# Proyecto BCOM TRANSFORMACIONES

## Guía de despliegue y uso de los DAG's con Airflow y Kubernetes

### Requisitos previos

* Tener instalado Dockerfile, en su versión más reciente
* Tener un repositorio público en DockerHub o un repositorio local de imágenes
* Tener instalado Kubernetes (k8) y kubectl
* (Opcional) Para ambiente de desarrollo, se recomienda instalar "Minikube" en vez de k8, la cual  
  simula un clúster de k8 dentro de un contenedor.
* Tener instalado el Minio Server en local o utilizar un AWS S3

## Despliegue de ambiente:

⚠️ *Los 2 siguientes pasos solo se deben realizar la primera vez. Y solo será  
necesario repetirlo si el archivo Dockerfile\_worker cambia y esto sucede si se necesitan  
más librerías externa para ejecutar los DAGs*

1.- Situarse en la carpeta **deploy-k8** y luego crear la imagen customizada de Airflow  
más las dependencias necesarias para ejecutar nuestro DAG. Por ejemplo, ejecutar el siguiente comando:

docker build -t airflow\_custom:1.0.0 -f Dockerfile\_worker .

2.- Ahora subimos nuestra imagen al DockerHub o repositorio local, para esto primero  
nos debemos loguear, crear el tag y realizar el push. Por ejemplo:

docker login  
docker tag airflow\_custom:1.0.0 <nombrecuenta>/<repositorio>:<tag>  
docker push <nombrecuenta>/<repositorio>:<tag>

3.- Con la imagen ya subida a un repositorio, colocamos este nombre en el archivo  
**airflow-envvars-configmap.yaml**, además que aprovechamos en actualizar y validar  
todas las configuraciones presentes y también en nuestro  
archivo de Secrets **airflow-secrets.yaml**

AIRFLOW\_\_KUBERNETES\_\_WORKER\_CONTAINER\_REPOSITORY=<cuenta/SERVER-URL>/<repositorio>  
AIRFLOW\_\_KUBERNETES\_\_WORKER\_CONTAINER\_TAG="<tag>"

NOTA: *Detalle de cada variable de entorno usada por los POD y Airflow:*

AIRFLOW\_\_KUBERNETES\_EXECUTOR\_\_NAMESPACE: bcom-airflow # El namespace en k8s donde los workers serán creados  
 AIRFLOW\_\_SCHEDULER\_\_DAG\_DIR\_LIST\_INTERVAL: '30' # Intervalo de tiempo en ir a buscar nuevos dags en la carpeta de dags  
 AIRFLOW\_\_LOGGING\_\_LOGGING\_LEVEL: INFO # Nivel de log de Airflow (webserver, scheduler y workers)  
 AIRFLOW\_\_WEBSERVER\_\_DEFAULT\_UI\_TIMEZONE: America/Lima # Timezone de la Web de Airflow  
 AIRFLOW\_\_CORE\_\_DEFAULT\_TIMEZONE: America/Lima # Timezone del Scheduler de Airflow  
 AIRFLOW\_\_KUBERNETES\_\_WORKER\_CONTAINER\_REPOSITORY: cristianfernando/airflow\_custom # Ruta de imagen a usar para crear el worker POD  
 AIRFLOW\_\_KUBERNETES\_\_WORKER\_CONTAINER\_TAG: "0.0.1" # Tag a usar para crear el worker POD  
 AIRFLOW\_\_KUBERNETES\_\_LOGS\_VOLUME\_CLAIM: airflow-logs-pvc # Nombre del volumen usado para almacenar los logs  
 AIRFLOW\_\_KUBERNETES\_\_ENV\_FROM\_CONFIGMAP\_REF: airflow-envvars-configmap # Nombre del configMap donde estan defenidos las variables de entorno  
 AIRFLOW\_\_KUBERNETES\_\_POD\_TEMPLATE\_FILE: /opt/airflow/templates/pod\_template.yaml # Ruta del host donde esta el template para iniciar el worker  
 AIRFLOW\_\_CORE\_\_EXECUTOR: KubernetesExecutor # Tipo de Executor a usar por Airflow  
 AIRFLOW\_\_DATABASE\_\_SQL\_ALCHEMY\_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow # Ruta de conexión por default de Airflow con Postgres  
 AIRFLOW\_\_CORE\_\_DAGS\_ARE\_PAUSED\_AT\_CREATION: 'true' # Pausar (que no esten iniciados) los DAG's cuando se creen  
 AIRFLOW\_\_CORE\_\_LOAD\_EXAMPLES: 'true' # Colocar los DAG's de ejemplo en la web  
 \_AIRFLOW\_DB\_UPGRADE: 'true' # Actualizar la BD si es necesario  
 \_AIRFLOW\_WWW\_USER\_CREATE: 'true' # Crear usuario inicial para usar la web  
 \_AIRFLOW\_WWW\_USER\_USERNAME: admin # Username del usuario de la web  
 \_AIRFLOW\_WWW\_USER\_PASSWORD: admin # Contraseña del usuario de la web  
 S3\_DAGS\_DIR: 's3://prueba1234568/dags' # Ruta del S3 (si en caso se usa S3) donde estan alojados los DAG's a sincronizar  
 SYNCHRONYZE\_DAG\_DIR: '30' # Tiempo (en segundos) para ir a buscar actualizaciones en la carpeta de dags  
 MINIO\_SERVER: 'http://192.168.49.2:9000' # Ruta del Minio Server (Si en caso se usa Minio)   
 MINIO\_DAGS\_DIR: '/prueba-ca/dags' # Ubicación de los DAG's dentro del bucket de Minio

