E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Despliegue automático de infraestructura en la nube para el procesado de imágenes

Grado Universitario en
Ingeniería Informática

Trabajo Fin de Grado

David Casajús Calvet

Jesús Villadangos Alonso

Pamplona, 16-01-2023

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Contenido

1. Introducción 4
2. AWS vs Azure vs Google 5
2.1. Precios 5
2.2. Ingresos 6
2.3. Características 6
2.3.1. Servicio de Kubernetes 6
2.3.2. Regiones y zonas de disponibilidad 6
2.4. Opción elegida 7
3. Arquitectura del sistema 7
3.1. Aplicaciones contenerizadas 7
3.2. Docker 8
3.3. Kubernetes 8
3.3.1. Pod 8
3.3.2. Clúster 9
3.4. Infraestructura como código (IaC) 9
3.5. Hashicorp Terraform 9
3.5.1. Proveedores 10
3.5.2. Recursos 10
3.5.3. Variables 10
3.5.4. Módulos 10
3.5.5. Comandos útiles de Terraform 10
4. Requisitos del sistema 11
4.1. Requisitos funcionales 11
4.2. Requisitos no funcionales 11
5. Implementación 12
5.1. Crear contenedor de almacenamiento en S3 12
5.2. Instalación CLI AWS 12
5.3. Programa de copia de ficheros 12
5.4. Creando el contenedor de la aplicación 13
5.4.1. Definición de la imagen: Dockerfile 13
5.4.2. Construcción del contenedor 14
5.4.3. Subir contenedor al Elastic Container Registry 14
5.5. Definiendo la infraestructura de AWS con Terraform 15
5.5.1. Configuración general, config.tf 15
5.5.2. Definición de la red virtual privada en la nube (VPC), vpc.tf 16

5.5.3. Definición de los grupos de seguridad, security-groups.tf 16
5.5.4. Uniendo toda la configuración, eks-cluster.tfes 16
5.6. Creando la infraestructura de AWS con Terraform 17
5.7. Asignación del contenedor a la instancia creada 19
5.8. Obteniendo información de nuestro cluster 19
6. Eliminando la infraestructura 21
7. Errores surgidos durante el proceso 21
7.1. Problemas con la imagen de CentOS 21
7.2. Permisos AWS con Terraform 22
7.3. Error CrashLoopBackOff en los pods 22
8. Pruebas de carga 24
9. Pruebas de procesado de imágenes 24
10. Precios y gastos generados 25
10.1. AWS Elastic Compute Cloud (AWS EC2) 25
10.2. AWS Elastic Container Service for Kubernetes (AWS ECS) 26
10.3. Tabla de costes 26
10.4. Free tier de AWS 27
11. Gestión del proyecto 28
11.1. Primera fase 28
11.2. Segunda fase 28
11.3. Tercera fase 28
11.4. Cuarta fase 28
11.5. Control de versiones 28
11.6. Reuniones de seguimiento 28
12. Conclusiones y líneas futuras 29
12.1. Conclusiones técnicas 29
12.2. Conclusiones personales 29
12.3. Líneas futuras 29
13. Bibliografía 30

1. Introducción

Hoy en día la electricidad llega hasta los lugares más remotos del mundo, para ello son necesarios cientos de miles de postes eléctricos. Estos postes eléctricos permanecen a la intemperie 365 días al año, con lo que deben recibir un mantenimiento de forma periódica.

Al tratarse de estructuras de grandes envergaduras tener que ir revisándolos de uno en uno es un proceso poco eficiente, ya que además se deben recorrer grandes distancias.

En este caso el cliente es la empresa encargada del mantenimiento de los postes eléctricos de Pamplona, que quiere una mejora en el rendimiento del diagnóstico de los postes.

Para solucionar este problema, se propone utilizar drones para la inspección de los postes eléctricos. Mediante un dron se puede fotografiar un poste en cuestión de minutos para ser analizado más tarde mediante inteligencia artificial.

Una vez tomadas las fotos se procesan mediante inteligencia artificial usando redes neuronales para identificar las fotos en las que aparece un poste eléctrico. El procesado se realizará en la nube evitando que los operarios necesiten una infraestructura. Una vez procesadas, los operarios encargados de la revisión de los postes podrán acceder a un repositorio en la nube donde estarán las imágenes procesadas y podrán utilizarlas para evaluar los postes.

El procesado de imágenes se hará en AWS (Amazon Web Services), para gestionar la carga y rendimiento del servicio encargado de la detección de imágenes se utilizará Kubernetes, mediante Amazon EKS (Elastic Kubernetes Service). Mientras que las imágenes se almacenarán en Amazon S3 (Simple Storage Service).

El objetivo es definir la configuración necesaria para poder desplegar la infraestructura necesaria en AWS de forma automática, creando un entorno para el correcto procesado de las imágenes, utilizando infraestructura como código automatizando el aprovisionamiento de la infraestructura.

2. AWS vs Azure vs Google

El auge de las tecnologías cloud, ha hecho accesible para todo tipo de usuario disponer de servicios en poder de cómputo de clase mundial, desde multinacionales hasta dos jóvenes que desde sus casas desarrollan una nueva app, puedan acceder a infraestructura de primera a bajo costo.

En la actualidad existen 3 empresas que reinan en el mundo del Cloud Computing como son: Microsoft Azure, Google Cloud Platform y Amazon Web Service, cada una con una amplia gama de productos que cumplen diversas funciones. A estas tres se le podría añadir Alibaba Cloud la cual ha crecido mucho en los últimos años.

Uno de los grandes beneficios de los proveedores de servicios Cloud como Amazon Web Services, Google y Azure, son sus estrategias de precios competitivos y flexibles.

Al inicio se pagaba por uso mensual, sin embargo, las cosas están cambiando y girando hacia la tendencia de cobrar por segundo de potencia en cómputo.

Ilustración 1 Cuadrante Mágico 2020 para la infraestructura en la nube como servicio. Fuente Gartner

AWS es el líder indiscutible para servicios de infraestructura y plataformas en la nube, seguido por Microsoft y en tercer lugar Google. AWS se lleva el primer puesto gracias a su capacidad de ejecución y la integridad de la visión. Con esto AWS ya suma 10 años consecutivos siendo el líder de la nube.

2.1. Precios

Pay As You Go, hace que los precios se ajusten a las necesidades del proyecto, no es necesario comprar costosas soluciones de nube privada que se vuelven obsoletas en poco tiempo.

Tanto AWS como Azure como Google ofrecen un sinfín de opciones entre cientos de productos y servicios. Esto hace que comparar precios entre las diferentes plataformas sea algo muy difícil, ya que cada proveedor ofrece diferentes mecanismos de precios que cambian en función de la configuración utilizada.

El despliegue de una sola instancia en la nube ya viene con cientos de configuraciones posibles que harán que el precio cambie. Para decantarse por uno de los proveedores por el precio hay que tener especificado y muy claro que infraestructura y servicios se van a utilizar. Al tratarse de un ámbito que no he trabajado nunca es difícil decantarse en base a los precios. Aun así, AWS parece tener unos precios de entrada menos agresivos.

2.2. Ingresos

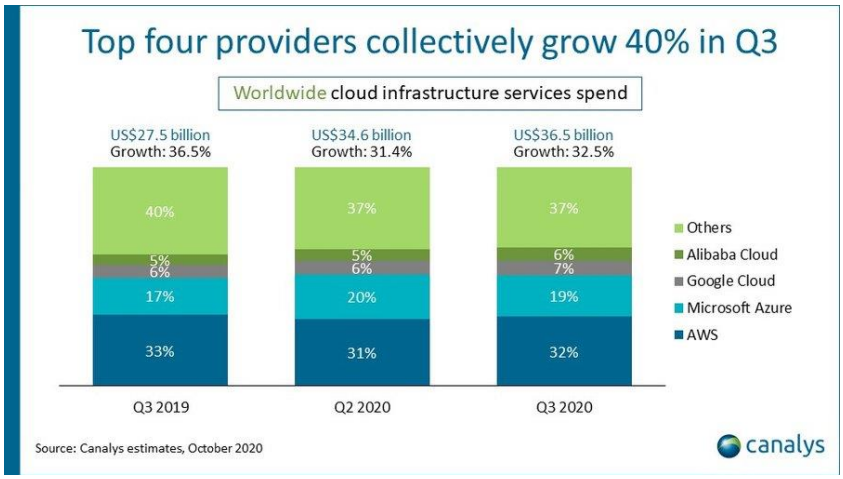


Ilustración 2 - Gasto mundial en servicios de infraestructura en la nube, tercer trimestre de 2020 Fuente: Canalys.

Como podemos observar AWS es el claro ganador respecto a Google y Azure con alrededor de un tercio de los ingresos, quedando Google muy por debajo de sus competidores con apenas un 6% de gasto mundial.

