Miniproyecto 2: Implementacion de Cluster Kubernetes en Azure y despliegue de aplicaciones usando Azure Kubernetes Service (AKS)

1st Jhon Alexander Quejada Urrutia
Ingeniero Mecatronico
Universidad Autonoma de Occidente
Cali, Colombia
jhon.quejada@uao.edu.co

1st Juan Camilo Valencia Solarte
Ingeniero Mecatronico
Universidad Autonoma de Occidente
Cali, Colombia
juan_c.valencia_s@uao.edu.co

***Abstract*—This document presents how a Kubernetes cluster is implemented in Microsoft's azure, from the moment of assembling each of the steps so that the implementation is successful, in summary, the assembly of the clustering based on the free student subscription and its limitations of resources, the way a container is prepared to be able to use a tool offered by the Microsoft application such as ACR to store in the cloud the necessary containers to launch and use our applications, in the same way, distribute services through the Kubernetes architecture, and finally the way we can monitor our resources and keep track of them**

I. Introducción

Hoy en día el uso de los contenedores ha facilitado la prestación de servicios debido a sus arquitectura donde ya se conoce evita mantener los procesos de una maquina virtual para levantar un servicio, donde se necesita una maquina física, una sistema operativo un hipervisor y por ultimo la maquina virtual, a comparación con la arquitectura de contenedores solo se necesita el servicio de la maquina física el sistema host y Docker daemons teniendo asi los microservicios optimizando los recursos de la maquina física. Pero no todo es tan bueno, pus al incrementar las solicitudes al servicio los microservicios se van a necesitar en grandes cantidades se vuelve una tarea tediosa mantener contenedor a contenedor su servicio, todo esto con el fin de mantener la disponibilidad, es por eso que se busca la manera de tener algo que controle todos estos contenedores y mantenga el servicio, optimizando los recursos del proveedor por medio de la imagen de un contenedor el cual contenga algun tipo de servicio. En este documento se mostrara la manera de como se puede usar Azure para lograr el objetivo anteriormente planteado.

II. Marco teórico

A. Entorno virtualizado

En informática un entorno virtualizado significa en resumidas palabras un hardware que no es real. El significado en nuestro contexto de trabajo, donde la maquina física o real, ejecuta una o múltiples maquinas virtualizadas en sus diferentes tipos de virtualización que se vera mas adelante, de esta manera se ejecuta una maquina virtual o también conocida hipervisor, en la siguiente figura se muestra gráficamente lo anterior mencionado.

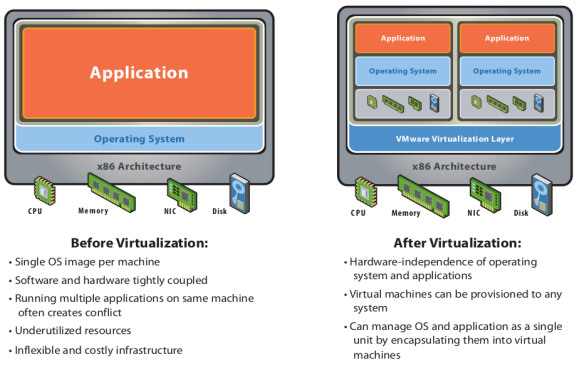

Fig. 1. Virtualización

Existen diferentes tipos de virtualización como se menciono anteriormente, entre ellas se encuentran:

1) Aplicaciones: Algunos ejemplos
   - Stream de aplicaciones
   - Escritorios remotos
2) Redes: Un ejemplo de ello puede ser Redes definidas por software o SDN donde algunos ejemplos pueden ser:
   - Nicira, o ahora conocida como VMware
   - Nuage Netwroks de Alcatel
3) Software basada en hipervisor: Es un método de virtualización sobre un mismo sistema para varias máquinas virtuales. En este tipo de virtualizaciones se suele usar un hipervisor, que es la capa intermedia entre el sistema real y las máquinas virtuales. Estos métodos sirven para virtualizar el hardware. Existen tres tipos:
   - Full virtualization
   - Para virtualization
   - Hardware assisted

4) Almacenamiento: De varios discos físicos se crea una capa virtual donde se crean volúmenes virtuales los cuales son asignados a servidores
5) Sistemas operativos/ Particionamiento: La virtualización del sistema operativo se realiza en el kernel, es decir, los administradores de tareas centrales de los sistemas operativos, este tipo de virtualización trae múltiples beneficios tales como:
   - Reduce el costo del hardware en masa, ya que las computadoras no requieren capacidades tan inmediatas.
   - Aumenta la seguridad porque todas las instancias virtuales se pueden supervisar y aislar.
   - Limita el tiempo que se destina a los servicios de TI, como las actualizaciones de software.