Adicionalmente, validar o actualizar los valores del archivo **airflow-secrets.yaml** (recordar que  
tiene encriptación con método **"base64"** - tipo de Secret **Opaque**)

AWS\_ACCESS\_KEY: bWluaW9hZG1pbg== # ACCESS\_KEY de AWS si en caso se usa bucket S3 para guardar los DAG's  
 AWS\_SECRET\_KEY: bWluaW9hZG1pbg== # SECRET\_KEY de AWS si en caso se usa bucket S3 para guardar los DAG's  
 MINIO\_USER: bWluaW9hZG1pbg== # Usuario de Minio para conectarse al bucket del Minio Server  
 MINIO\_PASSWORD: bWluaW9hZG1pbg== # Contraseña de Minio para conectarse al bucket del Minio Server

4.- **Como nuestro despliegue hace uso de NFS como sistema de persistencia de datos, tanto para la BD de Postgres,  
logs de Airflow y dags desarrollados**, entonces se tiene que configurar los servidores NFS usados.  
Para eso modificamos y/o validamos el archivo **airflow-volumes.yaml** (sección de *PersistentVolume*) para corrobar los hosts y paths de los  
volúmenes. Por ejemplo, parte del template donde está el volúmen para los dags desarrollados es el siguiente:

apiVersion: v1  
kind: PersistentVolume  
metadata:  
 name: airflow-dags-pv  
 namespace: bcom-airflow  
spec:  
 capacity:  
 storage: 300Mi # Espacio solicitado para el storage de los dags  
 accessModes:  
 - ReadWriteMany  
 storageClassName: airflow-dags   
 nfs:  
 server: 192.168.1.11 # Host del servidor NFS a usar  
 path: "/mnt/nfs\_share" # Path dentro del servidor NFS dedicado a almacenar los DAG's desarrollados

Luego de configurar la ruta del host y path del servidor NFS a usar para cada requisito, se modifica y/o valida  
cuánto storage del volumen se va a usar para tal propósito. Por ejemplo para los dags desarrollados, vamos a  
indicar que de los 300Mi, vamos a utilizar 200Mi. Esto también dentro del archivo airflow-volumes.yaml, pero en la  
sección de PersistentVolumeClaim (este uso de volumen se enlaza con el volumen creado antes por medio del  
*storageClassName*):

apiVersion: v1  
 kind: PersistentVolumeClaim  
 metadata:  
 name: airflow-dags-pvc  
 namespace: bcom-airflow  
 spec:  
 accessModes:  
 - ReadWriteMany  
 storageClassName: airflow-dags  
 resources:  
 requests:  
 storage: 200Mi # Se solicita un uso de 200Mi para el volumen de tipo airflow-dags

5.- Ejecutar el script **script\_apply.sh** para crear todos los configsMaps, volúmenes, secrets, deployments  
y servicios para correr Airflow.

