde github

Se debe ejecutar el siguiente comando posicionado en la carpeta donde se quiere bajar el repositorio.

```
git clone https://github.com/dmalvasi/unir-azure-tp2.git
```

Seleccionar el despliegue a realizar

El repositorio posee 2 diferentes tipos de despliegue:

- Desplegar arquitectura en entorno local utilizando el virtualizado **KVM** (ver despliegue).
- Desplegar arquitectura en **Azure**, el cual nos enfocaremos ahora.

Desde el directorio donde se bajo el repositorio remoto, debemos acceder a la carpeta **unir-azure-tp2** y luego a la carpeta **azure** la cual contiene todos los archivos necesarios para el despliegue.

```
cd unir-azure-tp2/azure
```

Dentro de esta carpeta, tendremos los archivos necesarios para desplegar la arquitectura con **Terraform**, dentro de la carpeta **terraform** y también los archivos necesarios para el despliegue del cluster de **Kubernetes** y la aplicación **Jenkins** dentro de la carpeta **ansible**.

Validamos que exista la llave publica

Para validar la existencia de la llave, ejecutamos

```
vi ~/.ssh/id_rsa.pub
```

Deberíamos ver un texto similar al siguiente

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCAQC3eZtqvPPK2uQcX2MyhA8RAyJZMKDk6X7AFyjkEdh4zq1NP+jz/SYydSUBeEyT3UF2Q+s5Qke6XFUEz  
1yVwfqb8SghqP8g4PHqd5PiUHa8eQyglK07f3LxraL+jfxDoX0mMa18Tc+hAvPJInRrD/7rqefCsZMMfVzxiIFxUaDSh0oiNrSrtgY0tDcpApFc8XBu5  
U3+/rz8ICb18oq7pm36CxcRYh1Y2cyVyeCmfdeSJwaBZe+t0JqF1cz+iYaeSRsuxe+OP1uAB3CrsxXL4UUC0u3aw06eTnDtQBAWS9EJgz+nhxMQPUMHRK
```

Configurando la cuenta de Azure

Este paso lo podemos realizar de dos maneras diferentes.

Configuro las credenciales previo instalación

Antes de desplegar la solución, debemos ingresar las credenciales para conectarnos a **Azure**, en caso de querer introducirlas de manera manual cuando se ejecute el proceso, ir al paso siguiente **Desplegando la solución**.

Debemos ingresar al archivo **credenciales.tf** ubicado en el directorio **./unir-azure-tp2/azure/terraform** y descomentar las líneas **default** y reemplazar **xx** con los valores correspondientes.

```
UNIR_DEVOPS > CASO PRACTICO 02 > REPO FINAL > unir-azure-tp2 > azure > terraform > credenciales.tf > variable "tenant_id"
1  variable "subscription_id" {
2    type = string
3    description = "Id de subcripcion"
4    #default = "XX"
5  }
6
7  variable "client_id" {
8    type = string
9    description = "Id de cliente"
10   #default = "XX"
11 }
12
13 variable "client_secret" {
14   type = string
15   description = "Id clave secreta"
16   #default = "XX"
17 }
18
19 variable "tenant_id" {
20   type = string
21   description = "Id tenant"
22   #default = "XX"
23 }
24
```

```
UNIR_DEVOPS > CASO PRACTICO 02 > dm_final > unir-azure-tp2 > azure > terraform >
1  variable "subscription_id" {
2    type = string
3    description = "Id de subcripcion"
4    default = "69235dba-6010-4cc8-a983-XXXXXXXXXXXX"
5  }
6
7  variable "client_id" {
8    type = string
9    description = "Id de cliente"
10   default = "7427b6d8-df4f-4e10-XXXX-XXXXXXXXXXXX"
11 }
12
13 variable "client_secret" {
14   type = string
15   description = "Id clave secreta"
16   default = "9U57QbNBCi_K2Tp-XXXXXXXXXXXXXXXXXXXX"
17 }
18
19 variable "tenant_id" {
20   type = string
21   description = "Id tenant"
22   default = "899789dc-202f-44b4-XXXX-XXXXXXXXXXXX"
23 }
24
25
26
```

Ingresando manualmente las credenciales Azure

Si no se realizó el paso anterior y queremos que el sistema nos pida las credenciales, ejecutamos el proceso según paso **Desplegando la solución** y nos solicitara los datos de la siguiente manera.

```
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
>> terraform plan ...
var.client_id
Id de cliente

Enter a value: 
var.client_secret
Id clave secreta

Enter a value: 
var.subscription_id
Id de subscripción

Enter a value: 
var.tenant_id
Id tenant

Enter a value: 
```

El inconveniente de esta opción es que lo debemos ingresar dos veces, una cuando ejecutamos el **plan** y la otra cuando la aplicábamos.

Agregando Workers

La solución actual solo despliega un worker por limitación de la cuenta. En caso de requerir desplegar mas workers se debe agregar los parámetros en el archivo **vars_workers.tf** ubicado en **./unir-azure-tp2/azure/terraform**

Para esto, se debe agregar a la lista, los datos del **nombre** y **privateIp** por cada nodo.

```
variable "vm_size_workers" {
  type = string
  description = "Tamaño de la máquina virtual"
  #default = "Standard_D1_v2" # 3.5 GB, 1 CPU
  default = "Standard_A2_v2" # 3.5 GB, 1 CPU
}

variable "workers" {
  type = list(object({
    name = string
    privateIp = string
  }))
  default = [
    {
      name = "Worker01"
      privateIp = "10.0.1.50"
    },
    {
      name = "Worker02"
      privateIp = "10.0.1.55"
    }
  ]
}
```

Desplegando la solución

Para este paso, se optó ingresar las credenciales en el archivo **credenciales.tf**

Para desplegar la solución, debemos ejecutar el archivo **deploy_kubernetes_azure.sh**

```
sh deploy_kubernetes_azure.sh
```

Al ejecutar el comando, se comenzara a desplegar la arquitectura en **Azure**

```
hmalva@DESKTOP-0T1KKQR: /mnt/c/DATOS/UNIR_DEVOPS/CASO PRACTICO 02/dm_final/unir-azure-tp2/azure$ sh deploy_kubernetes_azure.sh
##### DESPLIEGUE ARQUITECTURA AZURE (TERRAFORM) #####
Directorio de ejecución Terraform:
>> /mnt/c/DATOS/UNIR_DEVOPS/CASO PRACTICO 02/dm_final/unir-azure-tp2/azure/terraform

>> terraform init ...
Initializing modules...