B. Hipervisor VMM

Un hipervisor es el encargado de hacer el monitoreo y control de la o las maquinas virtuales, por esto también es el encargado de la asignación de recursos a estas maquinas virtuales en tiempo real, estas suelen soportar diferentes tipos de sistemas operativos o o también conocidos de manera resumida por sus siglas en ingles OS, existen dos tipos:

1) Tipo 1 (Nativos): unas de las principales características de este tipo es que corre directamente sobre el hardware, y no necesita un sistema operativo host

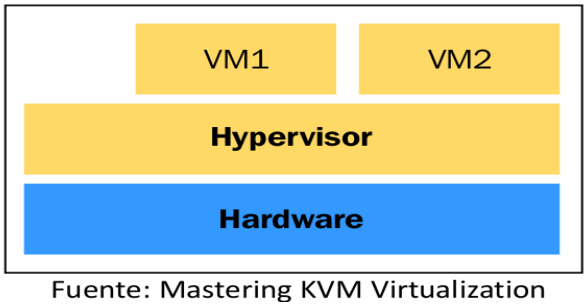

Fig. 2. Hipervisor tipo 1

2) Tipo 2: Este tipo de hipervisor reside sobre un sistema operativo (host) y depente de este para su operación.

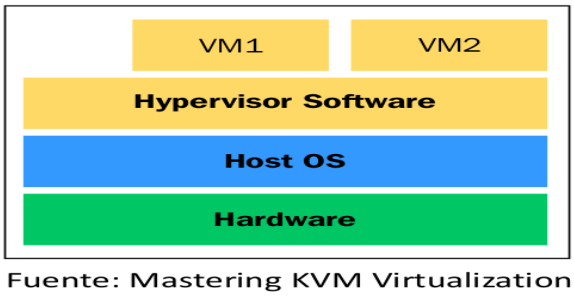

Fig. 3. Hipervisor tipo 2

C. Contenedores Linux

Los contenedores Linux son un conjunto de procesos separados del resto del sistema, todos los archivos que ejecutan provienen de imágenes diferentes, esto quiere decir que son portátiles y uniformes en todas las etapas, tales como desarrollo, prueba y producción

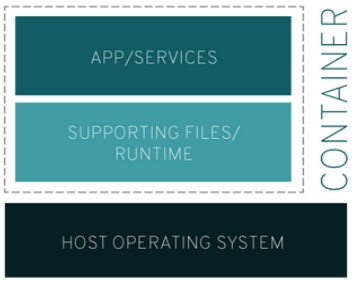

Fig. 4. Contenedor LXD

Cada contenedor tiene archivos que hacen que diferentes aplicaciones se soporten en el mismo para una tarea especifica. Los contenedores comparten el mismo kernel del sistema operativo y separan los procesos de las aplicaciones del resto del sistema.

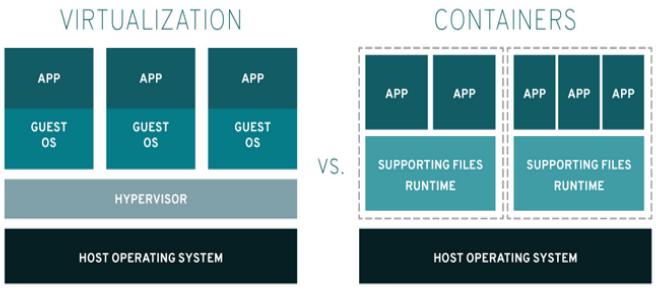

Fig. 5. Contenedor vs Virtualización

- LXD: Es un administrador de contenedores Linux, que se basa en imágenes prefabricadas, este ofrece gestión de redes en lo cabe, creación y configuración de puentes, túneles entre el host y el guest.
  Además ofrece la gestión de almacenamiento tales como backends, grupos de almacenamiento y volúmenes. Este es un claro ejemplo de virtualizacion de OS, por lo tanto se puede definir como un hipervisor pero no como los tradicionales (tipo 1 y 2), entonces este es definido como un tipo C o administrador de contenedores.
  A continuación la visualización del funcionamiento de LXD

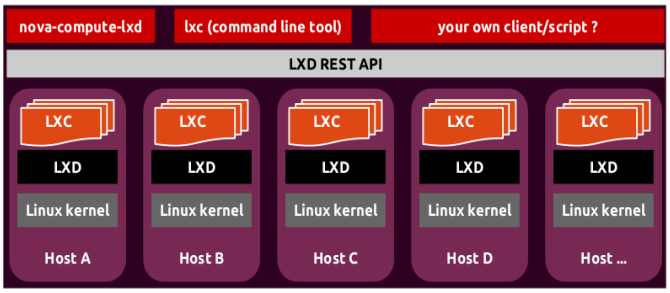

Fig. 6. Estructura LXD

D. Microsoft Azure

Microsoft Azure es una plataforma de productos y servicios en la nube enfocado en brindar soluciones,Desde el portal de Microsoft Azure se puede acceder a diferentes servicios de infraestructura y plataforma para contratar aquellos que sean necesarios para la empresa o proyecto. En apenas unos clics es posible disponer de Microsoft Azure funcionando y listo para trasladar el trabajo a la nube.