⚠️ *Nota*: Verifique si usará la sincronización de la carpeta **dags** con un bucket en Minio o en S3.  
De acuerdo a esto tendrá que ejecutar el archivo *sync-dags-deployment.yaml* o *sync-dags-deployment-s3.yaml* y  
colocarlo en los archivos **script-apply.sh** y **script-delete.sh**.

sh script\_apply.sh

Con este comando, tiene que esperar unos minutos para que todos los recursos esten levantados y corriendo  
normalmente. Todos los recursos serán creados en el namespace **bcom-airflow** (configurado en los  
templates)

6.- (OPCIONAL) Si en caso a deployado usando Minikube, será necesario exponer el puerto del Airflow Webserver  
para que desde su local puede ingresar a la web. La forma más simple es exponiendo directamente el puerto del  
servicio del clúster a nuestro localhost con la ayuda de *kubectl*. Ejecutamos el siguiente comando:

kubectl port-forward <ID-POD-AIRFLOW-WEBSERVER> 8081:8080 -n bcom-airflow

Ahora desde su navegador puede ingresar la ruta http://localhost:8081 para ver la Web de Airflow

7.- Validamos que nuestros POD's están corriendo en estado "Running" con el siguiente comando:

kubectl get pods -n bcom-airflow

## Agregar o actualizar DAG's en Airflow

Para actualizar el DAG ya existente de **TACOMVENTAS Y PROMOCIONES RESIDENCIAL** o agregar un nuevo  
DAG en este despliegue de Airflow, se siguen los siguientes pasos:

1.- Verificar que nuestro ambiente en Kubernetes está levantado correctamente. Ejecutamos el comando:

kubectl get pods -n bcom-airflow

2.- Los archivos actualizados del dag colocarlos en el bucket y prefijo correspondiente de **dags**.  
Se deben colocar en la ruta configurada en la variable de entorno del archivo **airflow-envvars-configmap.yaml**.  
Si usamos S3, sería en la variable *S3\_DAGS\_DIR*, si fuera Minio se usará *MINIO\_DAGS\_DIR*.  
La variable muestra la url completa <bucket>/<prefijo>. Por ejemplo:

S3\_DAGS\_DIR: 's3://prueba1234568/dags'  
MINIO\_DAGS\_DIR: '/prueba-ca/dags'

3.- Solo quedaría esperar n segundos, de acuerdo a las variables *SYNCHRONYZE\_DAG\_DIR* (el POD sync  
sincronizará el bucket con la carpeta dags cada n segundos) y *AIRFLOW\_\_SCHEDULER\_\_DAG\_DIR\_LIST\_INTERVAL*  
(el scheduler Airflow irá a buscar actualizaciones en su carpeta dags cada n segundos). Luego del tiempo  
establecido se podrá ver los cambios en la web de Airflow.

## Ejecutar DAG TACOMVENTAS y PROMOCIONES\_RESIDENCIAL

*Requisitos para ejecutar el DAG:*

* El ambiente este desplegado correctamente y los POD's en estado "Running"
* Crear la conexión en la web de Airflow (usuario admin tiene estos permisos) con el nombre que está configurado en app\_conf.yml con  
  parámetro **s3\_conn\_id**.
* Validar todas las rutas (bucket y prefijos) y configuraciones de los 7 insumos en app\_conf.yml.

1.- En la web de Airflow, ubicamos nuestro DAG con nombre: **"BCOM\_DAG\_TRANSFORMACION\_TACOMVENTAS\_PROMOCIONESRESIDENCIAL"**  
(configurado como constante en dag\_transformacion\_tacomventas\_promoresidencial.py) y lo inicializamos  
por medio de su checkbox (usuario admin puede realizarlo).

2.- De acuerdo a la configuración actual, el DAG se ejecutará cada 2 horas (configurado en app\_conf.yml con parámetro  
**schedule**).

## Desplegar Minio Server (Single Node) - Para ambiente de desarrollo

Vamos a instalar Minio Server y tener la funcionalidad de crear buckets, prefijos y subir archivos en  
ellos. Minio reemplazaría AWS S3 dado que ya nos va a proporcionar los buckets.