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/azurerm from the dependency lock file
- Reusing previous version of hashicorp/local from the dependency lock file
- Using previously-installed hashicorp/azurerm v2.46.1
- Using previously-installed hashicorp/local v2.2.2

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.
>> terraform plan ...
```

Luego de unos largos minutos, si todo salió bien nos mostrara que finalizo la instalación.

NOTA: En caso de este ejecutable, realiza el despliegue tanto de la arquitectura como así el cluster de **Kubernetes** y la aplicación **Jenkins**.

Accediendo a la configuración de la aplicación

Obtener la ip de acceso al master

Ya que no se cuenta con un **balanceo** en **Azure**, cada vez que desplegamos la aplicación se creara diferentes **ips** publicas para los respectivos nodos **Master**, **NFS** y **Workes**.

Primero debemos obtener la **ip** publica del **master**, esta la podemos obtener de dos maneras.

Visualizando el log del despliegue, una vez concluido el mismo nos mostrara las ips de los equipos creador.

```
#####
#
# DESPLIEGUE KUBERNETE + JENKINS [ TP UNIR ]
# MALVASI DIEGO
#
#####

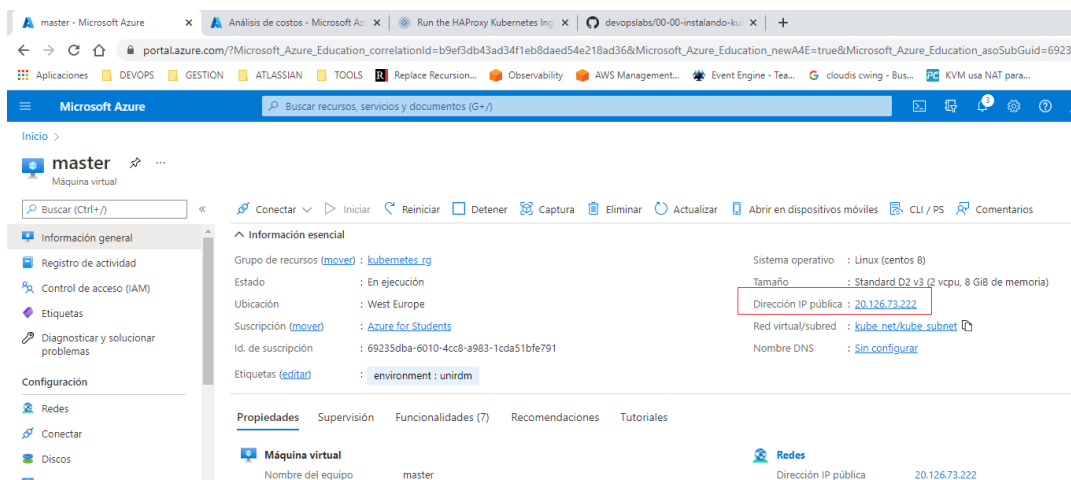
Master ip Azure
-----
'20.224.120.138'

NFS ip Azure
-----
'20.224.125.84'

Workers ips Azure
-----
20.224.123.191

dma1va@DESKTOP-6TJKNQR:/mnt/c/DATOS/UNIR_DEVOPS/CASO_PRACTICO_02/dm_final/unir-azure-tp2/azure$
```

La otra opción, es ingresare a **Azure** y obtener la **ip** publica del **master**.



Ingresamos al nodo master

Con la ip publica, desde el host donde se realizó la instalación (o el host con la llave privada que corresponde a la clave publica utilizada) accédenos a la consola de **host master**. El usuario es **ansible**, se puede modificar (ver modificar usuario de instalación)

```
ssh ansible@20.126.73.222
```

Donde 20.126.73.222 es la ip publica del **master**


```
SecureAnywhere
Live Support @ https://www.secureanycloud.com
AUTHORIZED USE ONLY.
Welcome! Support Email support@secureanycloud.com
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Mon Mar 14 15:12:12 2022 from 181.229.131.11
[ansible@master ~]$
```

Una vez que ingresamos correctamente, vamos a validar la instalación para luego si obtener el puerto que asigno el ingress controller.

Valido la correcta asociación de los workers al nodo

Debemos validar que todos los workers se hayan vinculado correctamente. Para esto ejecutamos:

```
sudo kubectl get nodes
```

Validamos en la salida que tanto el master como los workers se encuentren en **status** "Ready"

```
[ansible@master ~]$ sudo kubectl get nodes
NAME        STATUS   ROLES    AGE   VERSION
master      Ready    control-plane,master   24m   v1.23.4
worker01    Ready    <none>    4m39s v1.23.4
[ansible@master ~]$
```

Valido la correcta ejecución Jenkins

Ahora, vamos a validar que el **pod** de Jenkins se encuentre en estado **Running**

```
sudo kubectl get pods -n jenkins-namespace
```

```
[ansible@master ~]$ sudo kubectl get pods -n jenkins-namespace
NAME                                READY   STATUS    RESTARTS   AGE
jenkins-76458dc85d-9mpqb           1/1     Running   0           4m20s
[ansible@master ~]$
```

Obtenemos el puerto donde se publica la aplicacion

Ahora debemos obtener el puerto que nos asigno el **ingress controller** para acceder a nuestra aplicación.

Para esto, ejecutamos:

```
sudo kubectl get svc --namespace=haproxy-controller
```

Luego, validamos y **ANOTAMOS** el puerto que fue asignado, en este caso el **30563**