1) Cluster Azure: Es un conjunto de máquinas virtuales conectadas a la red, en las que se implementan y administran los microservicios. Un clúster de Service Fabric que se ejecuta en Azure es un recurso de Azure y se implementa mediante Azure Resource Manager.

2) ACR o Azure Container Registry: Es un registro privado para imágenes de contenedor. Un registro de contenedor privado permite compilar e implementar aplicaciones y código personalizado de forma segura.

E. Kubernetes

Kubernetes es una plataforma desarrollada por Google para manejar, escalar y realizar deploy de las aplicaciones empaquetadas en contenedores como Docker. Nos permite realizar la orquestación de diferentes contenedores pudiendo crear copias de uno o varios contenedores haciendo más fácil escalar nuestra aplicación, puede eliminar los contenedores que están fallando y crear nuevos para que nuestros servidores tengan la misma posibilidad de máquinas disponible. Su estructura se puede ver de la siguiente manera:

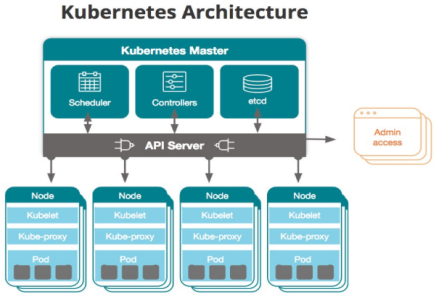

Fig. 7. Estructura de Kubernetes

1) Nodo: Los nodos de Kubernetes tienen dos tipos, los 'Workers" y los 'Manager', los 'workers' son los encargados de realizar e informar las tareas asignadas, mientras que el 'manager' es el encargado de recopilar y estar informado de todos los nodos, a continuación la una visualización gráfica de los dos tipos de nodo:

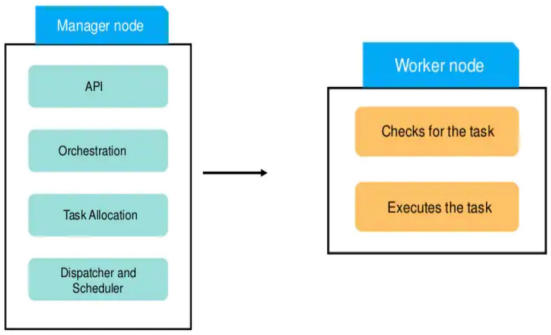

Fig. 8. Tipos de nodos

2) Pod: Los pods son los objetos más pequeños y básicos que se pueden implementar en Kubernetes. Un pod representa una instancia única de un proceso en ejecución en tu clúster. Los pods contienen uno o más contenedores, como los contenedores de Docker. Cuando un pod ejecuta varios contenedores, estos se administran como una sola entidad y comparten los recursos del pod. En general, ejecutar varios contenedores en un solo pod representa un caso práctico avanzado. A continuación una imagen que aclara la estructura de un pod.

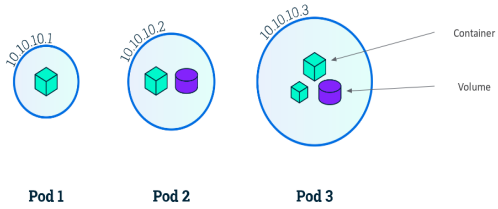

Fig. 9. Tipos de nodos

3) Servicio y labels: Un servicio es un conjunto de nodos que contiene sus pods con sus respectivos contenedores, estos pueden ser globales o replicados, cuando la prestación de servicios se requiere aumentar su capacidad se aumentan las replicas y se puede observar estos tres términos en la siguiente imagen:

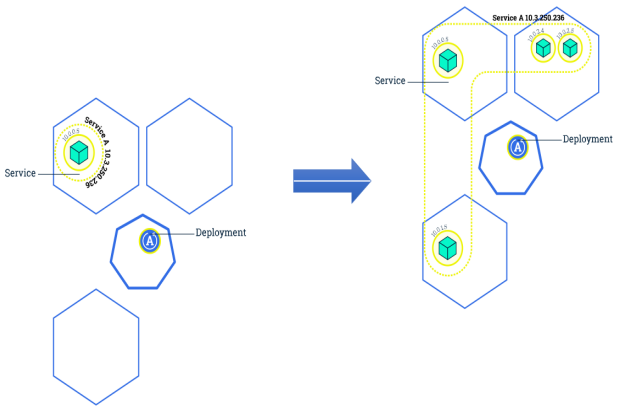

Fig. 10. Tipos de nodos

4) Despliegue: El despliegue de una aplicación se hace mediante un archivo con extensión '.yaml' el cual especifica las versiones de las imágenes de contenedores, esto permite hacer el despliegue de la aplicación a al cluster de Azure, una vez la aplicacion se ha desplegado en el Cluster, el servicio de Kubernetes permite administrar y controlar la aplicación desplegada