1.- Descargar el instalador y ejecutarlo con los siguientes comandos (Para Debian/Ubuntu):

wget https://dl.min.io/server/minio/release/linux-amd64/archive/minio\_20230602231726.0.0\_amd64.deb -O minio.deb  
 sudo dpkg -i minio.deb

2.- Vamos a crear una carpeta para que se alojen ahí todos los buckets que en un futuro se creen. Ejecutamos:

mkdir ~/minio/

3.- Ejecutamos el servicio de Minio e indicamos la carpeta creada y el puerto para acceder a su web (o  
también llamada consola):

minio server ~/minio --console-address :9090

*NOTA:* Por defecto minio setea sus credenciales de root con los siguientes valores:

MINIO\_ROOT\_USER: minioadmin  
MINIO\_ROOT\_PASSWORD: minioadmin

4.- Luego accedemos a la web de Minio por medio del puerto 9090 e ingresamos las credenciales por defecto  
y listo, tenemos Minio para poder crear buckets y serviría como reemplazo de AWS S3.

Referencia: [Instalar Minio Server y Opcionalmente Minio Client](https://www.youtube.com/watch?v=EsJHf5nUYyA&ab_channel=MinIO)

## Desplegar Proceso Airflow en Google Cloud Platform (GCP) - Google Kubernetes Engine (GKE)

Ahora se procede a instalar los componentes de Airflow para despliegue en GKE, para esto es necesario  
tener presente los siquientes requisitos:

* Tener instalada en su versión más reciente el gcloud (CLI de GCP) para usar los comandos de gcloud en nuestro terminal.  
  Referencia: https://cloud.google.com/sdk/docs/install
* Tener un proyecto creado en GCP (Esto desde la consola de GCP)
* Tener instalado "kubectl" para usar los comandos y manipular nuestro clúster de Kubernetes
* Tener conocimientos sobre Kubernetes y los comandos con kubectl
* Tener los permisos necesarios en tu cuenta de GCP para poder realizar los comandos siguientes  
  (particularmente para los servicios GKE, IAM, Filestore y VMs)