Los datos corresponden al año 2020, año de la pandemia del coronavirus, aunque la pandemia provocó una caída económica a nivel mundial, los servicios en la nube sufrieron un incremento de uso considerable. La pandemia exigió la necesidad del trabajo en remoto de tal manera que el uso del escritorio remoto aumentó un 94%. AWS y Azure fueron los principales proveedores de este servicio.

2.3. Características

En general las tres compañías ofrecen servicios muy similares, aunque los llamen de formas diferentes. Para el proyecto nos interesan los servicios relacionados con Kubernetes, contenedores y almacenamiento de datos. Por tanto, que una compañía ofrezca otros servicios es indiferente en este trabajo ya que no se van a requerir para conseguir los objetivos del trabajo.

2.3.1. Servicio de Kubernetes

Amazon Elastic Kubernetes Service (Amazon EKS) es un servicio de contenedores administrado para ejecutar y escalar aplicaciones Kubernetes en la nube o en las instalaciones.

Azure Kubernetes Service (AKS) ofrece la posibilidad desarrollar e implementar aplicaciones nativas de la nube, con canalizaciones de código a nube integradas y con límites de protección.

Google Kubernetes Engine ofrece dos modos de funcionamiento: Autopilot y estándar. El modo Autopilot es una solución automatizada y totalmente gestionada que se ocupa de toda la infraestructura del clúster para que no tengas que preocuparte por su configuración ni su monitorización y, al mismo tiempo, sigas ofreciendo una experiencia de Kubernetes completa.

2.3.2. Regiones y zonas de disponibilidad

Es importante saber dónde estarán los servidores que contengan nuestro servicio, aunque las tres compañías ofrecen servidores a lo largo de todo el mundo nos centraremos en los ofrecidos en el suroeste de Europa.

AWS actualmente dispone de 4 regiones, Londres, Paris, Irlanda y España. Casualmente la región de España fue puesta en marcha a finales de noviembre de 2022. Aunque actualmente no ofrece todos los servicios que si llegan a ofrecer las otras tres regiones. AWS confirmó que es cuestión de tiempo que la región española disponga de los mismos servicios.

Así mismo, Google cloud también dispone de regiones en España, Alemania e Inglaterra, ofreciendo todos sus servicios en las tres regiones.

Por desgracia Azure es la única que actualmente no tiene dispone de España como región y no hay ningún indicio que haga pensar que la habrá en un futuro cercano. Azure ofrece sus servicios en Francia y Reino Unido.

2.4. Opción elegida

Finalmente, la opción elegida es Amazon Web Services, las tres opciones ofrecen características similares en cuanto a funcionalidades y servicios, pero AWS es más adecuado en varios aspectos.

Por un lado, AWS ofrece unos precios menos agresivos respecto a Azure y Google. Al no haber usado nunca ninguno de estos servicios los precios de AWS son los más adecuados para la fase de aprendizaje.

Por otro lado, en Internet abundan los tutoriales y foros sobre servicios de AWS, lo cual facilita mucho la fase de aprendizaje, ya que las tres plataformas ofrecen amplia cantidad de servicios, con lo que al principio resultan complejos de entender.

AWS ofrece todo lo necesario para la aplicación que se plantea, desde AWS ECS (Elastic Container Service) donde alojar el contenedor con el código necesario para llevar a cabo la clasificación de las imágenes hasta AWS S3 (Simple Storage Service) donde almacenar todas las fotos para su posterior procesado en la nube.

AWS es la primera potencia en computación en la nube y por tanto es la opción más lógica.

3. Arquitectura del sistema

3.1. Aplicaciones contenerizadas

Contenerizar una aplicación es un método de virtualización que abarca a nivel de Sistema Operativo para implementar y ejecutar aplicaciones distribuidas sin lanzar una Máquina Virtual completa para cada aplicación.

Dentro de cada contendor se incluye toda la configuración y archivos necesarios para correr la aplicación contenida. Esto hace que la portabilidad entre distintas maquinas sea posible, ya que el contenedor puede ejecutarse en cualquier sistema sin necesidad de realizar cambios en el código.

Al ser una virtualización a nivel de Sistema Operativo aparecen inconvenientes a la hora de ejecutar contenedores en SO diferentes, por ejemplo, ejecutar un contenedor creado en Windows en Linux. Para solucionar este conflicto existe Docker.

3.2. Docker

Docker es una plataforma capaz de desarrollar, ejecutar y enviar aplicaciones. Docker permite separar las aplicaciones de sus infraestructuras.

Docker trata de crear contenedores ligeros y portables que puedan ejecutarse en cualquier máquina en la que haya Docker instalado, independientemente del sistema operativo, software o versiones que tenga instalados el usuario.

Docker es una herramienta diseñada para beneficiar tanto a desarrolladores, testers, como administradores de sistemas, en relación a las máquinas, a los entornos en sí donde se ejecutan las aplicaciones software, los procesos de despliegue, etc.

En el caso de los desarrolladores, el uso de Docker hace que puedan centrarse en desarrollar su código sin preocuparse de si dicho código funcionará en la máquina en la que se ejecutará.

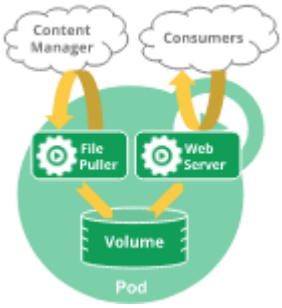
3.3. Kubernetes

Kubernetes es uno de los múltiples orquestadores de contenedores que existen. Es una plataforma de sistema distribuido de código libre para la automatización del despliegue, ajuste de escalado y manejo de aplicaciones en contenedores. Originalmente fue diseñada por Google, pero en 2014 fue liberada y donada a la Cloud Native Computing Foundation.

Las 4 características principales de Kubernetes son las siguientes:

- Balanceo de carga: permite balancear el tráfico de red entre los contenedores y la infraestructura.
- Despliegue automático: permite definir políticas de despliegue de forma que los contenedores se desplieguen automáticamente, sepan cómo prevenir caídas y fallos debido a posibles actualizaciones.
- Ajuste automático: usa, optimiza y adapta a la capacidad de cómputo disponible en cada momento, pudiendo definirse limites mínimos y máximos de cómputo.
- Autocuración: si uno de los contenedores falla, Kubernetes lo reiniciará hasta que funcione o tras varios intentos erróneos lo eliminará y creara uno nuevo para tratar de eliminar el fallo.

3.3.1. Pod



Un pod es un grupo de uno o más contenedores, donde el almacenamiento y la red son compartidos, así como especificaciones sobre como ejecutar los contenedores.

Los contenedores dentro de un pod pueden comunicarse e intercambiar datos mutuamente. Los pods son entidades relativamente efímeras, al igual que los contenedores. El ciclo de vida de un pod consiste en creación, asignación de identificador único (UID) y planificación en nodos donde permanecen hasta completar su tarea o eliminación del pod o nodo.

3.3.2. Clúster

Los clústeres son conjuntos de ordenadores o servidores que se gestionan juntos y participan juntos en la carga de trabajo. En un cluster puede haber diferentes nodos o servidores que dan servicio a diferentes aplicaciones individuales.

Los clústeres son los responsables de equilibrar la carga entre los distintos servidores que lo forman. Cuando instala una aplicación en un clúster, la aplicación se instala automáticamente en cada miembro del clúster. Puede configurar un clúster para proporcionar equilibrio de carga de trabajo con integración de servicios.

El proceso de tiempo de ejecución normal inicia automáticamente todos los componentes del servidor durante el proceso de inicio del servidor. Este proceso se aplica a todos los servidores, incluidos los servidores que forman parte de un clúster. Sin embargo, puede configurar servidores, incluidos los servidores que son miembros del clúster, de forma que no todos los componentes del servidor se inicien durante el proceso de arranque del servidor. Esta capacidad permite que el servidor consuma recursos según sea necesario, proporcionando así una ocupación más pequeña y manejable, y normalmente da como resultado una mejora del rendimiento.

3.4. Infraestructura como código (IaC)

En el pasado, la mayoría de las aplicaciones se diseñaban de forma que la infraestructura fuese a ser estática, es decir, no se contemplaba la posibilidad de aumentar o reducir las prestaciones (procesador, memoria, etc.) de la infraestructura. Esto cambió con la llegada de la nube, donde teóricamente el poder de cómputo sería ilimitado y se podría disponer del total casi de forma inmediata. Ese cambio ayudó a crear nuevas soluciones adaptadas a las nuevas capacidades, una de ellas fue Infraestructura como código.

La infraestructura como código permite definir la infraestructura en ficheros de configuración, al igual que los desarrolladores de software definen sus aplicaciones. Estos archivos de configuración son interpretados y transformados en infraestructura que se creará en la nube. Estos archivos de configuración permiten mantener diferentes versiones de la infraestructura permitiendo la evolución de esta o en caso de necesidad volver a un estado de infraestructura anterior.