```
[root@master ~]# kubectl get svc --namespace=haproxy-controller
NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP  PORT(S)                                     AGE
haproxy-kubernetes-ingress         NodePort    10.111.20.139   <none>       80:30563/TCP,443:32347/TCP,1024:31033/TCP  15m
haproxy-kubernetes-ingress-default-backend ClusterIP    10.106.20.90   <none>       8080/TCP                                    15m
```

DATOS REQUERIDOS

Aquí colocamos una tabla para que almacene los datos que utilizaremos luego para la configuración de la aplicacion

DATO	VALOR EJEMPLO	VALOR EJECUCION
Ip publica master	20.126.73.222	< SU VALOR >
Puerto ingeress	30563	< SU VALOR >

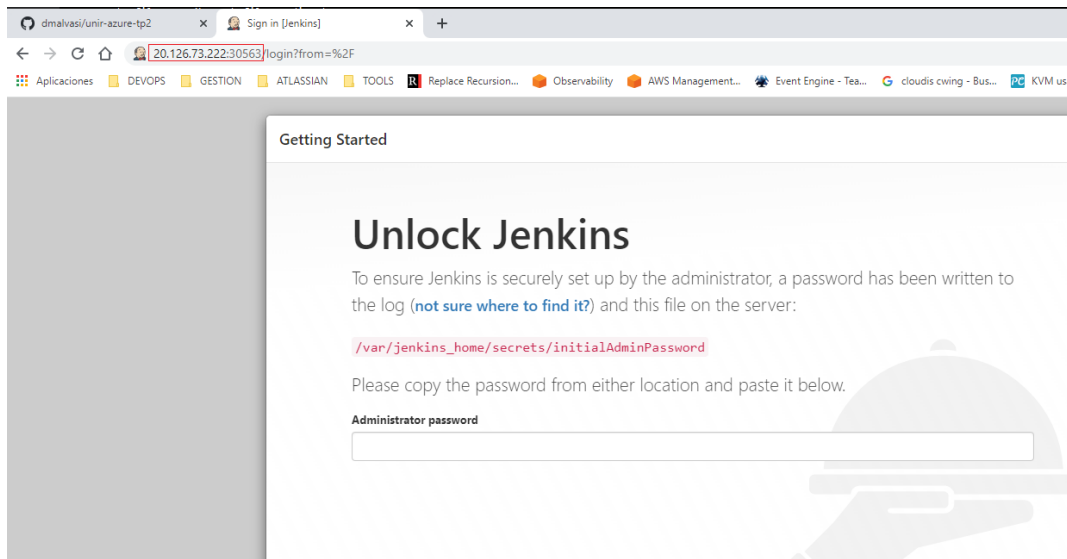
Configurando la aplicación

Ahora ya estamos en condiciones de ingresar a **Jenkins** y configurarla.

Para dar contexto al aplicativo **Jenkins**, decir que Jenkins es un servidor de automatización open source escrito en Java. Está basado en el proyecto Hudson y es, dependiendo de la visión, un fork del proyecto o simplemente un cambio de nombre. **“Wikipedia”**

Ingresando a la aplicación

Abriremos un navegador web (en este caso se utilizó chrome) desde cualquier pc con acceso a internet y utilizando la ip pública del master y el puerto del ingress ingresamos a Jenkins



Al ingresar por primera vez, nos solicitará una password de administrador. Veremos a continuación como conseguirla.

Obteniendo password de administrador

Para esto, primero debemos ingresar a la consola de comandos del nodo **master** (ver arriba).

Luego, debemos obtener el nombre del **pod** donde se está ejecutando **Jenkins**

```
sudo kubectl get pod -n jenkins-namespace
```

```
[root@master ~]# sudo kubectl get pod -n jenkins-namespace
NAME                                READY   STATUS    RESTARTS   AGE
jenkins-76458dc85d-z29md           1/1     Running   0           29m
[root@master ~]#
```

Luego, reemplazamos el nombre del **pod** en el siguiente comando para obtener la clave

Reemplazamos <NAME-POD> por el nombre del pod obtenido en el paso anterior.

```
sudo kubectl exec -n jenkins-namespace <NAME-POD> cat
/var/jenkins_home/secrets/initialAdminPassword
```

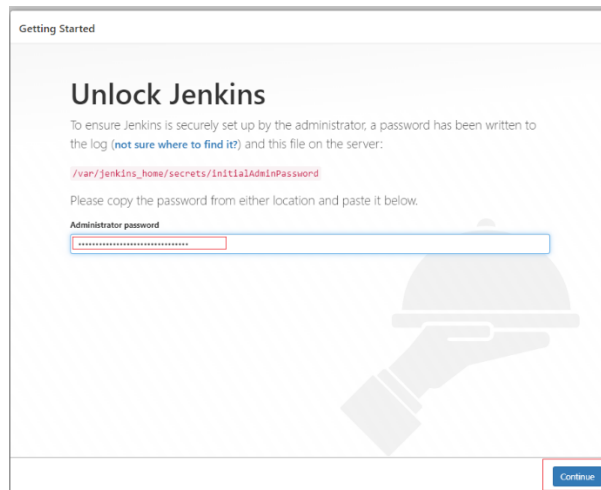
Ejemplo: `sudo kubectl exec -n jenkins-namespace jenkins-76458dc85d-z29md cat /var/jenkins_home/secrets/initialAdminPassword`

Una vez ejecutado, vamos a obtener la clave como se visualiza a continuación.