5) Rolling Updates: Esto permite a los desarrolladores actualizar las aplicaciones sin interrumpir el servicio esto se realiza actualizando gradualmente pods existentes con las nuevas funcionalidades, los nuevos pods son mapeados a nodos con recursos disponibles

III. IMPLEMENTACIÓN

Para el desarrollo de este miniproyecto se implementó un cluster de Kubernetes en la plataforma Azure, en donde el objetivo es poder desplegar aplicaciones web y de inteligencia artificial. Azure proporciona una gran capacidad de control de recursos para realizar nuestra implementación basada en Kubernetes, es importante tener en cuenta que toda la implementación se hizo en una cuenta para estudiantes. Consecuentemente podemos proceder a los apartados que se pedían en la implementación de este miniproyecto.

A. *Configuración del entorno*

Es necesario hacer una configuración de nuestro entorno para poder ejecutar los comando de azure **"az"** desde nuestro terminal. Para ello hacemos lo siguiente:

1. Instalamos los comando de az en el terminal:

sudo apt install az

Con esto, ya tenemos acceso completo a todos los comandos necesarios para poder realizar peticiones al Azure Portal desde nuestro terminal.

2. Instalar las dependencias del CLI de azure en el servidor:

curl -sL https://aka.ms/InstallAzureCLIDeb — sudo bash

3. Verificamos que se haya instalado correctamente ejecutando el siguiente comando en el terminal: **az**

Ejecutando el comando anterior, vamos a observar algo como esto:

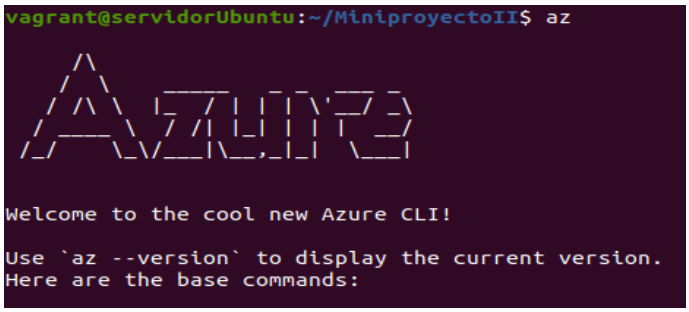

Fig. 11. Visualización de bienvenida a comando de Azure az

4. Instalamos kubectl en nuestro CLI con el siguiente comando:

az aks install-cli

Con estos pasos ya tenemos instalados todos los recursos necesarios para ejecutar comando de Azure en nuestro terminal.

B. *Instalación del ACR desde el terminal*

Para configurar nuestro repositorio de contenedores de Azure, podemos hacer los siguientes pasos desde el terminal:

1. Debemos asegurarnos de estar logeados en Azure desde el terminal, para eso usamos el siguiente comando:

az login

Ejecutando este comando vamos a obtener una instrucción de registro como la siguiente:

"To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code E7UECVMF3 to authenticate."

En donde debemos entrar en el link y registrar el código proporcionado para conectar nuestro terminal con el portal de azure.

2. Para crear recursos en Azure antes de crear un ACR, es necesario crear un grupo de recursos. En este se crearan los cluster, el ACR y demas recursos necesarios. Podemos crearlo con el siguiente comando:

az group create –name ***Definir un nombre de grupo*** *–location* ***Region***

Aquí, especificamos el nombre que va a llevar nuestro grupo de recursos y la región en dónde queremos crearlo.

3. Procedemos a crear el ACR (Azure Container Registry) de la siguiente manera:

az acr create –resource-group ***Nombre del resourcegroup*** *–name* ***Nombre del ACR*** *–sku Basic*

4. Verificamos que tenemos acceso al ACR, podemos conectarnos a este con el siguiente comando:

az acr login –name ***Nombre del acr***

Si todo sale correcto, veremos la siguiente instrucción - Login Succesed

5. Una vez configurado el ACR, podemos cargar nuestras imágenes desde la terminal al Azure Portal directamente a nuestro acr creado en nuestro de grupo de recursos.

5.1 Visualizamos las imágenes que tenemos descargadas con el comando docker images.

5.2 Una vez identificada la imagen que queremos subir, es necesario hacer un tag para agregarla a nuestro acr en el azure portal. Para ello usamos el siguiente comando:

docker tag ***nombre de la imagen login-server/nombre de la imagen:tag***

Si la imagen que vamos a subir ya tiene un TAG (v1 por ejemplo), se debe agregar ese TAG o versión.

5.3 Subimos la imagen tageada al portal de azure con el siguiente comando:

docker push ***nombre de la imagen Tageada***

6. Como el ACR que hemos creado es totalmente privado, tenemos que crear una entidad de servicios para permitir hacer conexiones al ACR de manera privada y poder hacer uso de las imágenes que hemos cargado en él. Es importante tener en cuenta que en este paso, vamos a obtener una serie de parámetros que vamos a tener que guardar en un bloc de notas:

- appleId
- displayName
- name
- password
- tenant

Ejecutemos el siguiente comando:

az ad sp create-for-rbac –skip-assignment

6.1 Configuramos el ACR:

Con el siguiente comando podemos obtener el id de nuestro ACR:

az acr show –resource-group ***Nombre del resourcegroup*** *–name* ***Nombre del acr*** *–query "id" –output tsv*

6.2 Para obtener el acceso corrector al cluster de aks que vayamos a crear para extraer las imagenes almacenadas en el ACR, debemos asignar el rol ACR pull, con el siguiente comando:

az role assignment create –assignee ***id de la aplicacion*** *–scope* ***id del ACR*** *–role acrpull*

C. Implementación de cluster Kubernetes en Azure

El cluster AKS (Azure Kubernetes Services) implementado se creó desde la plataforma de Azure y no desde el terminal, este tienen una capacidad de dos nodos únicamente debido a los limites de usabilidad de recursos que tiene nuestra cuenta para estudiantes. Observemos el paso a paso de como se creó:

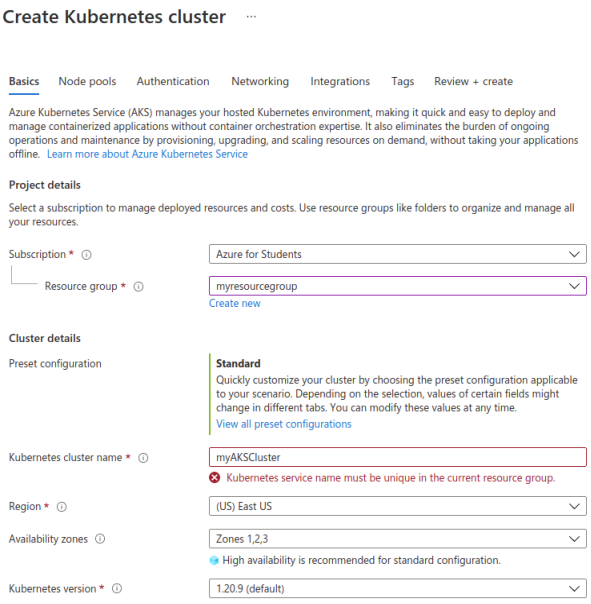

Fig. 12. Creación de cluster AKS

Podemos observar que hay varios campos requeridos para poder crear nuestro cluster AKS. En nuestro caso ya teníamos creado uno entonces por eso la plataforma de Portal Azure, hace la observación de que ya hay un cluster existente. Sin embargo, esto nos sirve de guía para tener en cuenta como se debe realizar el paso a paso para crear cluster mas adelante.

El resource group como su nombre lo indica, nos permite agrupas todos nuestros recursos que necesitemos para

realizar algun proceso especifico. Esto en cuanto a los detalles del proyecto que estamos creando. Ahora bien, tenemos los detalles de nuestro cluster AKS en donde especificamos el nombre que le vamos a asignar (**"myAKSCluster"**), la region de donde vamos a estar tomando los recursos que nos otorga la plataforma (**"(US) East US"**), las zonas disponibles en la region escogida y una version de kubernetes.

Otro apartado importante para configurar es el Primary node pool, aquí es importante tener en cuenta la capacidad de recursos que podemos usar en nuestra cuenta, en nuestro caso usamos un Node size **"Standar DS2 v2"**, que nos otorga una capacidad de (2 CPU y 7gb de memoria); se especificó que usara el método de escalado, el cual nos permite hacer un control mas efectivo de la capacidad de consumo de nuestros recursos y nos permite poder identificar si la cantidad de nodos que estamos otorgándole a nuestro cluster AKS estan permitidos dentro del rango de nuestra cuenta. Observe la configuración de la siguiente figura.

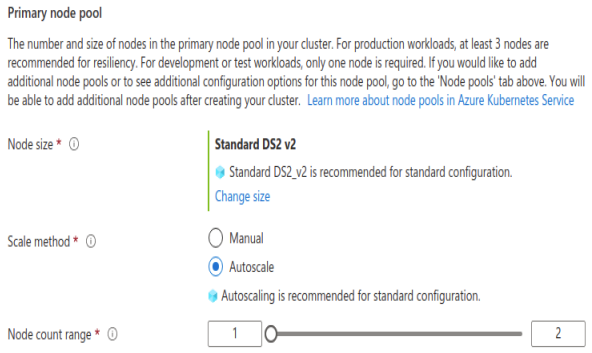

Fig. 13. Continuación creación de cluster AKS

D. Aplicación de clasificación de imágenes en Azure

Una de las implementaciones propuestas en la guía de miniproyecto es el despliegue de una aplicación que permita correr un algoritmo de deep learning para la clasificación de imágenes. En este apartado se hizo uso de una guía paso a paso de implementación proporcionada por la pagina de Kubermatic, en donde se hizo lo siguiente:

```
from gluoncv.model_zoo import get_model
import matplotlib.pyplot as plt
from mxnet import gluon, nd, image
from mxnet.gluon.data.vision import transforms
from gluoncv import utils
from PIL import Image
import io
import flask
app = flask.Flask(__name__)

@app.route("/predict",methods=["POST"])
def predict():
    if flask.request.method == "POST":
        if flask.request.files.get("img"):
            img = Image.open(io.BytesIO(flask.request.files["img"].read()))
            transform_fn = transforms.Compose([
            transforms.Resize(32),
            transforms.CenterCrop(32),
            transforms.ToTensor(),
            transforms.Normalize([0.4914, 0.4822, 0.4465], [0.2023, 0.1994, 0.2010
            img = transform_fn(nd.array(img))
            net = get_model('cifar_resnet20_v1', classes=10)
            net.load_parameters('net.params')
            pred = net(img.expand_dims(axis=0))
            class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                           'dog', 'frog', 'horse', 'ship', 'truck']
            ind = nd.argmax(pred, axis=1).astype('int')
            prediction = 'The input picture is classified as [%s], with probabilit
                         (class_names[ind.asscalar()], nd.softmax(pred)[0][ind].as
    return prediction

if __name__ == '__main__':
    app.run(host="0.0.0.0")
```

Fig. 14. Modelo de predicción para el algoritmo

Fuente: *https://www.kubermatic.com/blog/deploy-your-deep-learning-model-on-kubernetes-1/*

Para realizar el despliegue de nuestro modelo hacemos una contenerización de la aplicación haciendo uso de los contenedores Docker. En donde hacemos lo siguiente:

1. Una vez arrancada nuestra maquina virtual, creamos un directorio siguiendo los pasos de la fuente mencionada anteriormente, en este caso caso el nombre del directorio es **kubematic-dl**. Con el siguiente comando creamos ese directorio:

mkdir kubematic-dl

Para ingresar al repositorio creado ejecutamos **cd kubematic-dl**.

2. Una vez ingresamos al repositorio procedemos a crear los archivos necesarios para generar una imagen en donde estarán instalados todos los paquetes, requerimientos, librerías y módulos que va a necesitar nuestra aplicación:

2.1 Requirements.txt: En el archivo requirements.txt vamos a especificar todos los paquetes, librerias, modulos y/o dependencias que queremos instalar en nuestra imagen y que son necesarias para que nuestra aplicación funcione de la mejor manera.

```
flask
gluoncv
matplotlib
mxnet
requests
Pillow
```

Fig. 15. Archivo requirements

2.2 Dockerfile: En el archivo Dockerfile especificamos la imagen base que en este caso es la de python 3.6, las istrucciones para poder instalar los achivos que se encuentran en el requierements, y el servicio app.py que contiene el modelo de predicción, que va a estar corriendo para que el contenedor generado en Docker no se cierre.

```
FROM python:3.6
WORKDIR /app
COPY requirements.txt /app
RUN pip install --upgrade pip==21.3.1
RUN pip install -r ./requirements.txt
COPY app.py /app
CMD ["python", "app.py"]
```

Fig. 16. Archivo Dockerfile

2.3 app.py: En el archivo app-py se encuentra el modelo de predicción desarrollo por kubematic, con este pode-

mos hacer las pruebas del clasificador de imagenes, ya que el repositorio de kubermatic integra un modelo de entrenamiento basado en un dataset obtenido de CIFAR10 (Fuente: https://www.cs.toronto.edu/ kriz/cifar.html). Este contiene el mismo codigo que se encuentra en la fig. 14.

2.4 Creación de la imagen: Una vez organizado nuestro repositorio con todos los archivos necesarios, procedemos a crear nuestra imagen Docker con el siguiente comando:

docker build -t kubematic-dl:latest .

Teniendo ya nuestra imagen creada, podemos subirla al azure portal directamente en nuestro acr para usarla después en nuestro archivo .yaml que contiene el despliegue de nuestra aplicación. Podemostambiés generar un contenedor para hacer pruebas locales con el siguiente comando:

docker run -d -p 5000:5000 kubematic-dl

2.5 Creación de la app: Como ya tenemos nuestra imagen creada y subida a nuestro portal azure, en el grupo de recursos en donde tenemos el acr y el cluster, podemos hacer el despliegue de nuestra aplicación. Para ello debemos crear un archivo .yaml que va a contener la configuración de nuestra aplicación, con el siguiente comando:

vim deployment-kubermatic-dl.yaml

En el interior de ese archivo deployment-kubermatic-dl.yaml ingresamos lo siguiente:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubermatic-dl-deployment
spec:
  selector:
    matchLabels:
      app: kubermatic-dl
  replicas: 3
  template:
    metadata:
      labels:
        app: kubermatic-dl
    spec:
     containers:
     - name: kubermatic-dl
       image: kubermatic00/kubermatic-dl:latest
       imagePullPolicy: Always
       ports:
       - containerPort: 8080
```

Fig. 17. Archivo deployment-kubermatic.yaml

Por ultimo procedemos a crear nuestra aplicación con el comando apply de la siguiente manera:

kubectl apply -f deployment-kubematic-dl.yaml

Exponemos el puerto por donde queremos correr el aplicativo e indicamos el servicio que queremos correr, en este caso un load balancer. Ejecutando el siguiente comando:

kubectl expose deployment kubermatic-dl-deployment –type=LoadBalancer –port 80 –target-port 5000

Como ya tenemos nuestro aplicativo creado y en funcionamiento, podemos hacer las pruebas correspondientes usando el siguiente comando. Cabe resaltar que se descargaron unas cuantas imágenes de la web para poder realizar la prueba.

curl -X POST -F img=@dog.jpg http://52.152.200.212/predict

Los resultados para la clasificación de la imagen de un **perro** fueron los siguientes:

```
The input picture is classified as [dog], with probability 0.974
```

Fig. 18. Predicción de la red usando una imagen de un perro

Los resultados para la clasificación de la imagen de un **gato** fueron los siguientes:

```
The input picture is classified as [cat], with probability 0.845
```

Fig. 19. Predicción de la red usando una imagen de un gato

Los resultados para la clasificación de la imagen de un **avión** fueron los siguientes:

```
The input picture is classified as [airplane], with probability 0.996
```

Fig. 20. Predicción de la red usando una imagen de un avión

E. Aplicación de nuestro interés en Azure

Para la implementación de aplicativo de interés, hicimos uso de un repositorio que otorga microsoft, el cual se llama azure-vote-front, en donde el objetivo es interactuar en una votación acerca de la mascota de preferencia de cada usuario, entre las clases de elección se encuentran perros y gatos. En este aplicativo web, hay dos botones que nos permiten realizar la votación entre perro y gato y un botón de reset para volver a realizar la votación nuevamente. Esta implementación se llevó a cabo de la siguiente manera:

Hasta este punto de desarrollo, ya hemos abarcado a detalle como fue la configuración de nuestro acr, de nuestro cluster aks y como fue el proceso para subir las imágenes al portal de azure en el interior de nuestro acr, esto nos da una mayor flexibilidad para explicar la implementación de el aplicativo web azure-vote-front, ya que el proceso es relativamente similar a como se realizó con el aplicativo de clasificación de imágenes, lo que cambia en el proceso es lo siguiente:

Verifiquemos si estamos logeados con el acr en nuestro portal azure, desde el terminal con el siguiente comando:

az acr login –name ***nombre del acr***

Obtengamos las credenciales de nuestro cluster:

az aks get-credentials –name myAKSCluster –resource-group myresourcegroup

El paso diferenciador que hicimos con este aplicativo web fue clonar desde un repositorio de microsoft, la carpeta con las imágenes y todo el código Frontend preparado directamente a nuestro acr. Esto con el siguiente comando:

az acr import –name ***nombre del acr*** *–source mcr.microsoft.com/azuredocs/azure-vote-front:v2 –image azure-vote-front:v2*

Ahora, podemos ver las imágenes que tenemos cargadas en nuestro repositorio del portal azure con el siguiente comando:

az acr repository list –name ***nombre del acr*** *–output table*

```
vagrant@servidorUbuntu:~/MiniproyectoII$ az acr repository list --name acrjaqu --output table
Result
---------------
azure-vote-front
kubermatic-dl
```

Fig. 21. Visualización de imágenes del repositorio de Portal Azure desde el terminal

Procedemos a crear nuestra aplicación, para ello se creó una archivo en la raíz de nuestra maquina virtual llamado azure-vote-all-in-one-redis.yaml con el comando **vim**.

Nota: Por cuestiones de espacio y visualización en el documento, decidimos poner el contenido del archivo azure-vote-all-in-one-redis.yaml al final en el apartado de anexos. Sin embargo aquí dejamos la fuente de la pagina donde también se encuentra la guía y el contenido del archivo

Fuente aplicativo: https://purple.telstra.com/blog/how-to-deploy-docker-images-to-azure-kubernetes-services-aks

Una vez cargada nuestra aplicación solo nos queda realizar pruebas en nuestro navegador para verificar si realmente está funcionando correctamente, esto lo hacemos de la siguiente manera:

Con el siguiente comando, podemos obtener información acerca del servicio que estamos corriendo, para conocer la dirección IP externa de nuestro contenedores y poder correrlo en la web.

kubectl get services

Vamos a observar algo como esto:

```
vagrant@servidorUbuntu:~/MiniproyectoII$ kubectl get services
NAME                     TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
azure-vote-back          ClusterIP      10.0.9.89      <none>          6379/TCP       2d3h
azure-vote-front         LoadBalancer   10.0.66.241    20.85.178.124   80:32625/TCP   2d3h
kubermatic-dl-deployment LoadBalancer   10.0.179.180   52.152.200.212  80:31772/TCP   3h14m
kubernetes               ClusterIP      10.0.0.1       <none>          443/TCP        2d3h
```

Fig. 22. Visualización de servicios activos

Aquí podemos observar que la dirección IP externa de nuestro contenedor es la siguiente: 20.85.178.124 . Con esta dirección IP podemos ir al navegador web instalado en nuestro computador para realizar las pruebas, con lo que veremos algo así:

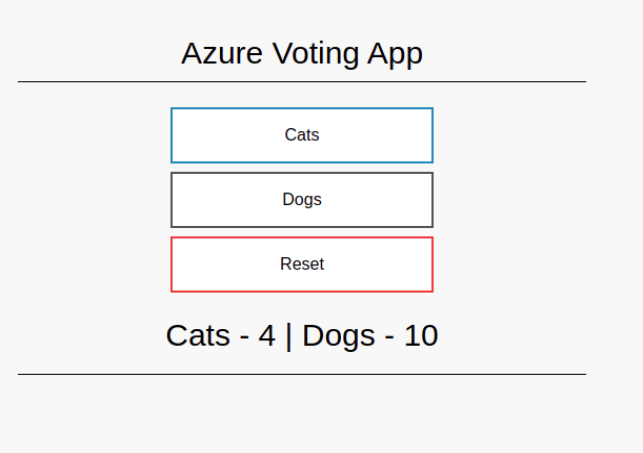

Fig. 23. Aplicación web Azure-vote-front

Nuestro servicio está corriendo de la mejor manera, y podemos observar algunas de las pruebas que se hicieron en el aplicativo web una vez montados.

IV. Conclusiones

- Estos servicios de nube como Azure, AWS entre otros facilita la administración de los microservicios, a pesar de las limitaciones de la licencia estudiantil, se puede aprender lo suficiente sobre este proveedor de servicios con un crédito el cual nos permite explorar muchas de sus beneficios.
- El uso de ACR nos permite tener las imágenes de los contenedores dentro de la nube del mismo proveedor de servicios, lo cual facilita el llamado de imágenes Docker, por otra parte esto permite tener los contenedores de forma privada en la nube.
- Una de las ventajas que encontramos, es que nuestras aplicaciones desplegadas en nuestro cluster se pueden acceder desde cualquier lugar, al principio considerábamos que estas se desplegaban en nuestra red local, no teníamos por entendido que estas se ejecutaban de manera publica en la red de internet por medio de la IP que nos suministraba el servicio de Kubernetes, hasta que por curiosidad intentamos correr la IP que nos brindaban

en el dispositivo diferente de nuestra red, y funciono perfectamente.

- Por ultimo se puede afirmar que la documentación que se encuentra de los diferentes proveedores de servicios de nube se pueden encontrar muy fácilmente en internet, específicamente en Ingles, pero por lo nuevo del servicio de Kubernetes muchas de los comandos que encontrábamos en los vídeos tenian comando que estaban desactualizados, y habia que encontrar la manera de adaptarlos a la versión actual, documentándose sobre los cambios sufridos a lo largo de las actualizaciones.

REFERENCES

[1] https://azure.microsoft.com/es-es/overview/what-is-azure/
[2] https://docs.microsoft.com/es-es/azure/service-fabric/service-fabric-cluster-creation-via-arm
[3] https://azure.microsoft.com/es-es/services/container-registry/overview
[4] https://cloud.google.com/kubernetes-engine/docs/concepts/pod?hl=es
[5] Just me and Opensource https://www.youtube.com/watch?v=LhF2HxqKPzk y https://www.youtube.com/watch?v=3UTvQ4ZNV1Yt=4s
[6] Oscar Mondragon https://www.youtube.com/watch?v=jQQS3uGyOGQt=962s
[7] Material de clase especializacion en Inteligencia Artificial Universidad Autónoma de Occidente Cali, Curso de Computación en la Nube

ANEXOS

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
      - name: azure-vote-back
        image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
        env:
        - name: ALLOW_EMPTY_PASSWORD
          value: "yes"
        ports:
        - containerPort: 6379
          name: redis
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
spec:
  ports:
  - port: 6379
  selector:
    app: azure-vote-back
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
      - name: azure-vote-back
        image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
        env:
        - name: ALLOW_EMPTY_PASSWORD
          value: "yes"
        ports:
        - containerPort: 6379
          name: redis
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
spec:
  ports:
  - port: 6379
  selector:
    app: azure-vote-back
```

Fig. 24. Archivo azure-vote-all-in-one-redis.yaml