1.- Setear nuestro proyecto con gcloud. Por ejemplo de la siguiente manera:  
  
```text  
gcloud config set project airflow-gke-338120  
  
Donde "airflow-gke-338120" es el ID del proyecto

2.- Crear el clúster en gke por vía de comandos. Por ejemplo:

gcloud container clusters create airflow-cluster \  
--machine-type n1-standard-4 \  
--num-nodes 1 \  
--region "asia-east1-a"  
  
Donde:

* "airflow-cluster" es el nombre del clúster
* "n1-standard-4" es el tipo de instancia que tendrá los nodos del clúster
* "1" será el N° de nodos que tendrá nuestro clúster
* "asia-east1-a" será la región donde crearemos el clúster

Hay que tener en cuento lo siguiente:

* Si nos pide autenticación para crear el clúster, lo hacemos, esto mediante nuestro correo electrónico  
  de gmail o nuestra cuenta en GCP.
* Tener en cuenta los límites o quotas que tenemos para nuestra cuenta y la región donde  
  estamos desplegando nuestro clúster y demás recursos (Esto si estamos usando GCP con el método de quotas)

3.- Una vez creado el clúster, se procede a configurarlo para que tenga los permisos necesarios y usar  
otros recursos de GCP, particularmente el bucket donde se alojan nuestros DAGS.  
Este método habilitará "Workload Identity" tanto para el clúster y sus nodos para que el clúster  
actúe en nombre de las cuentas de servicios que se generarán y estas cuentas serán las que tengan  
los permisos necesarios para utilizar otros recursos GCP. Referencia:  
[Habilitar Workload Identity](https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity?hl=es-419&_ga=2.68907346.-259342969.1690208432" \l "console_1)  
Para esto usamos los siguientes comandos:

# Activamos la configuración de Workload Identity para nuestro clúster creado. Por ejemplo:  
 gcloud container clusters update airflow-cluster2 \  
 --region=asia-east1-a \  
 --workload-pool=airflow-test-394115.svc.id.goog  
   
 \* Donde "airflow-cluster2" es nombre de nuestro clúster, "asia-east1-a" región donde desplegamos   
 y "airflow-test-394115" es el ID del proyecto GCP  
   
 # Actualizamos la configuración de nuestro grupo de nodos para que tenga la configuración del clúster  
 gcloud container node-pools update default-pool \  
 --cluster=airflow-cluster2 \  
 --region=asia-east1-a \  
 --workload-metadata=GKE\_METADATA  
   
 \* Donde "default-pool" es el nombre de nuestro grupo de nodos (default-pool es el nombre por defecto),  
 "airflow-cluster2" es nombre de nuestro clúster y "asia-east1-a" región donde desplegamos  
   
 # Obtenemos las credenciales de nuestro clúster creado y poder usarlo con kubectl en nuestro terminal  
 gcloud container clusters get-credentials airflow-cluster2 \  
 --region=asia-east1-a  
   
 \* Donde "airflow-cluster2" es nombre de nuestro clúster y "asia-east1-a" región donde desplegamos

4.- Luego de configurar nuestro clúster, vamos a crear un namespace y cuenta de servicio de la  
siguiente manera:

# Creamos un namespace, por ejemplo con nombre "bcom-airflow"  
kubectl create namespace bcom-airflow  
  
# Creamos un "service account" con nombre "bcom-airflow"  
kubectl create serviceaccount bcom-airflow \  
 --namespace bcom-airflow

5.- Ahora vamos a crear una cuenta de servicio en IAM de GCP con el mismo nombre que creamos anteriormente  
nuestro "service account" y darle los permisos necesarios:

# Creamos la cuenta de servicio  
gcloud iam service-accounts create bcom-airflow \  
 --project=airflow-test-394115

* Donde "bcom-airflow" es el nombre de la cuenta de servicio IAM y "airflow-test-394115" es el ID del proyecto GCP

# Agregamos a nuestra cuenta creada, los permisos que necesita con el comando:

gcloud projects add-iam-policy-binding airflow-test-394115 \  
--member "serviceAccount:bcom-airflow@airflow-test-394115.iam.gserviceaccount.com" \  
--role "roles/owner"

* Donde "airflow-test-394115" es el ID del proyecto GCP, "bcom-airflow" es el nombre de la cuenta  
  de servicio y "roles/owner" es el rol que tiene nuestra cuenta sobre el proyecto.

# Ahora vamos a asociar esta cuenta a nuestro clúster

gcloud iam service-accounts add-iam-policy-binding bcom-airflow@airflow-test-394115.iam.gserviceaccount.com \  
--role roles/iam.workloadIdentityUser \  
--member "serviceAccount:airflow-test-394115.svc.id.goog[bcom-airflow/bcom-airflow]"

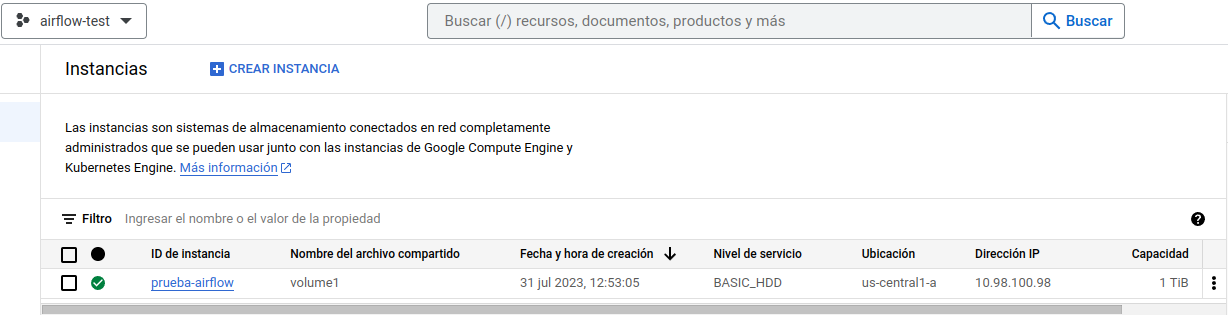
* Donde "bcom-airflow" es el nombre de la cuenta de servicio, "airflow-test-394115" es el ID  
  de nuestro proyecto, además que los "bcom-airflow" es por [namespace]/[serviceaccount]

6.- Actualizamos nuestro "service account" local para que use nuestra cuenta de servicio IAM  
  
```text  
kubectl annotate serviceaccount bcom-airflow \  
 --namespace bcom-airflow \  
 iam.gke.io/gcp-service-account=bcom-airflow@airflow-test-394115.iam.gserviceaccount.com

* Donde "bcom-airflow" hace referencia al "service account" local, namespace y cuenta de servicio IAM.  
  Y "airflow-test-394115" es el ID del proyecto GCP

Listo! Tenemos nuestro clúster configurado a la espera de las configuraciones de los otros recursos:  
Filestore y Buckets.