```
[root@master ~]# sudo kubectl exec -n jenkins-namespace jenkins-76458dc85d-z29md cat /var/jenkins_home/secrets/initialAdminPassword
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
287cbbd4af3f4acda4520507ff26f5aa
[root@master ~]#
```

Ingresando la clave de administrador

Con la clave obtenida en el paso anterior, la copiamos y la introducimos en **Administrator password** y hacemos clic en **continuar**.

The image shows the 'Unlock Jenkins' screen in the Jenkins web interface. At the top, it says 'Getting Started' and 'Unlock Jenkins'. Below this, a message states: 'To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server: /var/jenkins_home/secrets/initialAdminPassword'. It then asks the user to 'Please copy the password from either location and paste it below.' There is a text input field labeled 'Administrator password' with a red border and a red outline. A 'Continue' button is located at the bottom right of the form. A faint background image of a hand holding a tray is visible.

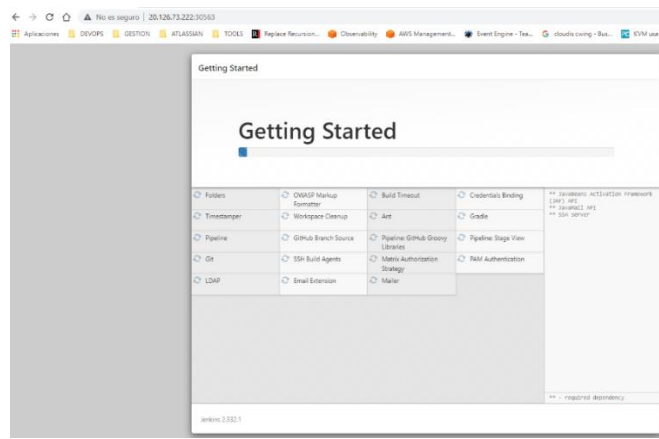
Luego debemos configurar los **plugins** que utilizara Jenkins. Para este caso, vamos a instalar los **plugins sugeridos**.

Instalando Plugins

Para instalar los plugins recomendados, hacemos clic en **Install suggested plugins**

The image shows the 'Bienvenido a Jenkins' (Welcome to Jenkins) screen in the Jenkins web interface. At the top, it says 'Getting Started' and 'Bienvenido a Jenkins'. Below this, a message states: 'Plugins extend Jenkins with additional features to support many different needs.' There are two main buttons: 'Install suggested plugins' (highlighted with a red border and red outline) and 'Select plugins to install'. The 'Install suggested plugins' button has a sub-message: 'Install plugins the Jenkins community finds most useful.' The 'Select plugins to install' button has a sub-message: 'Select and install plugins most suitable for your needs.' A faint background image of a hand holding a tray is visible.

Esto hará que se comiencen a instalar los plugins por defecto.



Una vez finalizada la instalación nos solicitara ingresar nuestros datos de usuario administrador.

Ingresando datos usuario administrador

Para continuar la configuración, el sistema nos solicita los datos para crear el usuario administrador.

Getting Started

Create First Admin User

Usuario:

Contraseña:

Confirma la contraseña:

Nombre completo:

Dirección de email:

Jenkins 2.332.1 Skip and continue as admin Save and Continue

Completamos los datos correspondientes como en el ejemplo a continuación.

Getting Started

Create First Admin User

Usuario:

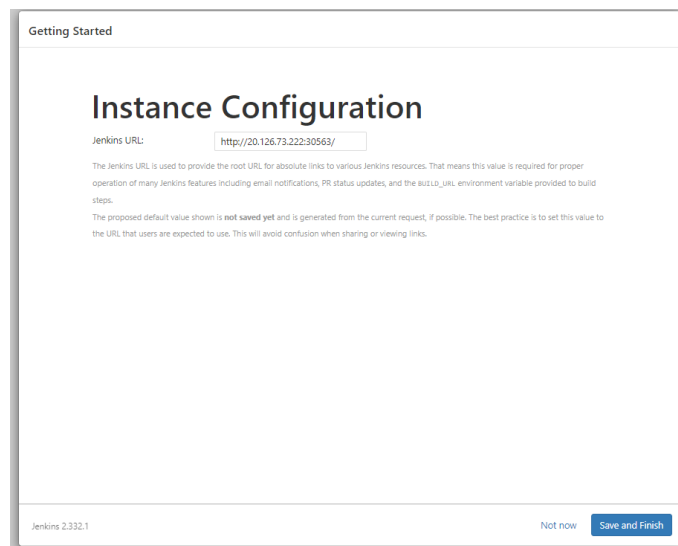
Contraseña:

Confirma la contraseña:

Nombre completo:

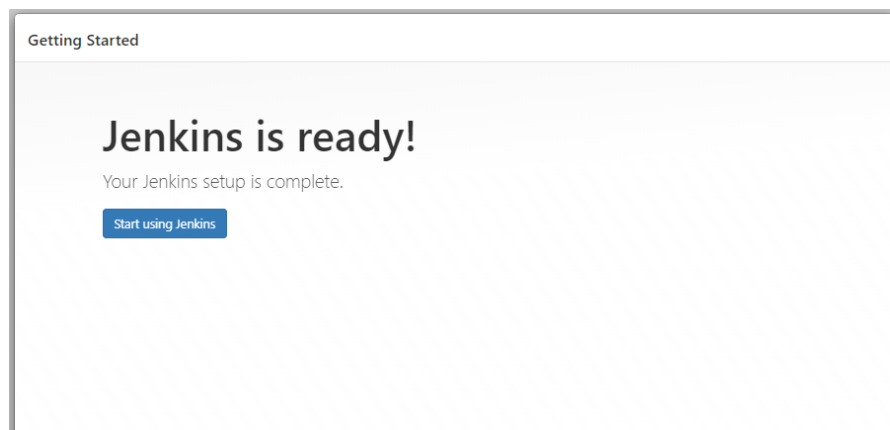
Dirección de email:

Por ultimo nos solicitara la url de la instancia de configuración, dejamos la por default.



NOTA: se requiere n balanceo para no perder la ip cada vez que se reinicie el nodo master. En su defecto dejar como ip fija la del master.

Listo, ahora ya podemos utilizar Jenkins

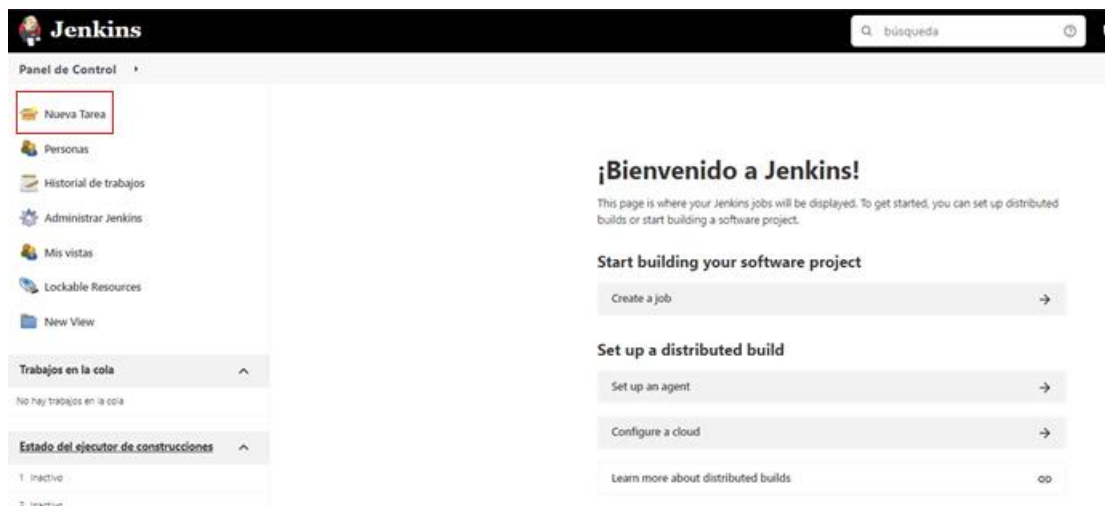


Ejemplo uso de la aplicación

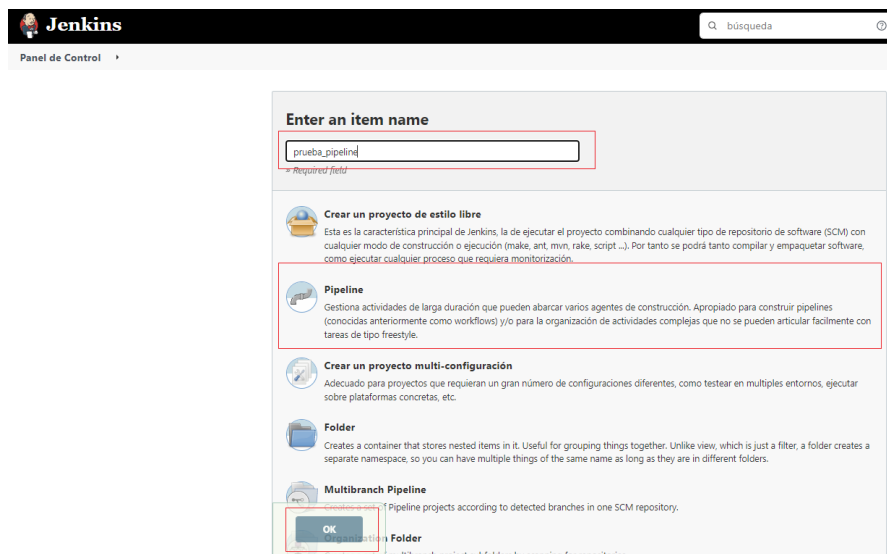
A continuación, veremos un pequeño ejemplo de cómo utilizar **Jenkins**

Creando primer Job

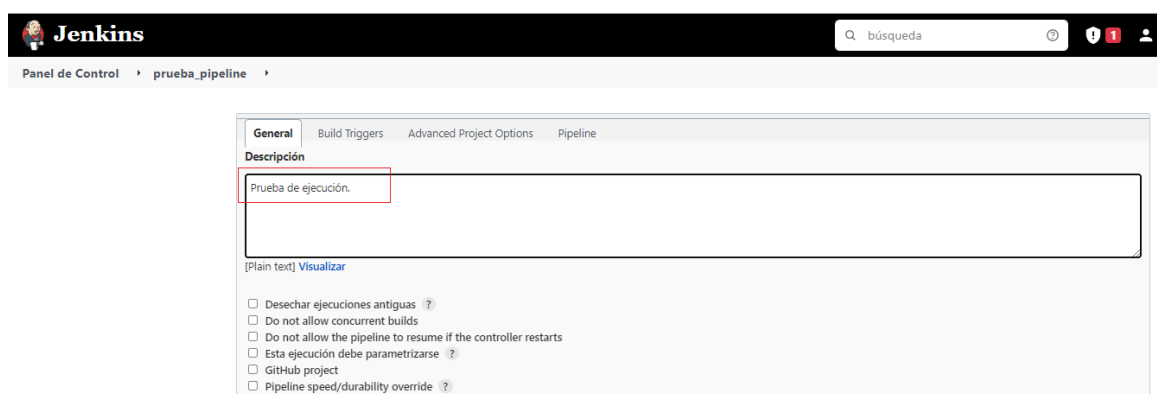
Para crear el primer job, debemos hacer clic en **Nueva Tarea**



Esto abrirá una nueva ventana para cargar el Job. Colocamos el nombre **prueba_pipeline** y hacemos clic en **Pipeline**



Agregamos una descripción



Y luego, abajo agregamos el script a ejecutar, en el campo **Script** y luego hacemos clic en **Guardar**

General Build Triggers **Advanced Project Options** Pipeline

Advanced Project Options

Pipeline

Definition

Pipeline script

Script ?

```

1 pipeline {
2   environment{
3     stageCompile = ':x:'
4     stageDependencies = ':x:'
5     stageTest = ':x:'
6     stageClient = ':x:'
7     stageArchive = ':x:'
8   }
9   agent any
10  stages {
11    stage('any'){
12      steps{
13        echo 'Obtengo directorio de ejecucion'
14        sh "pwd"
15        echo 'Obtengo el nombre del equipo (en este caso el contenedor)'
16        sh "echo $HOSTNAME"
17      }
18    }
19  }
20 }
21
22

```

try sample Pipeline...

☒ Use Groovy Sandbox ?

Guardar Apply

Script

```

pipeline {
  environment{
    stageCompile = ':x:'
    stageDependencies = ':x:'
    stageTest = ':x:'
    stageClient = ':x:'
    stageArchive = ':x:'
  }
  agent any
  stages {
    stage('any'){
      steps{
        echo 'Obtengo directorio de ejecucion'
        sh "pwd"
        echo 'Obtengo el nombre del equipo (en este caso el contenedor)'
        sh "echo $HOSTNAME"
      }
    }
  }
}

```

Una vez guardado, ejecutamos el job haciendo clic en **construir ahora**

Jenkins

Panel de Control prueba_pipeline

Back to Dashboard

Status

Changes

Construir ahora

Configurar

Borrar Pipeline

Full Stage View

Rename

Pipeline Syntax

Historia de tareas **Tendencia**

Filter builds...

#1 14 mar 2022 4:21

Atom feed Para Todos

Atom feed para los errores

Pipeline prueba_pipeline

Prueba de ejecución.

Recent Changes

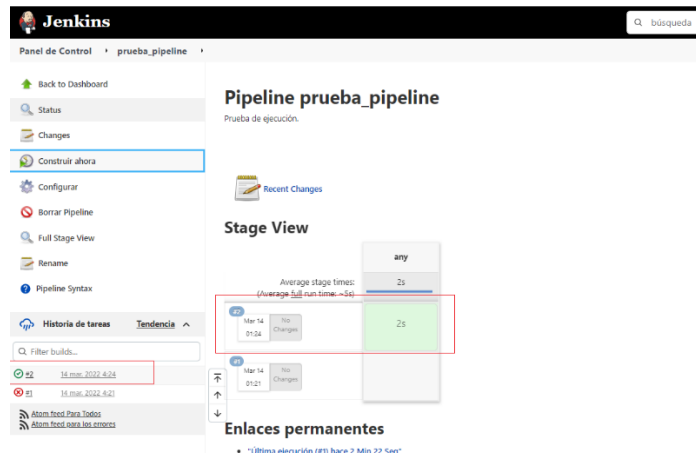
Stage View

No data available. This Pipeline has not yet run.

Enlaces permanentes

- Última ejecución (#1) hace 2 Min 22 Seg"
- Última ejecución fallida (#1) hace 2 Min 22 Seg"
- Última ejecución fallida (#1) hace 2 Min 22 Seg"
- Last completed build (#1) hace 2 Min 22 Seg"

El sistema comenzara a ejecutar el Job como se visualiza a continuación. Una vez finalizado, hacemos clic en **#2** (donde 2 es el numero de ejecución) para ver la salida de la ejecución.



La salida, podemos ver el directorio donde se ejecuto el Job y el nombre del equipo, que en este caso es el Contenedor dentro del pod de igual nombre.

The screenshot shows the Jenkins console output for build #2 of the pipeline 'prueba_pipeline'. The output is displayed in a 'Salida de consola' (Console Output) view. The output text shows the directory path and the container name.

```
Started by user Diego Malvasi
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/prueba_pipeline
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (any)
[Pipeline] echo
Obtengo directorio de ejecucion
[Pipeline] sh
+ pwd
/var/jenkins_home/workspace/prueba_pipeline
[Pipeline] echo
Obtengo el nombre del equipo (en este caso el contenedor)
[Pipeline] sh
+ echo jenkins-76458dc85d-z29md
jenkins-76458dc85d-z29md
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```


Problemas encontrados

Azure

- Al utilizar **Terreform modules** para reutilizar de una manera mejor la funcionalidad de terraform, cuando se desplegaban los equipos los mismos no se veían a nivel de red. Esto se debió a que cuando se creó el módulo **az_network** también se colocó para que cree la **net** y **sub-net** por lo que me creaba una para cada equipo. Se quitó la creación de la misma del módulo y se colocó como un archivo **networks.tf** en la raíz para que lo cree una sola vez.
 - Al querer utilizar los roles en **ansible**, no me ejecutaba las tareas de ese role y no daba ningún error. El problema era que la carpeta **tasks** se llamaba **task**.
Para poder detectar el error se utilizó la opción **strategy: debug** que permite para en diferentes partes del playbook para analizar las variables. Para poder detenerse en una acción, se debe agregar **debugger: on_skipped** por ejemplo. En este caso detiene la ejecución de una acción no es ejecutada debido a la condición que se colocó. Si se quiere para siempre en esa acción se utiliza **always**.
Con esto se detectó que nunca llamaba a las tareas.
 - También se utilizó la acción **debug** para ver las respuestas de las ejecuciones conjunto con la propiedad **register**, pero luego se reemplazó en muchos casos con la opción **-v o -vv** (depende nivel de debug) en la ejecución del playbook.
 - Inicialmente, se creó un **rol genérico**, por ejemplo, para **instalar paquetes**, pero se vio que no era lo más entendible y se interpretó que no era la función de un rol ya que se repetía mucho código y la instalación de los diferentes tipos de host poseían una alta cohesión.
Para resolver este inconveniente, se separó en diferentes grupos de acciones que se podían reutilizar. Para esto el código se separó en:
 - **prepare_host**: se encarga de las acciones comunes de todos los hosts. Como actualizar la zona horaria
 - **prepare_nodes**: se encarga de las acciones comunes de los diferentes tipos de host que componen el cluster de kubernetes (master/workers)
 - **install_master**: Se encarga de las acciones que solo se deben ejecutar en el master
 - **install_workers**: Se encarga de las acciones que solo se deben ejecutar en los workers
 - **install_nfs**: Se encarga de las acciones que solo se deben ejecutar en el host de nfs
 - **install_jenkins**: despliega la aplicación en el kubernetes.
- Así, de esta manera se puede realizar ejecuciones separadas que de la forma inicial que se realizó no se podía.
- En ansible se declaró una variable a nivel de rol dentro de la carpeta **vars** y el archivo se llamaba **vars_host.yaml** pero ansible no lo detectaba. Luego de investigar, para que lo ejecute se debe llamar **main.yaml** o el archivo debe estar dentro de una carpeta **main**
 - Problemas al resolver los nombres de api completos en ansible, se cambió a nombre corto **ansible.posix.firewalld** por **firewalld**
 - Cuando se ejecutó el playbook realizado para la instalación local, se detectó que algunas funciones no andaban, como ser **dnf** o **systemd**. Según los foros, esto se debía a la versión de ansible que se estaba ejecutando. Se actualizó la misma pero el inconveniente continuaba.
Se realizó una actualización del SO **dnf update** y se solucionó, pero el despliegue tardaba mucho. Se fue probando hasta que se verificó que solo actualizando **dnf** y **systemd** todo funcionaba.

- No se creaba el contenedor, no se podía detectar a que se debía el inconveniente en los logs del pod (ya que no estaba creado) o el describe de deploy del app. Mirando en los foros, se detectó que el Replicaset también posee un describe y se detecto que no se creaba por problemas con la configuración del nfs.
- Una vez desplegada la aplicación, no se podía acceder a la aplicación desde una pc remota. Esto se debió a que además de abrir los puertos en el host se debía abrir en la virtual con una regla de entrada.

→ Mover ▾ Eliminar Actualizar Enviar comentarios

^ Información esencial Vista JSON

Grupo de recursos (mover) : [kubernetes_rg](#) Reglas de seguridad pers... : 2 de entrada, 0 de salida

Ubicación : West Europe Asociado con : 0 subredes, 1 interfaces de red

Suscripción (mover) : [Azure for Students](#)

Id. de suscripción : 69235dba-6010-4cc8-a983-1cda51bfe791

Etiquetas (editar) : [environment : unirdm](#)

Puerto == todo Protocolo == todo Origen == todo Destino == todo Acción == todo

Prioridad ↑↓	Nombre ↑↓	Puerto ↑↓	Protocolo ↑↓	Origen ↑↓	Destino ↑↓	Acción ↑↓
Reglas de seguridad de entrada						
500	Http_master	31000-35000	Tcp	Cualquiera	Cualquiera	✓ Allow
1001	sshsecurity_master	22	Tcp	Cualquiera	Cualquiera	✓ Allow
65000	AllowVnetInBound	Cualquiera	Cualquiera	VirtualNetwork	VirtualNetwork	✓ Allow

- Problemas con NFS
 - Permisos, no se podía acceder con root a la creación, se detecto que faltaba permisos inicialmente .
 - **Problemas para acceder por IP privada (A Resolver)**
- Inicialmente se colocaba el listado de ips que se deben autorizar para el nfs. Pero luego para practicidad y hasta ver como levantar parámetros en ansible desde un archivo externo (por ejemplo config.json) se deja la opción para que acepte todas las ips.
- Para poder asociar los nodos al worker, inicialmente se intentó guardar el token de conexión al master en una variable, para que luego esta pueda acceder desde el despliegue de los workers. Pero al parecer al ser playbooks diferentes, no almacena la variable. Se realizo el guardado en un archivo.
- Inicialmente se implementó cri-o como gestor de contenedores, pero al no funcionar se intentó con doker que me fue más fácil para entender. Se deja ambas implementaciones.
- El comando update_cache: yes no funciona para azure .
- No se encontró api para algunas acciones, se utilizaron Comunity como **community.general.modprobe**
- Se presento el inconveniente el en archivo de despliegue del app, ya que para el pv hay que definir la IP publica de nfs.
Para solucionarlo, se agrego una palabra conocida, para luego reemplazarla con la ip del nfs antes de copiar el archivo al servidor para desplegarlos
- Al no conocer Linux y conexiones ssh, al principio me costó entender los motivos por los cuales no se conectaba ansible a los equipos. Luego se comprendía que en el host destino estaba la clave publica en la carpeta `~/.ssh/authorized` y en el local en `known_hosts`
- Cada vez que se ejecutaba el despliegue de la arquitectura tenía que eliminar las conexiones anteriores del archivo `~/.ssh/known_hosts`. Para evitar esto, se configuro en el inventario de ansible, el argumento para que no chequee el host en los hosts conocidos. El comando es **ansible_ssh_common_args='-o StrictHostKeyChecking=no'**

Local

- Compartir realizar un Bridge de la placa de red de la notebook.
- Por desconocimiento, inicialmente se averigüe como instalar Linux en forma silenciosa para que de esta manera poder generar las virtuales e instalarla.
- Se utilizó imagen **CentOS-8-GenericCloud-8.4.2105-20210603.0.x86_64.qcow2** y se debía actualizar la imagen y repos.

```
- sudo sed -i -e "s|mirrorlist=|#mirrorlist=|g" /etc/yum.repos.d/CentOS-*  
- sudo sed -i -e  
"s|#baseurl=http://mirror.centos.org|baseurl=https://vault.centos.org|g"  
/etc/yum.repos.d/CentOS-*  
- dnf clean all  
- dnf swap centos-linux-repos centos-stream-repos
```

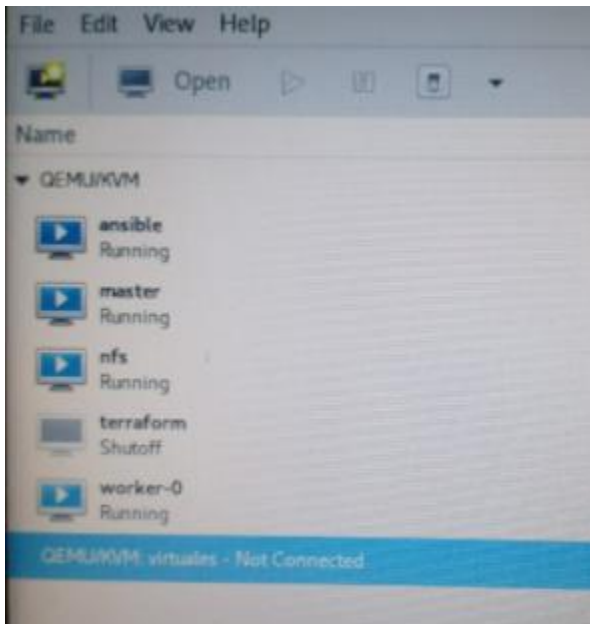
Luego de unos días, comenzó a fallar la página <https://vault.centos.org> por lo que debió cambiar por la imagen CentOS-8-GenericCloud-8.4.2105-20210603.0.x86_64.qcow2

- Como deploy inicial, se realizó por separado el deploy de cada tipo de máquina (nfs, node y workers) para que de esta manera poder tener más control ya que me fue muy difícil realizar el despliegue. Para ejecutar toda la arquitectura, se realizó un archivo **sh**, la desventaja es que hay que aceptar 3 veces el despliegue.
- Se intentó ejecutar Qemu en forma remota para crear la arquitectura. No se logró con el comando **qemu+ssh://ansible@192.168.1.39**. Para lanzar la ejecución de Terraform hay que conectarse a una máquina local donde está instalado el KVM.
- Cuando se utiliza cloud_init luego cuando inicia el host, tardaba un tiempo en estar disponible primero por la actualización del SO, se intentó que la instalación se realice antes de levantar el equipo, pero no se logró. La solución fue poner temporalmente al nombre del host **initializing-nombre_host** y luego cuando finalizaba la instalación se cambió al nombre del host solamente.
- Problemas con NFS
 - Problemas para acceder como root para prueba de creación y para la conexión del usuario utilizado para levantar, en mi caso **ansible**
- Problemas para desplegar Portainer, se llegó a la ventana de "Portainer Loading..." y no se pudo resolver. Se cambió aplicación a Jenkins

DESPLIEGUE KSM - LOCAL

Inicialmente se intento desplegar la arquitectura en forma local ya que provengo del área de análisis funcional/Administración del aplicativo Remedy y no estoy familiarizado con el SO Linux y con ninguna herramienta del área de Devops.

Por lo cual se instaló sobre una notebook un sistema **CentOS 8** con el sistema de visualización de KVM.



Para este caso no se automatizo el despliegue y gran parte no modularizo como por ejemplo Terraform.

IMPLEMENACION LOCAL

Para este caso, se utilizó cloud-init para la configuración inicial.

Componentes utilizados para el despliegue AWS

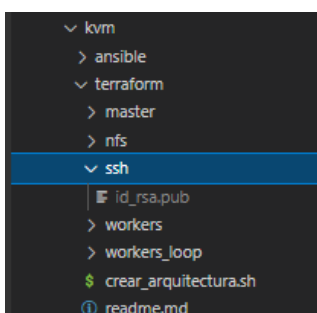
Componente	Versión
WSL (Windows 10)	Ubuntu 20.04.3 LTS
Terraform	terraform v1.1.7
Ansible	core 2.9.27sh
python	version 3.8.10

Ingresamos a la carpeta kvm terraform

Ejecutamos el comando:

```
cd /kvm/terraform
```

Colocamos la clave publica si la maquina de ansible se encuentra en otro equipo. Es decir, la implementación copiará la llave publica del equipo donde esta instalado **KVM** y la que definamos en **kvm/terraform/ssh/id_rsa.pub**



Luego, para la creación de la arquitectura ejecutamos

```
sh crear_arquitectura.sh
```

```
+ network interface {
+   + addresses      = [
+       + "192.168.122.50",
+     ]
+   + hostname      = (known after apply)
+   + mac           = "52:54:00:60:40:a5"
+   + network_id    = (known after apply)
+   + network_name  = "default"
+ }
}

# libvirt volume.volumen-qcow2 will be created
+ resource "libvirt_volume" "volumen-qcow2" {
+   + format = "qcow2"
+   + id     = (known after apply)
+   + name   = "disk-master.qcow2"
+   + pool   = "images"
+   + size   = (known after apply)
+   + source = "/home/libvirt/images/CentOS-8-GenericCloud-8.4.2105-20210603.0.x86_64.qcow2"
+ }

Plan: 3 to add, 0 to change, 0 to destroy.

Warning: Interpolation-only expressions are deprecated

on libvirt.tf line 42, in resource "libvirt_domain" "domain-distro":
42:   name = "${var.hostName}"

Terraform 0.11 and earlier required all non-constant expressions to be
provided via interpolation syntax, but this pattern is now deprecated. To
silence this warning, remove the "${" sequence from the start and the ")"
sequence from the end of this expression, leaving just the inner expression.

Template interpolation syntax is still used to construct strings from
expressions when the template includes multiple interpolation sequences or a
mixture of literal strings and interpolations. This deprecation applies only
to templates that consist entirely of a single interpolation sequence.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: █
```

Esto se repite 3 veces, una por cada tipo de

Una vez creada las virtuales, hay que esperar para conectarse por ssh hasta que termine la actualización. Si ingresamos vi virtualizador, veremos que el nombre del host es **initializing-nombre**. Hasta que no figure solo el nombre del host, el equipo no esta listo para su utilización.

Preinstalación de Cluster Kuberne y Aplicacion

Antes de comenzar la instalación, debemos agregar al archivo **hosts** del equipo la relación de **ip-nombre** para que ansible puede conocer al host ya que no se cuenta con un dns

hosts

```
192.168.122.11 worker0 worker01.dm.ar
192.168.122.22 worker1 worker02.dm.ar
192.168.122.30 worker3 worker03.dm.ar
192.168.122.50 master master.dm.ar
192.168.122.60 nfs nfs.dm.ar
192.168.122.70 docker docker.dm.ar
192.168.122.80 ansible ansible.dm.ar
```

Instalación de Cluster Kuberne y Aplicacion

Una vez que finalizo de actualizar los host (**es decir el nombre cambio de initializing-nombre a nombre final**) podemos ejecutar la instalación de ansible.

Para esto vamos a kvm\ansible\ y ejecutamos **exec_kubernete.sh**

```
cd /kvm/ansible
```

Ejecutamos el despliegue.

```
exec_kubernetes.sh
```

Asociar el nodo con el master

Una vez que se finalizó la instalación de **ansible**, debemos ingresar al nodo **master** y ejecutar el siguiente comando para obtener el token.

```
kubeadm token create --print-join-command
```

Este es un ejemplo de respuesta, debemos guardar este token.

```
kubeadm join 192.168.122.50:6443 --token tkwzb7.utyppppz51kuzuh9 --discovery-token-ca-cert-hash sha256:2414729be9a524d9cf1d22b5f7151a4c5da46aa1d472b74a153e54ba13ff80be
```

```
[root@worker0 ~]# kubeadm join 192.168.122.50:6443 --token tkwzb7.utyppppz51kuzuh9 --discovery-token-ca-cert-hash sha256:2414729be9a524d9cf1d22b5f7151a4c5da46aa1d472b74a153e54ba13ff80be
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserer and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

[root@worker0 ~]#
```

Ejecutamos comando en Workers

Ahora debemos ejecutar el comando obtenido en el paso anterior en cada worker

```
kubeadm join 192.168.122.50:6443 --token tkwzb7.utyppppz51kuzuh9 --discovery-token-ca-cert-hash sha256:2414729be9a524d9cf1d22b5f7151a4c5da46aa1d472b74a153e54ba13ff80be
```

Validamos los nodos

Una vez ejecutado, validamos que se hayan asociados los nodos correctamente.

```
kubectl get nodes
```

```
[root@master ~]# kubectl get nodes
NAME      STATUS    ROLES          AGE   VERSION
master    Ready     control-plane,master   36m   v1.23.4
worker0   NotReady  <none>          41s   v1.23.4
[root@master ~]#
```