Existen varias herramientas para IaC, entre ellas está Hashicorp Terraform (la cual usaré para el proyecto), AWS cloudFormation, Pulumi o Ansible entre otros tantos. El motivo de la elección de Terraform es la posibilidad de gestionar Kubernetes.

3.5. Hashicorp Terraform

Terraform es una herramienta de infraestructura como código que permite crear, cambiar y modificar versiones de recursos locales y en la nube de forma segura y eficiente. Define una estructura y sintaxis generales para el código y lo ejecuta para construir la infraestructura necesaria.

Terraform es declarativo, es decir, define cómo quiere que sea la infraestructura al final de su creación, no define los pasos que se han de llevar a cabo para conseguir el resultado final deseado.

La estructura de terraform se divide en cuatro elementos principales: proveedores, recursos, variables y módulos.

3.5.1. Proveedores

Aquí es donde se define qué proveedor de nube se está utilizando para la aplicación, en este caso será AWS, además se puede especificar qué versión debe utilizar, en caso de no especificarse utilizará la última versión disponible.

3.5.2. Recursos

Definición de la infraestructura necesaria como son: instancias, redes, almacenamiento, etc. Para cada recurso deben especificarse dos parámetros, el tipo de recurso y el id del recurso.

3.5.3. Variables

Variables del entorno necesarias, divididas en tres partes.

- Entrada: variables para solicitar información al usuario o runtime.
- Salida: variables que devuelven valores sobre las ejecuciones realizadas.
- Locales: variables que el desarrollador establece

3.5.4. Módulos

Usados para agrupar conjuntos de recursos y variables que son utilizados de forma conjunta en la arquitectura definida.

3.5.5. Comandos útiles de Terraform

- *terraform init*: comando que inicializa el proyecto de terraform, ejecutado una única vez por proyecto.
- *terraform plan*: comando que comprueba qué elementos se crearán, modificarán o eliminarán antes de ejecutar cualquier comando. Muestra por consola todos los elementos implicados, indicando cuales de ellos serán creados, modificados o eliminados.
- *terraform apply*: primero ejecuta terraform plan para mostrar los cambios que se van a realizar, una vez el usuario esté de acuerdo ejecutará los cambios en la nube.
- *terraform destroy*: elimina todos los recursos creados en la nube cuando ya no se necesita.

4. Requisitos del sistema

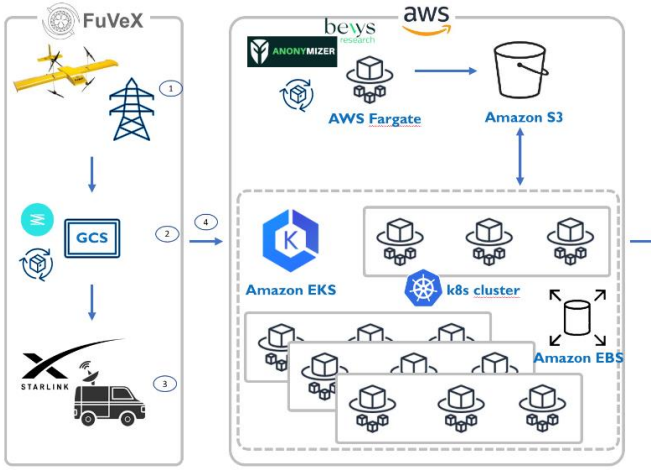


Ilustración 3 Diagrama general con la idea inicial del proyecto

Este es el diagrama principal del funcionamiento del sistema. Se pueden distinguir dos tipos de requisitos, los funcionales y los no funcionales. Por un lado, los requisitos funcionales definen qué debe hacer el sistema. Por otro lado, los requisitos no funcionales definen como debe ser el sistema.

El flujo básico es el siguiente, el operario sube saca las fotografías y las sube al almacenamiento S3, una vez subida se procesan en la nube identificando las fotografías válidas y finalmente se dejan esas fotografías en otro almacenamiento de S3 al que los operarios podrán acceder para revisar los postes eléctricos.

4.1. Requisitos funcionales

1. El sistema debe permitir la subida de fotografías a AWS S3.
2. El sistema debe realizar las operaciones en la nube.
3. El sistema debe desplegar nodos según la capacidad de cómputo necesaria en cada momento.
4. El sistema debe eliminar nodos según la capacidad de cómputo necesaria en cada momento.
5. El sistema debe identificar las fotografías que contengan torres eléctricas.
6. El sistema debe eliminar las fotografías donde no haya torres eléctricas.
7. El sistema debe guardar una copia de las fotografías donde haya una torre eléctrica.
8. El sistema debe permitir eliminar fotografías de forma manual al administrador.
9. El sistema debe permitir añadir fotografías de forma manual al administrador.

4.2. Requisitos no funcionales

1. La aplicación debe estar contenerizada.
2. El sistema debe funcionar en la nube de Amazon.
3. El sistema debe tener un servicio de Kubernetes.
4. El sistema debe tener un clúster.
5. El sistema debe obtener las imágenes a procesar desde AWS S3.
6. El sistema debe tener una web desde donde cargar las fotografías en AWS S3.

5. Implementación

5.1. Crear contenedor de almacenamiento en S3

Primero debemos crear un espacio de almacenamiento para las imágenes de forma que se pueda acceder a ellas más tarde. Tras haber creado una cuenta en AWS, desde el panel de S3· seleccionamos la opción de *Crear Bucket* e introducimos el nombre deseado. En el espacio de almacenamiento vamos a distinguir 3 carpetas de cara a evitar conflictos al acceder al mismo recurso de forma simultánea: Sin procesar, procesando, procesadas.

5.2. Instalación CLI AWS

Para instalar el CLI primero descargamos el ejecutable y lo ejecutamos, una vez instalado deberemos configurar ciertos datos para poder tener acceso a nuestra cuenta de AWS.

```
PS D:\Universidad\David\4ºCurso\TFG> aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]:
```

Código 1 Configuración de AWS CLI

Para obtener el Id y la clave de acceso debemos crear una a través del panel de usuario de AWS. En este caso utilizaremos como región *eu-west-2* correspondiente a Londres. Normalmente se usa *JSON* como formato de salida. Ahora ya podemos conectarnos a los servidores de AWS usando el CMD.

5.3. Programa de copia de ficheros

El lenguaje de programación utilizado es Python. Para conectarse al almacenamiento de S3 a través de Python vamos a utilizar la librería boto3. En el paso anterior hemos creado la configuración para acceder a AWS a través de la línea de comandos, pero nuestra aplicación está pensada para ejecutarse en pods del propio AWS, por tanto, debemos especificar en un fichero cuáles serán las credenciales que deben usar dichos pods. Para esto creamos un fichero llamado *credentials (los datos mostrados en la siguiente imagen son falsos).*

```
[default]
aws_access_key_id = ASDGFADFASDFASDFAS
aws_secret_access_key = lkfhasojdnfa4ksandflkj
region = eu-west-2
```

Código 2 Credenciales AWS

El siguiente bloque de código se conecta al contenedor S3, solicita los nombres de todos los recursos del contenedor y copia todas las imágenes que haya en la carpeta *sin procesar* en la carpeta *procesando.* Para que a continuación sean procesadas mediante la red neuronal y se decida qué imágenes se deben conservar y cuales eliminar.

```python
import boto3
s3 = boto3.resource('s3')
dest = s3.Bucket('prueba-tfg-david')
for my_bucket_object in dest.objects.all():
    if "sin procesar" in my_bucket_object.key and "." in my_bucket_object.key :
        aux = my_bucket_object.key.split("/")
        source = {'Bucket': 'prueba-tfg-david', 'Key': my_bucket_object.key}
        dest.copy(source, 'procesando/'+aux[len(aux)-1])

```

Código 3 Programa de copia de ficheros de en S3

5.4. Creando el contenedor de la aplicación

Una vez tenemos la aplicación funcionando es hora de crear la imagen y poder utilizarla en nuestros pods. Hay que recordar que para esto es necesario tener instalado Docker en nuestra máquina.

5.4.1. Definición de la imagen: Dockerfile

Un Dockerfile es un archivo o documento de texto simple que incluye una serie de instrucciones que se necesitan ejecutar de manera consecutiva para cumplir con los procesos necesarios para la creación de una nueva imagen.

A este conjunto de instrucciones se le conoce como línea de comandos y serán los encargados de indicar los pasos a seguir para el ensamblaje de una imagen en Docker, es decir, los elementos necesarios para el desarrollo de un contenedor en Docker.

Como base para la imagen se ha utilizado el sistema operativo CentOS, en el cual se crea un usuario con las credenciales necesarias para poder utilizar los comandos de AWS. Para poder correr el programa anterior se debe instalar Python y todas las librerías utilizadas en el programa. Por ello, se han instalado las siguientes librerías:

- Boto3 → Librería para la conexión con AWS
- Matplotlib → Librería para visualización de datos en Python
- Numpy → Librería para funciones matemáticas, como la generación de números aleatorios
- Tensorflow → Librería para aprendizaje automático
- Tensorflow_datasets → Colección de datos listos para usar

5.4.2. Construcción del contenedor

Una vez definido como debe ser nuestro contenedor, es hora de construirlo mediante el siguiente comando. Tras construir el contenedor ejecutaremos el contendor para poder probar nuestro programa, con el fin de verificar que toda la configuración realizada es correcta y el programa sigue funcionando.

```
1  docker build -t [nombreContenedor] [directorioCodigo]
2  Docker run -it [nombreContenedor] bash
```

Código 4 Creación y ejecución de un contenedor con Docker

5.4.3. Subir contenedor al Elastic Container Registry

Es hora de subir nuestro contenedor a AWS, ECR es un repositorio de imágenes. En primer lugar, debemos crear un repositorio, al crearlo nos devuelve cierta información por consola, es importante guardar el *repositoryUri* ya que nos hará falta para identificar nuestro repositorio.

```
PS D:> aws ecr create-repository --repository-name nombrerespositorio
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:eu-west-3:419101797974:repository/nombrerespositorio",
        "registryId": "5455454545454545",
        "repositoryName": "nombrerespositorio",
        "repositoryUri": "54545465465465.dkr.ecr.eu-west-3.amazonaws.com/nombrerespositorio",
        "createdAt": "2022-11-17T12:43:08+01:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": false
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

Código 5 Creación remota de repositorio

Ahora vamos a iniciar sesión en Docker para darle acceso a nuestro ambiente local para hacer push de la imagen a un repositorio, como es ECR. A continuación, debemos darle un nombre indicando además nuestro *repositoryUri* y finalmente la subimos mediante un push.

```
1  aws ecr get-login-password --region eu-west-2 | docker login --username AWS --password-stdin [repositoryUri]
2  docker tag [nombreImagen] [repositoryUri]/[nombreRepositorio]
3  docker push [repositoryUri]/[nombrerepositorio]
```

Código 6 Subida de imagen al repositorio remoto

Ya podemos ver nuestra imagen desde el panel del ECR.

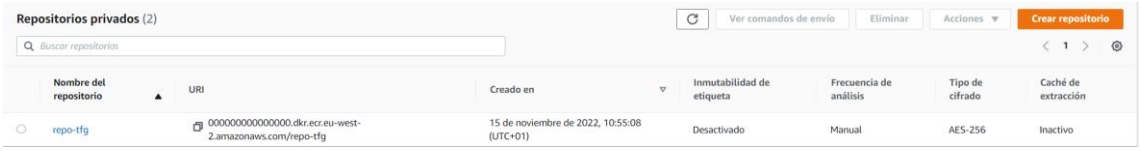


Ilustración 4 Panel de Elastic Container Registry

5.5. Definiendo la infraestructura de AWS con Terraform

En primer lugar, para poder utilizar terraform es necesario descargar el ejecutable y añadirlo a las variables del entorno. Podemos descargarlo desde la página oficial de HashiCorp Terraform.

Una vez tenemos terraform en nuestro ordenador podemos comenzar a definir la infraestructura que deseamos que se cree en AWS.

5.5.1. Configuración general, config.tf

```
variable "region" {
  default     = "eu-west-2"
  description = "Region of AWS"
}

provider "aws" {
  region = var.region
}

data "aws_availability_zones" "available" {}

locals {
  cluster_name = "tfg-${random_integer.suffix.result}"
}

resource "random_integer" "suffix" {
  min = 100
  max = 999
}
```

Código 7 Configuración básica Terraform

Aquí se define la configuración básica. En *provider* se debe especificar el proveedor de nube que se va a utilizar, en este caso AWS. AWS ofrece la posibilidad de elegir entre diferentes regiones por tanto también se tiene que especificar en cual se quiere crear, *eu-west-2* corresponde con Londres.

El cluster debe tener un nombre único en la región, por ellos se le añade un número aleatorio al final.

5.5.2. Definición de la red virtual privada en la nube (VPC), vpc.tf

```
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "3.2.0"

  name                 = "tfg-vpc"
  cidr                 = "10.0.0.0/16"
  azs                  = data.aws_availability_zones.available.names
  private_subnets      = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets       = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]
  enable_nat_gateway   = true
  single_nat_gateway   = true
  enable_dns_hostnames = true

  tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }

  public_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/elb"                      = "1"
  }

  private_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb"             = "1"
  }
}

```

Código 8 Configuración de red virtual privada en la nube

La VPC de AWS permite lanzar recursos de AWS en una red virtual que el usuario ha definido según sus necesidades. Esta red virtual se parece mucho a una red tradicional que operaría en su propio centro de datos, con los beneficios de usar la infraestructura escalable de AWS.

Para las pruebas se definen tres zonas de disponibilidad con el fin de evitar problemas si alguna de las subredes deja de funcionar.

5.5.3. Definición de los grupos de seguridad, security-groups.tf

```
resource "aws_security_group" "worker_group_mgmt_one" {
  name_prefix = "worker_group_mgmt_one"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port = 22
    to_port   = 22
    protocol  = "tcp"

    cidr_blocks = [
      "10.0.0.0/8",
    ]
  }
}

```

Código 9 Configuración de grupos de seguridad

Una vez configurada la VPC creamos los grupos de seguridad. Estos se encargarán de autorizar o denegar el tráfico en nuestra red privada. En este caso se muestra cómo se habilita el tráfico al puerto 22 desde nuestra red local.

Únicamente se muestra un grupo de seguridad, pero la estructura de los no mostrados es la misma.

5.5.4. Uniendo toda la configuración, eks-cluster.tfes

```
module "eks" {
  source          = "terraform-aws-modules/eks/aws"
  version         = "17.24.0"
  cluster_name    = local.cluster_name
  cluster_version = "1.20"
  subnets         = module.vpc.private_subnets

  vpc_id = module.vpc.vpc_id

  workers_group_defaults = {
    root_volume_type = "gp2"
  }

  worker_groups = [
    {
      name                          = "group-1"
      instance_type                 = "t2.small"
      additional_security_group_ids = [aws_security_group.worker_group_mgmt_one.id]
      asg_desired_capacity          = 3
    },
  ]
}
```

Código 10 Configuración de cluster y grupos de trabajo

El archivo *eks-cluster.tfes* es donde se une toda la configuración que hemos creado. Utiliza un módulo público de AWS, reutilizable para simplificar la creación de clusters. En este archivo se hace referencia a las variables que hemos definido anteriormente, como por ejemplo *local.cluster_name* que está definido en el primer archivo que hemos creado.

En el apartado de worker_groups se define qué tipo de instancia queremos utilizar. En este caso se utilizará una de tipo t2, la cual posee las siguientes características:

- Procesador Intel Xeon Scalable de hasta 3,3 GHz (Haswell E5-2676 v3 o Broadwell E5-2686 v4)
- Procesadores Intel Xeon de alta frecuencia
- CPU ampliable, que se rige por créditos de CPU y rendimiento base constante
- Tipo de instancia de uso general de bajo costo e incluida en la capa gratuita*
- Equilibrio de recursos informáticos, de memoria y de red

Instancia	CPU virtual*	Créditos por hora de CPU	Memoria (GiB)	Almacenamiento	Rendimiento de red
t2.nano	1	3	0,5	Solo EBS	Bajo
t2.micro	1	6	1	Solo EBS	De bajo a moderado
t2.small	1	12	2	Solo EBS	De bajo a moderado
t2.medium	2	24	4	Solo EBS	De bajo a moderado
t2.large	2	36	8	Solo EBS	De bajo a moderado
t2.xlarge	4	54	16	Solo EBS	Moderado
t2.2xlarge	8	81	32	Solo EBS	Moderado

Ilustración 5 características de las instancias T2 en AWS

Las instancias T2 de Amazon EC2 son instancias de rendimiento ampliable que proporcionan un nivel base de rendimiento de la CPU con la capacidad de ampliarse por encima del nivel básico.

Las instancias T2 ilimitadas pueden conservar un rendimiento de CPU alto durante el tiempo que una carga de trabajo lo necesite. Para la mayoría de las cargas de trabajo generales, las instancias T2 ilimitadas ofrecerán rendimiento amplio sin cargo adicional. Si la instancia requiere un mayor uso de la CPU durante un período prolongado, también puede hacerlo con un cargo fijo adicional de 5 centavos de USD por hora de CPU virtual.

El rendimiento base y la capacidad de ampliarse se rigen por los créditos de la CPU. Las instancias T2 reciben créditos de CPU continuamente a un índice fijo en función del tamaño de la instancia, acumulando así créditos de CPU cuando están inactivas y consumiéndolos cuando están activas.

5.6. Creando la infraestructura de AWS con Terraform

Una vez definida la infraestructura es hora de hacerla realidad. Primero debemos inicializar Terraform.

```
1  terraform init
```

Código 11 Comando para la inicialización de terraform

Terraform init descarga todos los módulos y archivos necesarios para el proveedor de servicio en la nube AWS. En caso de que todo vaya correctamente nos mostrará un mensaje diciendo que terraform está listo, en caso contrario mostrará por consola que ha generado el error.

Gracias al siguiente comando terraform permite al usuario encargado de crear la infraestructura ver que es lo que se va a crear en AWS, aquí veremos que se crean cosas que no hemos definido en los ficheros anteriores, no hay que preocuparse, son configuraciones necesarias para que nuestra red privada funcione correctamente.

```
1  terraform plan
```

Código 12 Comando para ver planificación en terraform

```
1  terraform apply
```

Código 13 Comando para crear la infraestructura

Después de revisar que todo es correcto podemos por fin crear la infraestructura con el último comando.

Terraform apply primero hace un terraform plan y muestra lo que se va a crear y pregunta al usuario si está de acuerdo, una vez el usuario acepta se procede a la creación de la infraestructura. Durante la creación si ocurriese algún problema se eliminará todo lo que ya se haya creado y se mostrará al usuario cual ha sido el error, normalmente causado por falta de permisos en el usuario configuración para el CLI de AWS.

Una vez finalizado el proceso ya podremos ver nuestra VPC desde el panel de AWS.

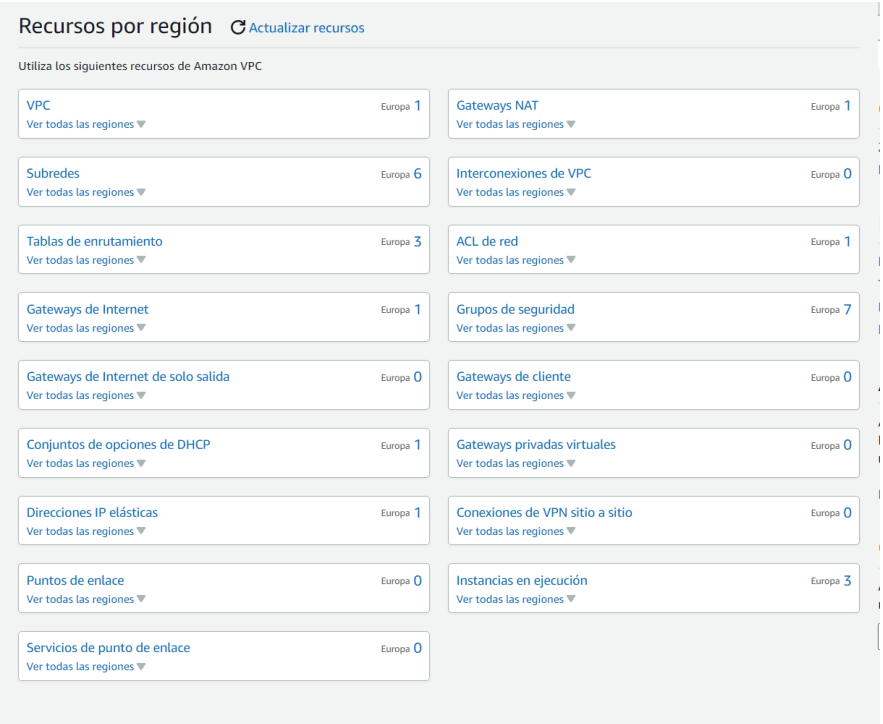


Ilustración 6 Panel de la red en la nube privada (VPC)

5.7. Asignación del contenedor a la instancia creada

Para poder acceder por la línea de comandos a nuestro nuevo cluster debemos ejecutar el siguiente comando, para añadir el contexto a la configuración de kubectl.

```
aws eks --region $(terraform output -raw region) update-kubeconfig --name $(terraform output -raw cluster_name)
```

Código 14 Acceso al cluster por la línea de comandos

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tfg
  labels:
    app: tfg
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tfg
  template:
    metadata:
      labels:
        app: tfg
    spec:
      containers:
      - name: tfg
        image: xxxxxxxxxxx.dkr.ecr.eu-west-2.amazonaws.com/xxxx
        resources:
          requests:
            memory: "1024Mi"
            cpu: "1000m"
          limits:
            memory: "2048Mi"
            cpu: "2000m"
```

Código 15 Configuración del contenedor a ejecutar

Una vez tenemos el clúster creado y la instancia de la maquina en funcionamiento es hora de asignarle el contenedor que debe ejecutar. Esto se define en el archivo de configuración *manifest.yml* donde se indica cual es el contenedor que debe ejecutar, así como el número de replicas y la capacidad mínima y máxima de memoria y cpu.

En el apartado *image* se debe poner el *repositoryUri* de la imagen que creamos anteriormente.

Una vez configurado nuestro manifest.yml debemos ejecutar los siguientes comandos. Primero creamos un namespace y después aplicamos nuestro manifiesto en dicho namespace.

```
kubectl create namespace tfg

kubectl apply -f manifest.yml -n tfg
```

Código 16 Creación del namespace y aplicación del manifiesto

Con esto nuestra aplicación debería comenzar a ejecutarse en el cluster.

5.8. Obteniendo información de nuestro cluster

```
PS Terraform> aws eks list-clusters
{
    "clusters": [
        "tfg-490"
    ]
}
```

Código 17 Listado de los clusters activos

Mostramos el listado de clusters para ver el nombre que le ha asignado a nuestro nuevo cluster. Ya que lo configuramos para tener un número aleatorio después del prefijo tfg.

```
PS Terraform> kubectl get nodes
NAME                                         STATUS   ROLES    AGE   VERSION
ip-10-0-1-25.eu-west-2.compute.internal      Ready       57m   v1.20.15-eks-fb459a0
```

Código 18 Listado de los nodos activos

Primero, vamos a explorar cuántos workers están ejecutándose en este clúster. El siguiente comando retornará el worker que creamos, la versión de Kubernetes y la edad de cada instancia de Amazon Elastic Compute Cloud desde la creación o actualización.

Ahora nos interesa obtener la información de nuestros pods, para ellos podemos ejecutar el siguiente comando. El flag *-n* es para indicar el nombre del namespace del cual queremos obtener los pods. En caso de querer obtener todos los pods que tenemos podemos usar la flag *–all-namespaces*

```
PS Terraform> kubectl get nodes
NAME                  READY   STATUS    RESTARTS   AGE
tfg-5ff4465ff9-68skt   1/1     Running   0          9s
```

Código 19 Información sobre nodos activos

El comando nos muestra el nombre del pod, si está listo, el estado del pod, el número de reinicios y el tiempo que llevan en ejecución. Los pods se auto-curan, es decir, en caso de ocurrir algún error durante el despliegue o ejecución el pod se eliminará y se volverá a crear automáticamente. Por ello en caso de que el número de reinicios sea elevado deberíamos revisar nuestro código ya que algo no está funcionando de la manera correcta.

Para obtener los logs que ha generado nuestro contenedor podemos ejecutar el siguiente comando.

```
kubectl get logs [nombrePod] -n [namespace]
```

Código 20 Obtención de los logs de un pod

Si ha ocurrido alguna excepción durante la ejecución aquí podremos ver que es lo que ha causado el error.

Puede que el error haya ocurrido antes de comenzar la ejecución, durante la fase de creación del pods. Gracias al comando describe podemos ver los pasos que se han llevado a cabo durante todo el proceso de creación del pod y quizá poder ver donde ha ocurrido el error.

```
Events:
  Type     Reason     Age                From               Message
  ----     ------     ----               ----               -------
  Normal   Scheduled  18m                default-scheduler  Successfully assigned tfg/tfg-5ff4465ff9-68skt to ip-10-0-1-25.eu-west-2.compute.internal
  Normal   Pulled     18m                kubelet            Successfully pulled image "419101797974.dkr.ecr.eu-west-2.amazonaws.com/tfg" in 142.448704ms
  Normal   Pulled     14m                kubelet            Successfully pulled image "419101797974.dkr.ecr.eu-west-2.amazonaws.com/tfg" in 115.696693ms
  Normal   Pulled     10m                kubelet            Successfully pulled image "419101797974.dkr.ecr.eu-west-2.amazonaws.com/tfg" in 101.180458ms
  Normal   Pulled     6m25s              kubelet            Successfully pulled image "419101797974.dkr.ecr.eu-west-2.amazonaws.com/tfg" in 91.122165ms
  Normal   Pulling    98s                kubelet            Pulling image "419101797974.dkr.ecr.eu-west-2.amazonaws.com/tfg"
  Normal   Created    98s                kubelet            Created container tfg
  Normal   Started    98s                kubelet            Started container tfg
  Normal   Pulled     98s                kubelet            Successfully pulled image "419101797974.dkr.ecr.eu-west-2.amazonaws.com/tfg" in 124.113251ms
```

Código 21 Logs del pod en ejecución

6. Eliminando la infraestructura

AWS cobra sus servicios por uso, por tanto, no nos interesa tener la infraestructura activa de forma permanente. Ahora bien, eliminar todo lo que hemos creado desde la cuenta de AWS resulta muy tedioso, ya que gran parte de la configuración y servicios creados están relacionados unos con los otros y debemos seguir un orden a la hora de eliminarlos.

Además, hay servicios activos que quizás no conozcamos y que si no los eliminamos seguirán generando costes. AWS dispone de un panel en el que se muestran cuantos servicios activos hay, pero no hay una forma directa de acceder a ellos.

Terraform es capaz de eliminar todo lo que hemos creado con solo un comando. Esto hace que durante el desarrollo los costes bajen mucho, ya que con dos comandos podemos levantar la infraestructura necesaria para hacer pruebas y con otro eliminarla para que no siga generando costes.

Si que es verdad que tiene un par de inconvenientes, por un lado, el proceso de creación y eliminación de la infraestructura toma su tiempo. Por otro lado, AWS cobra por cada vez que se crea un cluster, por ello tampoco se puede estar generando y eliminando a la ligera.

Hay que encontrar el equilibrio, cuando sabes que no vas a utilizar el clúster en 2 días o más sí que es aconsejable eliminarlo, en caso contrario no merece la pena ya que el coste de volver a crearlo es parecido al de mantenerlo activo.

7. Errores surgidos durante el proceso

A continuación, se explican los errores más importantes ocurridos durante el proceso.

7.1. Problemas con la imagen de CentOS

A la hora de crear el contenedor con la imagen de CentOS me encontré con el siguiente error al ejecutar el comando de construcción.

```
 => ERROR [ 2/13] RUN yum install -y epel-release
------
 > [ 2/13] RUN yum install -y epel-release:
#6 1.679 CentOS Linux 8 - AppStream                        92  B/s |  38  B     00:00
#6 1.689 Error: Failed to download metadata for repo 'appstream': Cannot prepare internal mirrorlist: No URLs in mirrorlist
------
executor failed running [/bin/sh -c yum install -y epel-release]: exit code: 1
```

Código 22 Error en la creación del contenedor

Por diseño, CentOS utiliza una *mirror list* para descargar paquetes e instalar actualizaciones del sistema operativo. Esta lista es dinámica y generalmente puede ser dominios *.edu* que alojan los archivos del sistema operativo. En algunos entornos donde el acceso a Internet está restringido y la inclusión de cientos de dominios en blanco no es factible, la inclusión en blanco solo un dominio es más práctico. Para solucionar el problema obligaremos a yum a conectarse solo al *mirror list* oficial de CentOS, añadiendo las siguientes líneas al Dockerfile antes de utilizar yum.

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

Código 23 Solución del error

7.2. Permisos AWS con Terraform

Para que terraform pueda crear la infraestructura definida es necesario que el usuario correspondiente tenga configurados los siguientes permisos, ya que en caso contrario terraform fallará en el proceso de creación

- AmazonEC2ContainerRegistryFullAccess
- AmazonEC2FullAccess
- IAMFullAccess
- AmazonECS_FullAccess
- AmazonElasticContainerRegistryPublicFullAccess

Como los nombres indican estos permisos otorgan acceso completo a los servicios referenciados. Esto no es la mejor solución, pero si la más práctica de cara a un desarrollo en fase de aprendizaje. Cada servicio dispone de permisos específicos para cada acción.

Una vez este todo en funcionamiento y quede bien definido como debe ser la infraestructura se deberían eliminar los permisos generales y añadir únicamente los estrictamente necesarios.

7.3. Error CrashLoopBackOff en los pods

Tras la aplicar el manifiesto a los pods creados examinando su estado nos encontramos con la siguiente sorpresa.

```
NAME                   READY   STATUS             RESTARTS   AGE
tfg-5ff4465ff9-h7vmj   0/1     CrashLoopBackOff   1          34s
```

Código 24 Estado del pod

Por algún motivo desconocido el pod estaba teniendo un error, al tener el error Kubernetes trata de auto curarlo, eliminando el pod y volviéndolo a crear. Pero en este caso el error ocurría de igual manera.

El pod únicamente ejecuta un programa que copia un fichero de un almacenamiento de S3 a otro, con lo que es improbable que eso sea la causa del error, además de que el programa ha sido probado en local previamente y funcionaba correctamente.

Tras buscar de cerca del error las principales causas son:

- Error de configuración, como un error tipográfico en el archivo de configuración
- Recursos no disponibles, no consigue acceder a la imagen configurada
- Problemas en el nodo.
- Falta de recursos, tanto de CPU como de memoria

El primer paso para solucionar el error es obtener la información del pod para ver si podemos sacar algo en claro.

```
kubectl describe pods tfg-5ff4465ff9-h7vmj -n tfg
Events:
  Type     Reason     Age              From               Message
  ----     ------     ----             ----               -------
  Normal   Scheduled  70s              default-scheduler  Successfully assigned tfg/tfg-5ff4465ff9-h7vmj to ip-10-0-1-175.eu-west-2.compute.internal
  Normal   Pulled     50s              kubelet            Successfully pulled image "419101797974.dkr.ecr.eu-west-2.amazonaws.com/tfg" in 19.53737158s
  Normal   Pulled     48s              kubelet            Successfully pulled image "419101797974.dkr.ecr.eu-west-2.amazonaws.com/tfg" in 91.499289ms
  Normal   Pulled     32s              kubelet            Successfully pulled image "419101797974.dkr.ecr.eu-west-2.amazonaws.com/tfg" in 96.144106ms
  Normal   Pulling    3s (x4 over 70s)  kubelet            Pulling image "419101797974.dkr.ecr.eu-west-2.amazonaws.com/tfg"
  Normal   Created    3s (x4 over 49s)  kubelet            Created container tfg
  Normal   Started    3s (x4 over 49s)  kubelet            Started container tfg
  Normal   Pulled     3s               kubelet            Successfully pulled image "419101797974.dkr.ecr.eu-west-2.amazonaws.com/tfg" in 88.716884ms
  Warning  BackOff    2s (x5 over 47s)  kubelet            Back-off restarting failed container
```

Código 25 Eventos ocurridos en el pod donde ocurre el error

Podemos ver como sí que ha conseguido asignar y descargar la imagen. Además, el contenedor se ha llegado a crear y ejecutar correctamente. Con esta información podemos descartar que sea un error de acceso a la imagen o de configuración del Dockerfile.

Puede que el error venga por falta de recursos del pod, con lo que vamos a aumentar la capacidad de CPU y de memoria. Pero tras aumentar al máximo la memoria y la CPU el error sigue persistiendo.

Recordemos que se ha definido una instancia de tipo t2.small, lo cual debería ser más que suficiente para lo que queremos realizar. Aun así, para evitar dudas vamos a probar con una instancia con más capacidad, como por ejemplo la t2.large. Otra vez más el error vuelve a ocurrir con lo que podemos descartar que sea por falta de recursos.

Vamos ahora con el último problema potencialmente probable, problemas en el nodo.

```
kubectl describe node ip-10-0-1-175.eu-west-2.compute.internal
Conditions:
  Type             Status  LastHeartbeatTime                 LastTransitionTime                Reason                       Message
  ----             ------  -----------------                 ------------------                ------                       -------
  MemoryPressure   False   Thu, 29 Dec 2022 18:28:08 +0100   Thu, 29 Dec 2022 16:40:34 +0100   KubeletHasSufficientMemory   kubelet has sufficient memory available
  DiskPressure     False   Thu, 29 Dec 2022 18:28:08 +0100   Thu, 29 Dec 2022 16:40:34 +0100   KubeletHasNoDiskPressure     kubelet has no disk pressure
  PIDPressure      False   Thu, 29 Dec 2022 18:28:08 +0100   Thu, 29 Dec 2022 16:40:34 +0100   KubeletHasSufficientPID      kubelet has sufficient PID available
  Ready            True    Thu, 29 Dec 2022 18:28:08 +0100   Thu, 29 Dec 2022 16:41:18 +0100   KubeletReady                 kubelet is posting ready status
```

Código 26 Descripción de las condiciones del nodo

Aparentemente no hay problemas en el nodo con lo que también podemos descartar esta causa por ahora.

Tras consultar muchos foros, videos y tutoriales vi que el error podía ser debido a una finalización prematura del contenedor, es decir, que el contenedor finalizara lo que tenía que hacer demasiado rápido, de forma que kubectl lo detectara como un error.

Finalmente, añadiendo un simple sleep al programa de forma que no finalice muy rápido el error desaparece.

8. Pruebas de carga

Hasta ahora solo se ha probado a ejecutar programas simples, pero qué pasará cuando tenga que analizar cientos de fotografías. Las fotografías sacadas con el dron utilizado en el laboratorio pesan alrededor de 70MB, para evitar costes altos en las pruebas de rendimiento se ha utilizado una única foto de 7MB la cual ha sido subida 1000 veces al bucket de S3, donde el nombre de cada copia será la hora en la que se ha subido.

La subida se ha realizado con un pequeño programa en Python corriendo dentro de uno de los pods que hemos creado anteriormente.

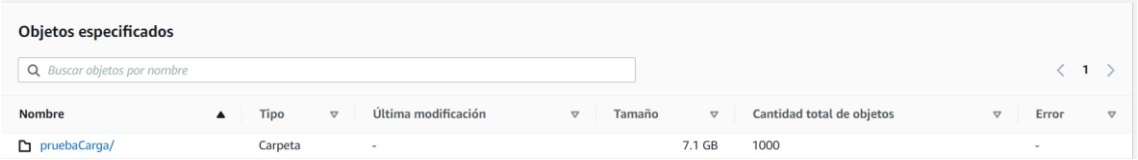
```python
s3 = boto3.resource('s3')
utc_dt = datetime.now(timezone.utc)
print("Fecha Inicio: "+format(utc_dt.astimezone().isoformat()).replace(":","-").replace("T"," ").split(".")[0])
i=0
for i in range(1000):
    utc_dt = datetime.now(timezone.utc)
    aux = format(utc_dt.astimezone().isoformat()).replace(":","-").replace("T"," ").split(".")[0] +"-"+ str(i)
        s3.meta.client.upload_file('descarga.jpg', 'tfg-david', "pruebaCarga/"+aux+".jpg")

print("Fecha Fin: "+format(utc_dt.astimezone().isoformat()).replace(":","-").replace("T"," ").split(".")[0])
```

Tras la ejecución se han obtenido los siguientes resultados:

Fecha Inicio: 2022-12-02 16:56:04 → **Fecha Fin**: 2022-12-02 17:41:36

Es decir, el programa ha tardado cerca de 45 minutos en subir 1000 fotos de 7MB. Haciendo un cálculo simple se obtiene que la velocidad de subida es cercana a los 2.6MB por segundo y que se sube una imagen cada 2.7 segundos.



Además, cabe destacar que durante la subida no ha ocurrido ningún error y todas las fotos se han almacenado correctamente.

9. Pruebas de procesado de imágenes

En cuanto al procesado de las imágenes, la red neuronal que se usará para la detección de los postes eléctricos ya está desarrollada en el centro de investigación de la UPNA. Para realizar las pruebas de procesado de imágenes se ha utilizado una red neuronal básica, partiendo de unos datasets con imágenes ya clasificadas gracias a la librería Tensorflow_datatsets. En este caso se trata de un dataset con imágenes de ropa, donde se clasifican según su categoría (camiseta, pantalón, zapatillas, etc.).

Se trata de un dataset con 60.000 datos de entrenamiento y 10.000 datos de prueba, si bien es cierto que las imágenes están en escala de grises y son de 28*28 pixeles la finalidad de las pruebas es comprobar si todo el proceso se ejecuta correctamente, ya que los tiempos obtenidos no son significativos debido a la diferencia en las características de las imágenes usadas y las que realmente se usaran.

Aun así, el pod no ha tenido ningún problema y ha conseguido entrenar la red neuronal en apenas 2 minutos. Y una vez entrenada es capaz de clasificar 100 imágenes en 6 segundos con un porcentaje de acierto del 86%.

Ilustración 7 Prueba de clasificación de 25 imágenes, con un 92% de acierto

10. Precios y gastos generados

Está muy bien tener el proyecto en funcionamiento, pero AWS no es un servicio gratuito y la pregunta del millón es, ¿Cuánto ha costado tenerlo funcionando? Para analizar los gatos se van a tener en cuenta los gastos generados en noviembre.

Hay que tener en cuenta que durante el proceso la mayoría de los despliegues no han ido bien a la primera y, por tanto, una parte de los costes corresponde a la investigación y pruebas realizadas.

Además, AWS ofrece un free tier es una gran cantidad de sus funcionalidades, siendo en algunas más amplias que en otras. Por ello, aunque no hayan generado costes, son gastos para tener en cuenta de cara a usarse en un entorno real y no de desarrollo.

Los gastos vienen todos de dos servicios, AWS Elastic Compute Cloud y AWS Elastic Container Service for Kubernetes.

10.1. AWS Elastic Compute Cloud (AWS EC2)

Costos relacionados con la computación en la nube. Para que las instancias de una subred privada puedan conectarse a servicios fuera de la VPC, pero los servicios externos no pueden iniciar una conexión con esas instancias en necesaria una NAT Gateway, la cual crea AWS por defecto al crear el cluster. Esta NAT Gateway tiene un coste de 0.05$ por hora. Además, hay un cobre de 0.05$ por GB de datos procesados a través de la NAT Gateway.

Amazon Elastic Compute Cloud NatGateway		$13.33
$0.05 per GB Data Processed by NAT Gateways	12.693 GB	$0.63
$0.05 per NAT Gateway Hour	254.000 Hrs	$12.70

Por otro lado, el gasto principal de computación en la nube lo crean las instancias en las que corren los pods, en este caso de tipo t2.small, como definimos en la configuración de

Terraform. Generando un gasto de 0.026$/hora. Durante las pruebas se ejecutaron varias instancias a la vez, de ahí que el número de horas de uso sea ciertamente alto, 741 horas.

Amazon Elastic Compute Cloud running Linux/UNIX		$19.28
$0.026 per On Demand Linux t2.small Instance Hour	741.555 Hrs	$19.28

Además, se cobra 0.116$ por GB de almacenamiento usado, siendo en el mes de noviembre 73GB. Cada vez que se lanza la instacia del contenedor instala la imagen de centos y las dependencias necesarias para ejecutar el codigo, al haber hecho muchas pruebas el uso de almacenamiento se eleva.

EBS		$8.47
$0.116 per GB-month of General Purpose SSD (gp2) provisioned storage - EU (London)	73.045 GB-Mo	$8.47

10.2. AWS Elastic Container Service for Kubernetes (AWS ECS)

Este servicio es el correspondiente a la creación del clúster. Como hemos visto en el apartado anterior tener un cluster activo genera gastos, por ello el cluster no ha estado en funcionamiento durante todo el mes.

Gracias a Terraform el cluster era creado los días que iba a estar trabajando en el proyecto y eliminado cuando no se iba a usar en varios días. De esta forma se evitaban sobrecostes por su utilización, ya que AWS cobra por cada hora en la que el cluster está activo.

Para ser exactos, el precio de tener el cluster activo por hora es de 0.1$/hora. El mes de noviembre el cluster estuvo activo durante 356.67 horas, es decir, cerca de unos 15 días.

▾ Elastic Container Service for Kubernetes		$35.67
▾ **EU (London)**		**$35.67**
Amazon Elastic Container Service for Kubernetes CreateOperation		$35.67
Amazon EKS cluster usage in EU (London)	356.677 Hours	$35.67

Suponiendo que el cluster debe estar activo durante todo el mes, se generaría un gasto de 75€ mensuales únicamente por tener el cluster activo.

10.3. Tabla de costes

Los costes totales del mes de noviembre son los siguientes:

SERVICIO	PRECIO	USO	COSTE
Amazon Elastic Compute Cloud NatGateway			13.33$
Data Processed by NAT Gateways	0.050$/GB	12.693GB	0.63$
NAT Gateway Hour	0.050$/Hora	254h	12.70$
Amazon Elastic Compute Cloud running Linux/UNIX			19.28$
On Demand Linux t2.small Instance Hour	0.026$/Hora	741.555h	19.28$
Elastic Block Store			8.47$
General Purpose SSD (gp2) provisioned storage	0.116$/GB	73.045GB	8.47$
Elastic Container Service for Kubernetes			35.67$
Amazon EKS cluster usage in EU	0.100$/Hora	356.677h	35.67$
			76.75$

Al precio total de 76.75$ hay que sumarle un 21% de IVA, con lo que el total a pagar es 92.87$, sin olvidar que el precio está en dólares y nosotros pagaremos el equivalente en euros.

10.4. Free tier de AWS

A parte de los gastos generados se han utilizado los siguientes servicios de la capa gratuita de Amazon:

- AWS Data Transfer:
  - 1.0 GB for free per month as part of AWS Free Usage Tier (Global-DataTransfer-Regional-Bytes)
- Amazon EC2 Container Registry:
  - 500 MB-month of Amazon EC2 Container Registry storage for new customers
- Amazon Simple Storage Service
  - 5 GB of Amazon S3 standard storage
  - 2,000 Put, Copy, Post or List Requests of Amazon S3
- Amazon Elastic Compute Cloud:
  - 30 GB of Amazon Elastic Block Storage in any combination of General Purpose (SSD) or Magnetic

En el caso del último servicio se superó la capa gratuita y como hemos podido observar antes generó un gasto de 8,47$. Los servicios gratuitos se renuevan cada mes automáticamente, con lo que, salvo que los servicios de la capa gratuita cambien podremos seguir usándolos cada mes.

11. Gestión del proyecto

La propuesta inicial del proyecto fue a principios de Julio y tras definir el tema y objetivo comencé a trabajar en él. Tras esto lo primero fue definir las fases del proyecto.

11.1. Primera fase

Investigación de los diferentes proveedores de servicios en la nube, ya que lo primero era decidir donde se iba a realizar el proyecto, analizando los servicios que ofrecían las diferentes opciones, siendo la elegida finalmente AWS.

11.2. Segunda fase

Al no haber trabajado nunca con AWS la segunda fase se centraría en aprender los conceptos y funcionamiento básicos, mediante la documentación, foros, videos y cursos online. Centrando el foco sobre todo en el despliegue de aplicaciones con Kubernetes.

11.3. Tercera fase

Una vez tenia los conocimientos básicos necesarios para poder entender que estaba haciendo tocaba empezar a hacer las primeras pruebas, creando y desplegando aplicaciones simples con el fin de entender y saber hacer el proceso de despliegue de infraestructura y una aplicación contenerizada en AWS.

11.4. Cuarta fase

Con los conocimientos adquiridos, por fin era hora de crear la infraestructura que se planteó desde un principio para el proyecto, aplicando todo lo aprendido para conseguir el resultado deseado.

11.5. Control de versiones

Durante todo el proceso he usado un repositorio en GitHub para ir almacenando el progreso y los cambios realizados. En el repositorio he almacenado tanto el código, como la memoria, facturas y cualquier tipo de documento, es decir, cualquier cosa relacionada con el TFG. En mi caso, al disponer de dos ordenadores, era muy importante tener siempre las últimas modificaciones en ambos ordenadores.

11.6. Reuniones de seguimiento

Durante todo el proceso hubo reuniones de seguimiento con el tutor, Jesús Villadangos, para comentar el progreso realizado, así como las complicaciones surgidas para tratar de ver cómo solucionarlas. A veces es necesarios el punto de vista de otra persona para poder continuar avanzado en puntos en los que estas atascado o en los que no sabes si vas bien encaminado.

12. Conclusiones y líneas futuras

12.1. Conclusiones técnicas

AWS es una plataforma con una potencia increíble, en cuanto a computación en la nube o despliegue de servicios se puede llegar a hacer prácticamente lo que te propongas, siendo la mayor limitación los conocimientos y no los servicios ofrecidos.

Terraform es una herramienta potentísima combinada con AWS. El hecho de poder definir cómo quieres que sea la infraestructura con unas pocas líneas de código y que ejecutando un comando se cree toda la infraestructura y tenga una máquina ejecutando el código que has creado en un servidor remoto es increíble.

Además, de cara al desarrollo el hecho de poder eliminar todo lo creado con un comando aporta mucha tranquilidad, sabiendo que no tienes por qué preocuparte por sobrecostes de olvidarte algo ejecutándose en la nube.

En cuanto a los programas creados, las librerías de Python existentes para conectarse al almacenamiento de AWS funcionan de excelente manera, haciendo que el desarrollo sea más fácil y no tengas que invertir demasiado tiempo en transferencia de ficheros.

Si que es cierto que aun habiendo decidido usar AWS, sus competidores más cercanos, como son Azure y Google, no tienen nada que envidiar en cuanto a los servicios ofrecidos, ya que en los tres casos ofrecen prácticamente lo mismo. La principal diferencia es la popularidad y el boca a boca, ya que en el ámbito de la informática casi todo el mundo conoce AWS, pero Google Cloud y Azure no.

12.2. Conclusiones personales

Tras la realización del trabajo puedo confirmar y afianzar lo que ya imaginaba, que el mundo de la computación en la nube es muy complejo y extenso. Aunque sienta que he aprendido muchísimo, tras haber invertido muchas horas aún sigo pensando que no se prácticamente nada.

El problema es que la barrera de entrada en cuanto a conocimientos es muy alta, al entrar por primera vez en AWS te encuentras con cientos de paneles, servicios, opciones y configuraciones diferentes, lo cual resulta muy abrumador, sobre todo en un trabajo como es el TFG en el que lo realizas de forma individual.

Ha resultado un reto interesante y atractivo aprender y entender como desplegar infraestructura con AWS, pero también ha resultado muy frustrante cuando no conseguía avanzar y no sabía ni por donde debía empezar. Aun así, valoro la experiencia de forma positiva.

Más allá de lo aprendido acerca de la computación en la nube, Kubernetes, Terraform y demás, me quedo con la capacidad para afrontar nuevos retos, superando los momentos difíciles y disfrutando de los buenos momentos.

12.3. Líneas futuras

En cuanto a líneas futuras, me gustaría aplicarlo en un entorno real, en el que se pruebe realmente la carga que sufrirán los pods procesando imágenes. Realizar pruebas con diferentes tipos de las instancias ofrecidas por AWS para encontrar la que mejor se ajuste a lo realmente necesitado.

13. Bibliografía

[1] AWS vs Azure vs Google: ¿Cuál es la mejor opción? [En línea]. Disponible en: http://dbandtech.com/aws-vs-azure-vs-google-cual-es-la-mejor-opcion [Accedido: 8-10-2022]

[2] ¿Qué es Docker? ¿Para qué se utiliza? [En línea] Disponible en: https://www.javiergarzas.com/2015/07/que-es-docker-sencillo.html [Accedido: 10-10-2022]

[3] Azure Kubernetes Service [En línea] Disponible en: https://azure.microsoft.com/es-mx/products/kubernetes-service/ [Accedido: 10-10-2022]

[4] Wikipedia- Kubernetes [En línea] Disponible en: https://es.wikipedia.org/wiki/Kubernetes [Accedido: 10-10-2022]

[5] Usando Kubernetes en local [En línea] Disponible en: https://lemoncode.net/lemoncode-blog/2021/6/12/usando-kubernetes-en-local-minikube-instalacion-en-windows [Accedido: 10-10-2022]

[6] ¿Qué es Dockerfile? [En línea] Disponible en: https://keepcoding.io/blog/que-es-dockerfile/ [Accedido: 16-10-2022]

[7] Contenerización de aplicaciones en Docker [En línea] Disponible en: https://www.nocountryforgeeks.com/contenerizacion-de-aplicaciones-en-docke [Accedido: 17-10-2022]

[8] ¿Qué es y para qué sirve Docker? [En línea] Disponible en: https://conocimientolibre.mx/docker/ [Accedido: 23-10-2022]

[9] Documentación Kubernetes [En línea] Disponible en: https://kubernetes.io/es/docs/concepts/ [Accedido: 26-10-2022]

[10] Documentación Terraform [En línea] Disponible en: https://developer.hashicorp.com/terraform [Accedido: 03-11-2022]

[11] AWS vs Azure en 2021 [En línea] Disponible en: https://kinsta.com/es/blog/aws-vs-azure [Accedido: 05-11-2022]

[12] Gartner Magic Quadrant for Cloud Infrastructure and Platform Services [En línea] Disponible en: https://www.gartner.com/en/documents/3989743 [Accedido: 07-11-2022]

[13] Documentación IBM Clústeres [En línea] Disponible en: https://www.ibm.com/docs/es/was-zos/9.0.5?topic=servers-introduction-clusters [Accedido: 07-11-2022]

[14] Cómo desplegar una aplicación en Kubernetes [En línea] Disponible en: https://aws.amazon.com/es/blogs/aws-spanish/picturesocial-como-desplegar-una-aplicacion-en-kubernetes/ [Accedido: 15-11-2022]

[15] Instancias T2 de Amazon EC2 [En línea] Disponible en: https://aws.amazon.com/es/ec2/instance-types/t2/ [Accedido: 20-11-2022]

[16] Kubernetes CrashLoopBackOff Error: What It Is and How to Fix It [En línea] Disponible en: https://komodor.com/learn/how-to-fix-crashloopbackoff-kubernetes-error/ [Accedido: 24-11-2022]

[17] Deep Learning: Clasificado imágenes con redes neuronales [En línea] Disponible en: https://www.lisdatasolutions.com/es/blog/deep-learning-clasificando-imagenes-con-redes-neuronales/ [Accedido: 28-11-2022]

[18] Copy files from Kubernetes to S3 and back [En línea] Disponible en: https://medium.com/nuvo-group-tech/copy-files-and-directories-between-kubernetes-and-s3-d290ded9a5e0 [Accedido: 01-12-2022]

[19] Develop a Docker Containerized Python API With Terraform, Gitlab, Kubernetes, and AWS [En línea] Disponible en: https://betterprogramming.pub/develop-a-docker-containerized-python-api-deployed-with-terraform-gitlab-kubernetes-and-aws-238234caaaf5 [Accedido: 05-12-2022]

[20] DOCKER & KUBERNETES : TERRAFORM AND AWS EKS [En línea] Disponible en: https://www.bogotobogo.com/DevOps/Docker/Docker_Kubernetes_Terraform_EKS.php [Accedido: 05-12-2022]

[21] A crash course on Terraform [En línea] Disponible en: https://blog.gruntwork.io/a-crash-course-on-terraform-5add0d9ef9b4 [Accedido: 07-12-2022]