**Desplegar Servidor NFS con GCP - Filestore**  
  
1.- Primero se tiene que crear una instancia NFS en la consola de GCP. Dependiendo de los  
requerimientos, se puede configurar el tamaño de disco, tipo de disco (HDD o SSD), entre otros.  
  
Para este caso básico, creamos una instancia básica de tipo HDD, con el mínimo de tamaño que es  
de 1Ti, en la región "asia-easat1-a", en la red por default y nombre de archivo compartido "volume1"  
  
Por ejemplo:



\*\*Tener claro y/o apuntar la "Dirección IP" que se muestra en la imagen\*\*  
  
⚠️ Advertencia: Estos pasos siguientes son para la creación de paths requeridos por el proceso  
que esten dentro del servidor NFS.  
  
2.- Ahora vamos a crear una instancia básica (Virtual machine) para poder acceder a nuestro NFS  
y crear los paths necesarios que usará nuestro proceso  
  
```text  
# Crear nuestra instancia en la región donde esta el NFS:  
gcloud compute instances create bastion --zone asia-east1-a

* Donde "bastion" es el nombre de la VM y "asia-east1-a" región donde esta nuestro NFS

# Luego vamos a acceder a la VM con el comando (por SSH):

gcloud compute ssh bastion --zone asia-east1-a

* Donde "bastion" es el nombre de la VM y "asia-east1-a" región donde esta nuestro NFS

3.- Ahora que estamos dentro de la VM, ejecutamos los siguientes comandos:  
  
```text  
# Actualizamos e instalamos el paquete necesario

* sudo -i
* apt update -y
* apt install nfs-common -y

# Creamos una carpeta donde montaremos nuestro NFS, montamos y nos colocamos ahi

* mkdir /mnt/volume1 (cualquier en vez de volume1)
* mount 10.98.100.98:/volume1 /mnt/volume1 (10.98.100.98 es la Dirección IP) y "volume1"  
  nombre del archivo compartido
* cd /mnt/volume1

# Una vez que estamos situados dentro de la carpeta compartida, creamos y damos permisos a las siguientes carpetas

* mkdir nfs\_share
* mkdir nfs\_logs
* mkdir nfs\_postgres
* chmod -R 777 nfs\_share
* chmod -R 777 nfs\_logs
* chmod -R 777 nfs\_postgres

! Listo, tenemos nuestro NFS con las carpetas listas para que nuestro proceso las tome.  
  
⚠️ Tener en cuenta lo siguiente:

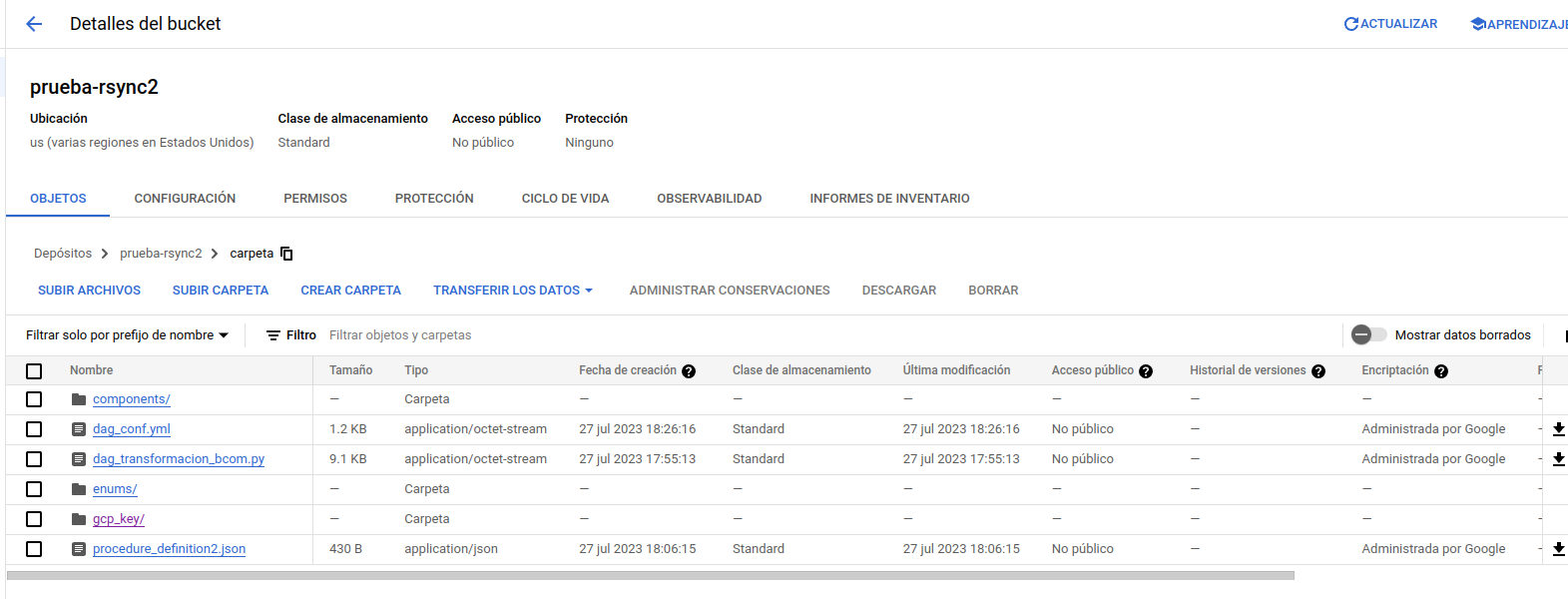
* La dirección IP debe colocarse en el template: "airflow-volumes.yaml" en el parámetro nfs.path  
  de todos los PersistantVolume - PV (son 3)

### **Creación de bucket en GCP - Google Cloud Storage – Buckets**

Luego de tener listo lo anterior, se tiene que crear un bucket donde se alojará nuestro  
código (los DAGS) para que el proceso pueda actualizarlos en caliente.

1.- Por ejemplo creamos el bucket "prueba-rsync2" y dentro de él una carpeta llamada "carpeta" (Puede  
ser cualquier nombre)

2.- Colocamos nuestro código dentro de la carpeta, se verá de la siguiente manera:



⚠️ Advertencia: Tener en cuenta esta ruta creada, para nuestro ejemplo la ruta creada del bucket  
y la carpeta sería: *gs://prueba-rsync2/carpeta*. La cual deberá ser colocada en nuestro  
archivo de configuración (configmap) en el parámetro de: **"GCS\_DAGS\_DIR"**

Listo. Con esto se ha configurado todos los servicios requeridos para levantar nuestro proceso

### Desplegando Airflow con GKE

⚠️ Ojo: Antes de desplegar nuestros componentes, hay que asegurarnos que cumple con los  
recursos de nuestro clúster (en cuestión de vCPU y memoria RAM), para esto hay que validar  
los templates que son de tipo despliegue (deployment), estos son los que en su nombre termina  
con la palabra *"deployment"*: "airflow-scheduler-deployment,yaml", "airflow-webserver-deployment.yaml",  
"postgres-deployment.yaml" y "sync-dags-deployment-gcs" (para este caso que estamos en GCP).

Además del archivo "airflow-envvars-configmap.yaml" que contiene los recursos para los workers.

Todos los templates mencionados tienen sus parámetros "containers.resources.limits" o  
"containers.resources.requests" las cuales indican el máximo y mínimo, respectivamente.

1.- Luego de tener ya configurado nuestro Bucket y NFS (Filestore), podemos desplegar nuestro sistema  
Airflow con Kubernetes. Para esto nos situamos dentro de la carpeta **"deploy-k8"** y ejecutamos  
el siguiente comando:

sh script-apply.sh

Esto arrancará todos los componentes y dentro de unos segundos o minutos estará todo arriba,  
listo para su uso.

2.- Para validar que están todos los componentes arriba, se ejecuta el siguiente comando:

kubectl get pods -n bcom-airflow

Deberá mostrarse algo similar a esto:

