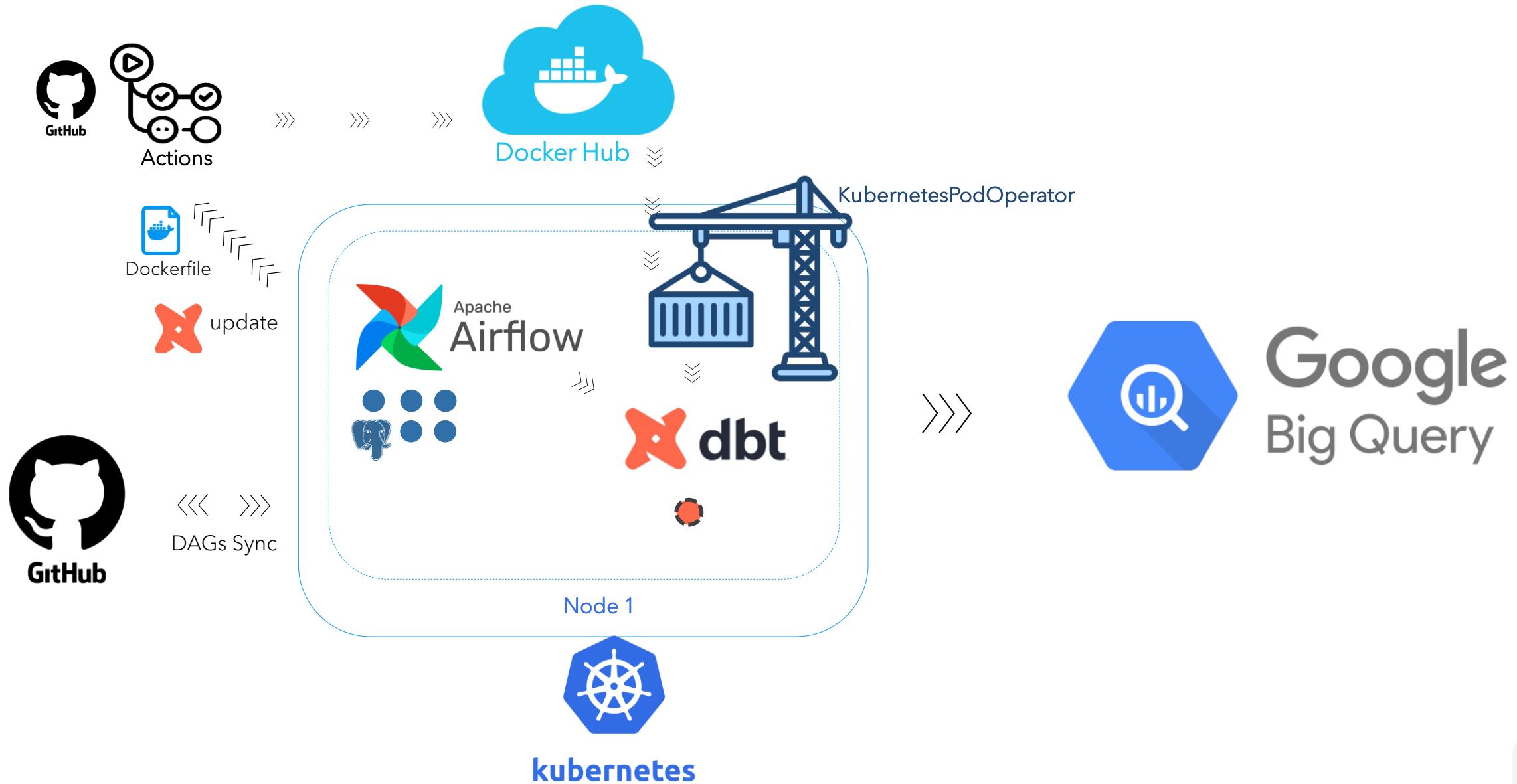


Automated BigQuery Transformations with dbt, Airflow, Kubernetes, and GitHub Actions

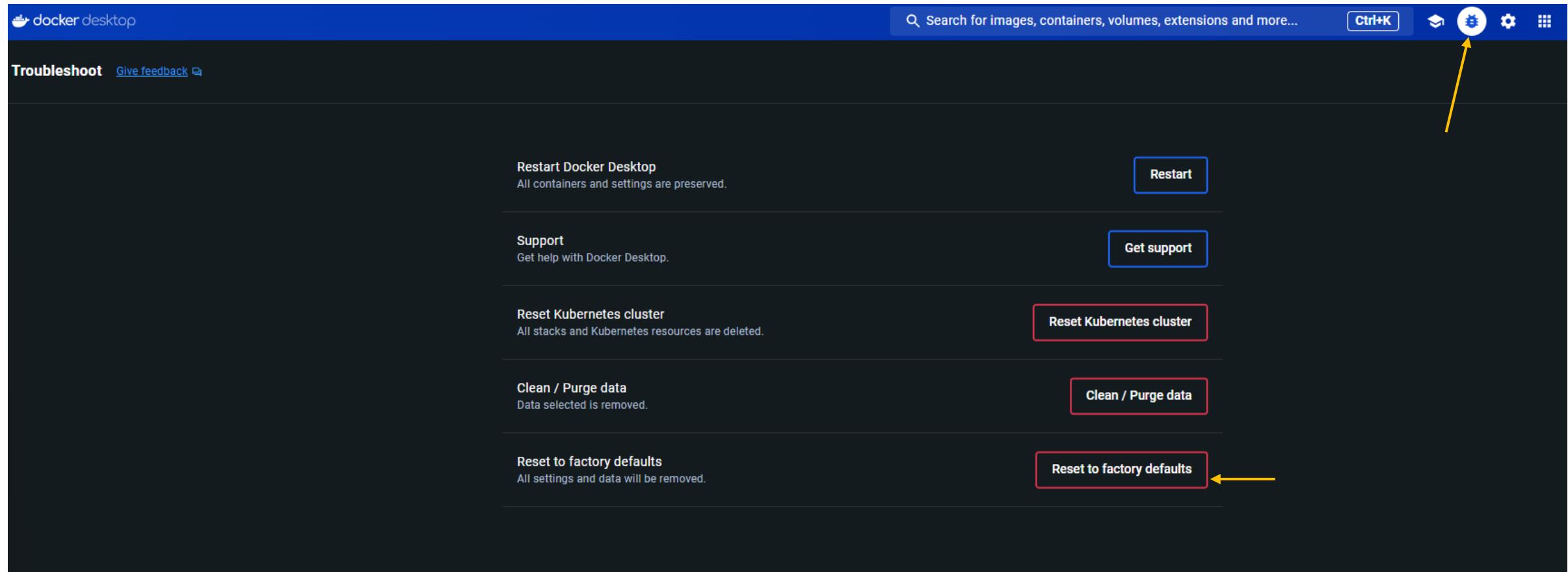


WORKFLOW

1. **Initialize Kubernetes:** Start by activating the Kubernetes cluster with docker desktop, setting the stage for the data pipeline.
2. **Set Up Tools:** Install key components: ***kubectl*** for command-line operations and ***Helm*** for package management.
3. **Configure Kubernetes Dashboard:** Establish a user interface for Kubernetes as a pod, enabling efficient management of cluster resources.
4. **Deploy Airflow Pods:** Roll out Airflow pods within Kubernetes, laying the groundwork for workflow orchestration.
5. **Establish Log Persistence:** Implement a system to ensure that logs are maintained persistently, critical for troubleshooting and analysis. Using ***PersistentVolumeClaim*** and ***ClusterRoleBinding***.
6. **Install dbt-BigQuery:** Seamlessly set up **dbt** for **BigQuery** to leverage powerful data transformation capabilities.
7. **Set up Big Query -GCP**
8. **Synchronize DAGs with GitHub:** Enable real-time updates of Directed Acyclic Graphs (DAGs) via GitHub, ensuring the workflow reflects the latest codebase.
9. **Create the dbt model and its DAGs**
10. **Set up Docker Hub:** create or log in your account and create a repository, making available a space where push the tagged image.
11. **Dockerize the dbt Project:** Package dbt project into a container, readying them for deployment, through ***dockerfile*** to ***Github actions*** where is build and pushed to ***Docker hub***.
12. **Orchestrate CI/CD with GitHub Actions:** Automate the integration and deployment process, streamlining updates and changes to the pipeline. Recieving changes into the ***dbt project***, triggers the **build** and **push** of the **dockerfile** containing the latest project version and making it available in docker hub to Airflow's DAG from Kubernetes.
13. **Run DAG Full workflow:** Activate the full automation potential by executing the DAG from Kubernetes, where the ***KubernetesPodOperator*** **fetches the latest Docker image from Docker Hub**. This image, equipped with **dbt**, runs the complete sequence—***dbt run, dbt seed, and dbt test***. A **temporary pod** is spun up for the duration of the transformations, with the results persisting in BigQuery, exemplifying end-to-end automation in action.

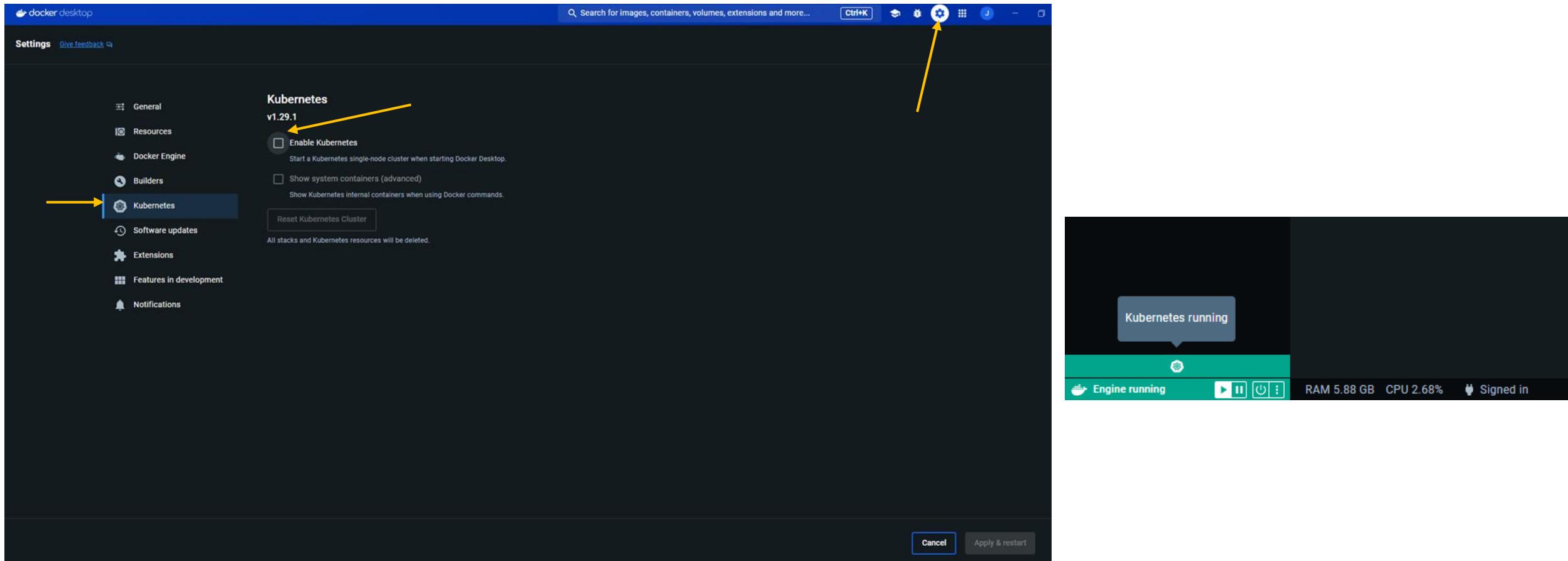
1. INITIALIZE KUBERNETES

- Open Docker Desktop as Admin
- Reset to factory defaults



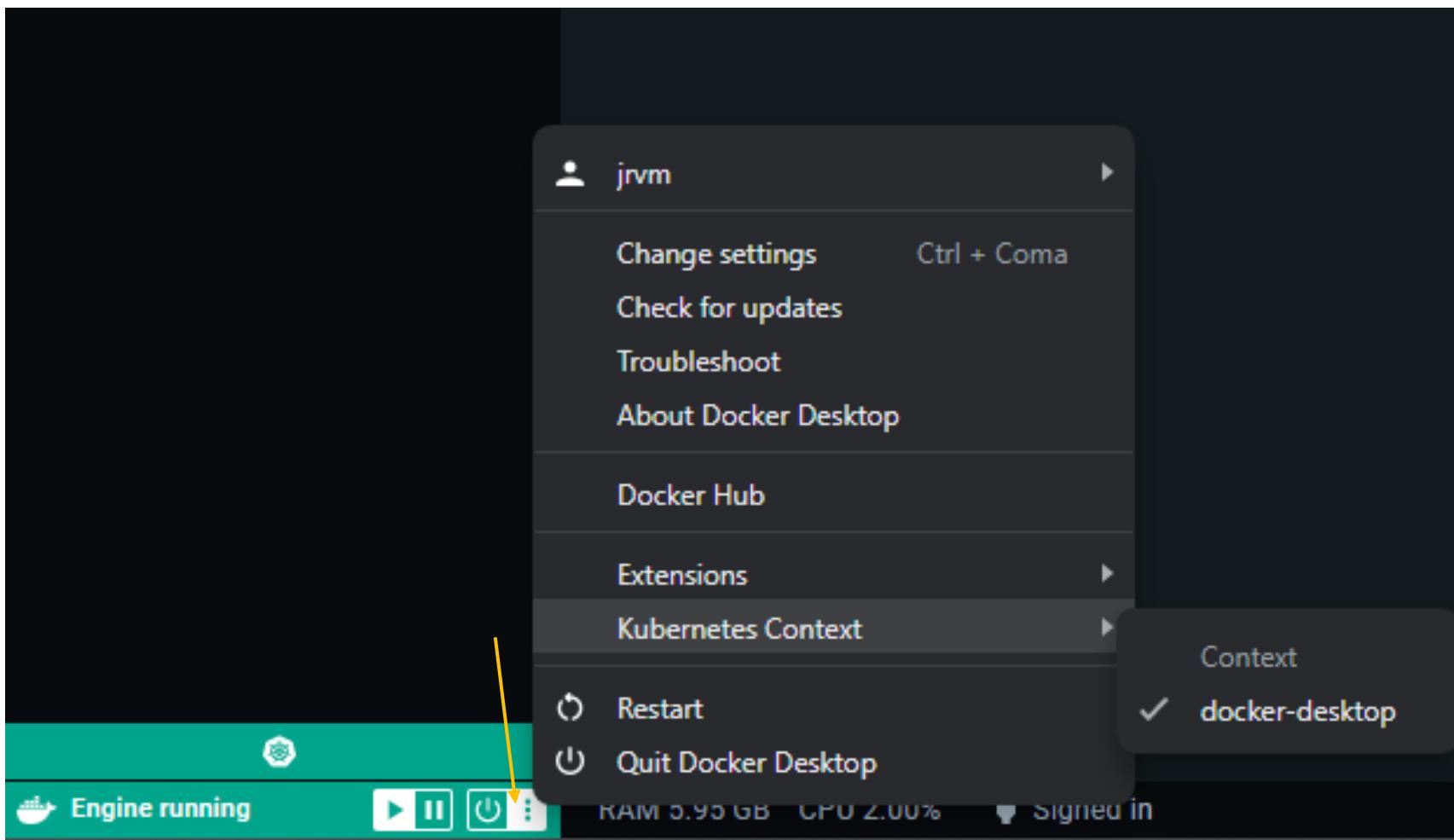
1. INITIALIZE KUBERNETES

- Enable Kubernetes
- Apply & restart
- Wait until running-see bottom left



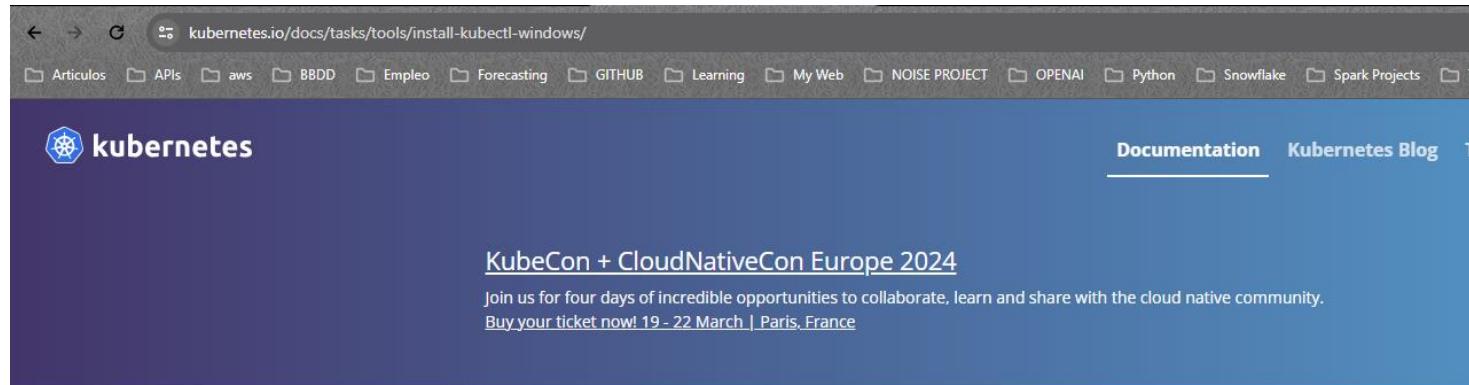
1. INITIALIZE KUBERNETES

- Ensure the kubernetes context is set as Docker-desktop



2. SET UP TOOLS

- **Kubectl:** follow the great official guide -> <https://kubernetes.io/docs/tasks/tools/>
- In my case I choosed Windows



A screenshot of the left sidebar menu on the Kubernetes Documentation site. It includes a search bar, a navigation tree, and a list of specific documentation pages. The navigation tree includes 'Documentation', 'Getting started', 'Concepts', 'Tasks' (which is expanded), and 'Install Tools' (which is also expanded). Under 'Install Tools', there are links for 'Install and Set Up kubectl on Linux', 'Install and Set Up kubectl on macOS', and 'Install and Set Up kubectl on Windows'. Other sections in the sidebar include 'Administer a Cluster', 'Configure Pods and Containers', and 'Monitoring, Logging, and Debugging'.

[Kubernetes Documentation](#) / [Tasks](#) / [Install Tools](#) / [Install and Set Up kubectl on Windows](#)

Install and Set Up kubectl on Windows

Before you begin

You must use a kubectl version that is within one minor version difference of your cluster. For example, a v1.29 client can communicate with v1.28, v1.29, and v1.30 control planes. Using the latest compatible version of kubectl helps avoid unforeseen issues.

Install kubectl on Windows

The following methods exist for installing kubectl on Windows:

- [Install kubectl binary with curl on Windows](#)
- [Install on Windows using Chocolatey, Scoop, or winget](#)

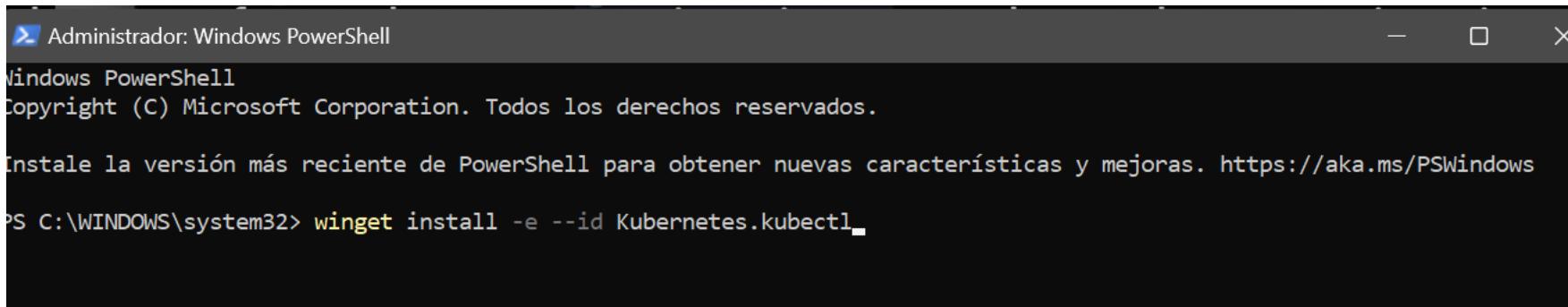
2. SET UP TOOLS

- **Kubectl:** follow the great official guide -> <https://kubernetes.io/docs/tasks/tools/>

> winget install -e --id Kubernetes.kubectl

> Type : Y

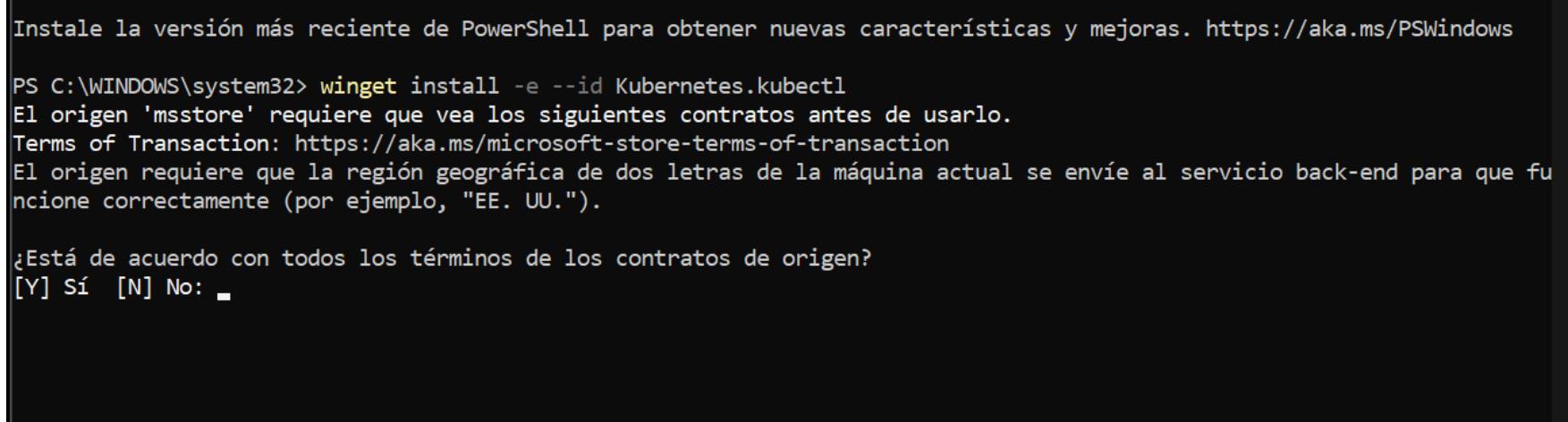
- Wait for downloading



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\WINDOWS\system32> winget install -e --id Kubernetes.kubectl
```



```
Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\WINDOWS\system32> winget install -e --id Kubernetes.kubectl
El origen 'msstore' requiere que vea los siguientes contratos antes de usarlo.
Terms of Transaction: https://aka.ms/microsoft-store-terms-of-transaction
El origen requiere que la región geográfica de dos letras de la máquina actual se envíe al servicio back-end para que funcione correctamente (por ejemplo, "EE. UU.").

¿Está de acuerdo con todos los términos de los contratos de origen?
[Y] Sí [N] No: 
```

2. SET UP TOOLS

- **Kubectl:** follow the great official guide -> <https://kubernetes.io/docs/tasks/tools/>
- Wait for downloading

```
Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows
PS C:\WINDOWS\system32> winget install -e --id Kubernetes.kubectl
El origen 'msstore' requiere que vea los siguientes contratos antes de usarlo.
Terms of Transaction: https://aka.ms/microsoft-store-terms-of-transaction
El origen requiere que la región geográfica de dos letras de la máquina actual se envíe al servicio back-end para que funcione correctamente (por ejemplo, "EE. UU.").

¿Está de acuerdo con todos los términos de los contratos de origen?
[Y] Sí [N] No: y
Encontrado kubectl [Kubernetes.kubectl] Versión 1.29.0
El propietario de esta aplicación le concede una licencia.
Microsoft no es responsable, ni tampoco concede ninguna licencia de paquetes de terceros.
Descargando https://dl.k8s.io/release/v1.29.0/bin/windows/amd64/kubectl.exe
[██████████] 48.6 MB / 48.6 MB
El hash del instalador se verificó correctamente
Iniciando instalación de paquete...
Alias de línea de comandos agregado: "kubectl"
Variable de entorno PATH modificada; reinicie el shell para usar el nuevo valor.
Instalado correctamente
PS C:\WINDOWS\system32> -
```

2. SET UP TOOLS

- **Kubectl:** follow the great official guide -> <https://kubernetes.io/docs/tasks/tools/>
- Confirm installed versión:

> kubectl version --client

```
PS C:\WINDOWS\system32> winget install -e --id Kubernetes.kubectl
El origen 'msstore' requiere que vea los siguientes contratos antes de usarlo.
Terms of Transaction: https://aka.ms/microsoft-store-terms-of-transaction
El origen requiere que la región geográfica de dos letras de la máquina actual se envíe al servicio back-end para que funcione correctamente (por ejemplo, "EE. UU.").

¿Está de acuerdo con todos los términos de los contratos de origen?
[Y] Sí [N] No: y
Encontrado kubectl [Kubernetes.kubectl] Versión 1.29.0
El propietario de esta aplicación le concede una licencia.
Microsoft no es responsable, ni tampoco concede ninguna licencia de paquetes de terceros.
Descargando https://dl.k8s.io/release/v1.29.0/bin/windows/amd64/kubectl.exe
[██████████] 48.6 MB / 48.6 MB
El hash del instalador se verificó correctamente
Iniciando instalación de paquete...
Alias de línea de comandos agregado: "kubectl"
Variable de entorno PATH modificada; reinicie el shell para usar el nuevo valor.
Instalado correctamente
PS C:\WINDOWS\system32> kubectl version --client
Client Version: v1.29.1
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
PS C:\WINDOWS\system32>
```

2. SET UP TOOLS

- **Kubectl:** follow the great official guide -> <https://kubernetes.io/docs/tasks/tools/>
- As admin in powershell:
- Access to user directory
- Create the directory “.kube” (if it not exist already)
 - > mkdir .kube
 - > cd .kube
 - > New-Item config –type file

```
PS C:\WINDOWS\system32> cd $env:USERPROFILE
PS C:\Users\jrver> mkdir .kube
mkdir : Ya existe un elemento con el nombre especificado: C:\Users\jrver\.kube.
En línea: 1 Carácter: 1
+ mkdir .kube
+ ~~~~~
+ CategoryInfo          : ResourceExists: (C:\Users\jrver\.kube:String) [New-Item], IOException
+ FullyQualifiedErrorId : DirectoryExist,Microsoft.PowerShell.Commands.NewItemCommand

PS C:\Users\jrver> cd .kube
>> New-Item config -type file
New-Item : El archivo 'C:\Users\jrver\.kube\config' ya existe.
En línea: 2 Carácter: 1
+ New-Item config -type file
+ ~~~~~
+ CategoryInfo          : WriteError: (C:\Users\jrver\.kube\config:String) [New-Item], IOException
+ FullyQualifiedErrorId : NewItemIOError,Microsoft.PowerShell.Commands.NewItemCommand

PS C:\Users\jrver\.kube> kubectl cluster-info
Kubernetes control plane is running at https://kubernetes.docker.internal:6443
CoreDNS is running at https://kubernetes.docker.internal:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
PS C:\Users\jrver\.kube>
```

2. SET UP TOOLS

- **Kubectl:** follow the great official guide -> <https://kubernetes.io/docs/tasks/tools/>
- Check kubernetes cluster info:
 - > Kubectl cluster-info
- Further debug and diagnose cluster problems
 - > Kubectl cluster-info dump

```
PS C:\Users\jrvrver\.kube> kubectl cluster-info
Kubernetes control plane is running at https://kubernetes.docker.internal:6443
CoreDNS is running at https://kubernetes.docker.internal:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
PS C:\Users\jrvrver\.kube> kubectl cluster-info dump
{
  "kind": "NodeList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "3495"
  },
  "items": [
    {
      "metadata": {
        "name": "docker-desktop",
        "uid": "3ecea5f7-68e1-4f16-bc70-142d69fdb17f",
        "resourceVersion": "3465",
        "creationTimestamp": "2024-03-04T10:01:59Z",
        "labels": {
          "beta.kubernetes.io/arch": "amd64",
          "beta.kubernetes.io/os": "linux",
          "kubernetes.io/arch": "amd64",
          "kubernetes.io/hostname": "docker-desktop",
          "kubernetes.io/os": "linux",
          "node-role.kubernetes.io/control-plane": "",
          "node.kubernetes.io/exclude-from-external-load-balancers": ""
        },
        "annotations": {
          "kubeadm.alpha.kubernetes.io/cri-socket": "unix:///var/run/cri-dockerd.sock",
          "node.alpha.kubernetes.io/ttl": "0",
          "volumes.kubernetes.io/controller-managed-attach-detach": "true"
        }
      },
      "spec": {},
      "status": {
        "capacity": {
```

2. SET UP TOOLS

- **Helm:** follow the great official guide -> <https://helm.sh/docs/intro/install/>
 - Choose the better way to install it according to your OS and preferences
 - In my case I performed **Windows installation with chocolatey**

The image shows two side-by-side screenshots. On the left is the official Helm website (helm.sh). It features a top navigation bar with links for Home, Docs, Charts, Blog, and Community. Below this is a large logo with the word 'HELM' and a crown icon. To the right of the logo is the text 'The package manager for Kubernetes'. A callout box at the bottom states 'Helm is the best way to find, share, and use software built for Kubernetes.' On the right is a screenshot of the 'Docs' section of the Helm website, specifically the 'Installing Helm' page. This page includes a sidebar with navigation links like Docs Home, Introduction, Quickstart Guide, and How-To. The main content area has a heading 'Installing Helm' and a paragraph explaining the guide's purpose. It also includes sections for 'From The Helm Project' and 'From the Binary Releases'.

2. SET UP TOOLS

- **Helm:** follow the great official guide -> <https://helm.sh/docs/intro/install/>
 - Choose the better way to install it according to your OS and preferences
 - In my case I performed **Windows installation with chocolatey**
 - > Choco install kubernetes-helm
 - > Y

```
    "metadata": {  
        "resourceVersion": "3495"  
    },  
    "items": []  
}  
PS C:\Users\jrver\.kube> choco install kubernetes-helm  
Chocolatey v2.0.0  
Installing the following packages:  
kubernetes-helm  
By installing, you accept licenses for the packages.  
Progress: Downloading kubernetes-helm 3.14.1... 100%  
  
kubernetes-helm v3.14.1 [Approved]  
kubernetes-helm package files install completed. Performing other installation steps.  
The package kubernetes-helm wants to run 'chocolateyInstall.ps1'.  
Note: If you don't run this script, the installation will fail.  
Note: To confirm automatically next time, use '-y' or consider:  
choco feature enable -n allowGlobalConfirmation  
Do you want to run the script?([Y]es/[A]ll - yes to all/[N)o/[P]rint): -
```

2. SET UP TOOLS

- **Helm:** follow the great official guide -> <https://helm.sh/docs/intro/install/>
- Wait for installing

```
PS C:\Users\jrver\.kube> choco install kubernetes-helm
Chocolatey v2.0.0
Installing the following packages:
kubernetes-helm
By installing, you accept licenses for the packages.
Progress: Downloading kubernetes-helm 3.14.1... 100%

kubernetes-helm v3.14.1 [Approved]
kubernetes-helm package files install completed. Performing other installation steps.
The package kubernetes-helm wants to run 'chocolateyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N)o/[P]rint): y

Downloading kubernetes-helm 64 bit
  from 'https://get.helm.sh/helm-v3.14.1-windows-amd64.zip'
Progress: 100% - Completed download of C:\Users\jrver\AppData\Local\Temp\chocolatey\kubernetes-helm\
Download of helm-v3.14.1-windows-amd64.zip (15.76 MB) completed.
Hashes match.
Extracting C:\Users\jrver\AppData\Local\Temp\chocolatey\kubernetes-helm\3.14.1\helm-v3.14.1-windows-
C:\ProgramData\chocolatey\lib\kubernetes-helm\tools
ShimGen has successfully created a shim for helm.exe
The install of kubernetes-helm was successful.
Software installed to 'C:\ProgramData\chocolatey\lib\kubernetes-helm\tools'

Chocolatey installed 1/1 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Enjoy using Chocolatey? Explore more amazing features to take your
experience to the next level at
  https://chocolatey.org/compare
PS C:\Users\jrver\.kube> _
```

2. SET UP TOOLS

- **Helm:** follow the great official guide -> <https://helm.sh/docs/intro/install/>
- Confirm instalation –pointing to .kube directory
 > helm version

```
experience to the next level at
https://chocolatey.org/compare
PS C:\Users\jrver\.kube> helm --version
Error: unknown flag: --version
PS C:\Users\jrver\.kube> helm version
version.BuildInfo{Version:"v3.14.1", GitCommit:"e8858f8696b144ee7c533bd9d49a353ee6c4b98d", GitTreeState:"clean", GoVersion:"go1.21.7"}
PS C:\Users\jrver\.kube> kubectl version
Client Version: v1.29.1
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.29.1
PS C:\Users\jrver\.kube> ■
```

2. SET UP TOOLS

- Check Kubernetes Cluster info and get current contexts

> kubectl cluster-info
> kubectl config get-contexts

```
Enjoy using Chocolatey? Explore more amazing features to take your
experience to the next level at
https://chocolatey.org/compare
PS C:\Users\jrver\.kube> helm --version
Error: unknown flag: --version
PS C:\Users\jrver\.kube> helm version
version.BuildInfo{Version:"v3.14.1", GitCommit:"e8858f8696b144ee7c533bd9d49a353ee6c4b98d", GitTreeState:"clean", GoVersion:"go1.21.7"}
PS C:\Users\jrver\.kube> kubectl version
Client Version: v1.29.1
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.29.1
PS C:\Users\jrver\.kube> kubectl cluster-info
Kubernetes control plane is running at https://kubernetes.docker.internal:6443
CoreDNS is running at https://kubernetes.docker.internal:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
PS C:\Users\jrver\.kube> kubectl config get-contexts
CURRENT      NAME           CLUSTER          AUTHINFO        NAMESPACE
*   docker-desktop   docker-desktop   docker-desktop
PS C:\Users\jrver\.kube>
```

3. CONFIGURE KUBERNETES DASHBOARD

- Pull the recommended .yaml for a default Kubernetes dashboard

```
> kubectl apply -f  
https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml
```

- After that it is needed to forwarding the ports and able to see the dashboard on the browser
 - Open a new terminal or leave this terminal running, otherwise the forwarding will be interrupted.
>kubectl proxy

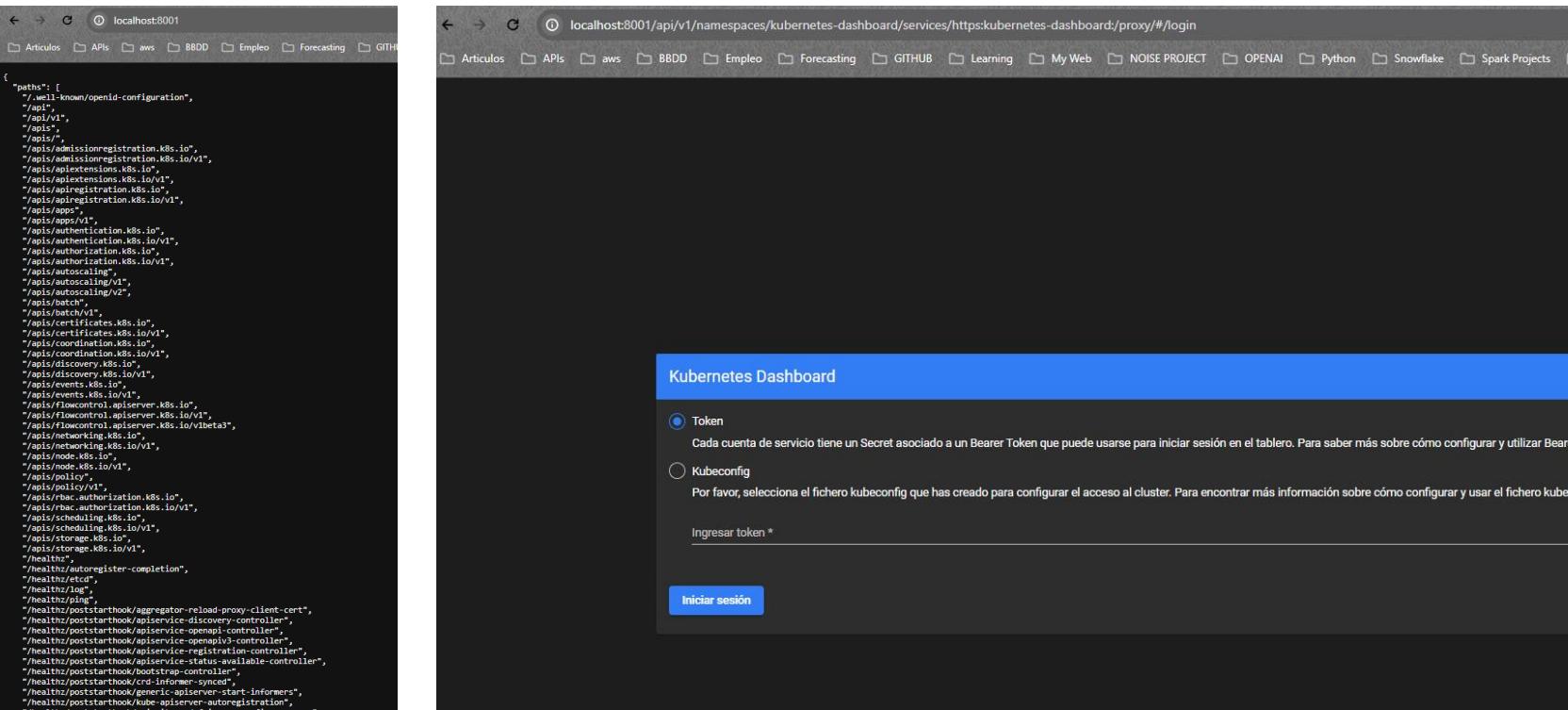
```
PS C:\Users\jrver\.kube> kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml  
namespace/kubernetes-dashboard created  
serviceaccount/kubernetes-dashboard created  
service/kubernetes-dashboard created  
secret/kubernetes-dashboard-certs created  
secret/kubernetes-dashboard-csrf created  
secret/kubernetes-dashboard-key-holder created  
configmap/kubernetes-dashboard-settings created  
role.rbac.authorization.k8s.io/kubernetes-dashboard created  
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created  
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created  
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created  
deployment.apps/kubernetes-dashboard created  
service/dashboard-metrics-scraper created  
deployment.apps/dashboard-metrics-scraper created  
PS C:\Users\jrver\.kube> kubectl proxy  
Starting to serve on 127.0.0.1:8001
```

3. CONFIGURE KUBERNETES DASHBOARD

- Go to localhost:8001 to see the paths you are able to see from this dashboard
- Paste the following command to get Access the dashboard:

> <http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/>

- The authentication screen is showed where to pass the token is asked



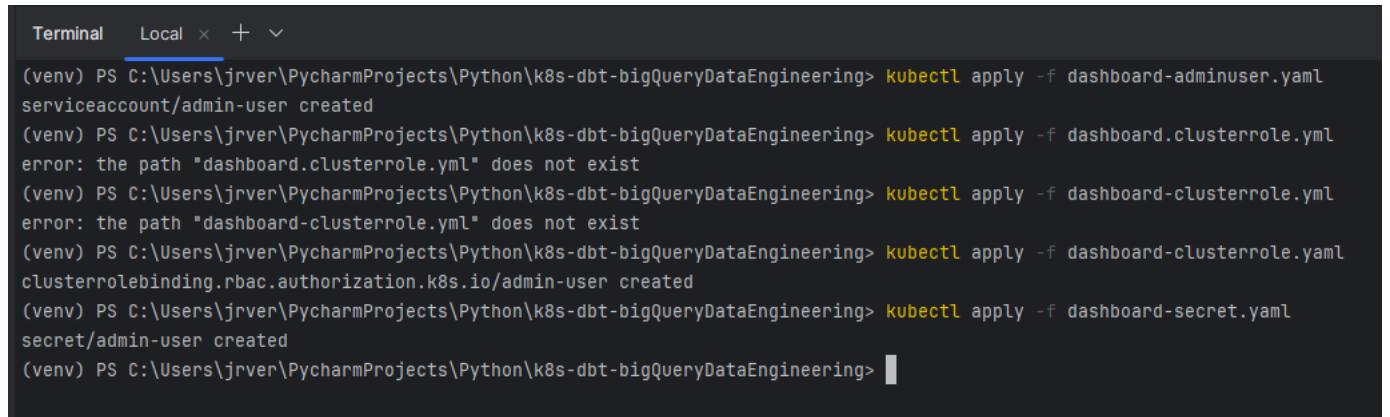
3. CONFIGURE KUBERNETES DASHBOARD

- Before getting the Token, create and apply the dashboard services:

```
>kubectl apply -f dashboard-adminuser.yaml
```

```
>kubectl apply -f dashboard-clusterrole.yaml
```

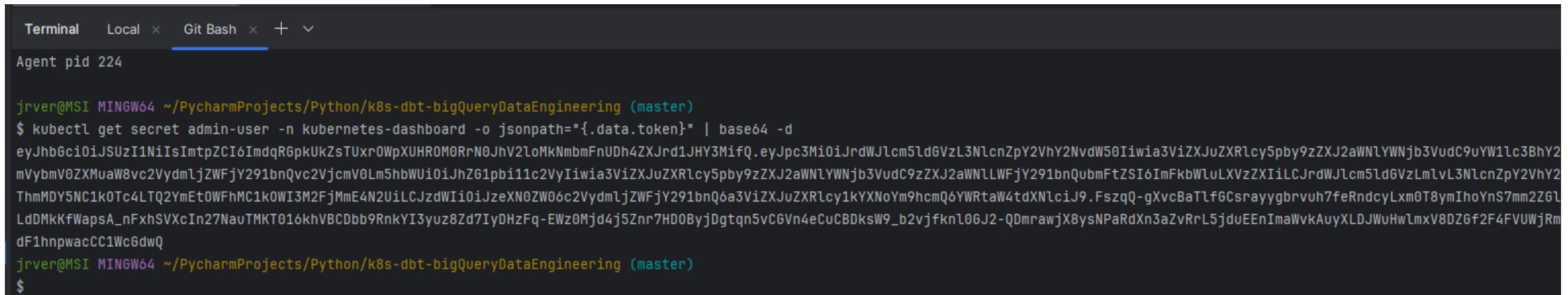
```
>kubectl apply -f dashboard-secret.yaml
```



```
Terminal Local + ▾
(venv) PS C:\Users\jrver\PycharmProjects\Python\k8s-dbt-bigQueryDataEngineering> kubectl apply -f dashboard-adminuser.yaml
serviceaccount/admin-user created
(venv) PS C:\Users\jrver\PycharmProjects\Python\k8s-dbt-bigQueryDataEngineering> kubectl apply -f dashboard.clusterrole.yaml
error: the path "dashboard.clusterrole.yaml" does not exist
(venv) PS C:\Users\jrver\PycharmProjects\Python\k8s-dbt-bigQueryDataEngineering> kubectl apply -f dashboard-clusterrole.yaml
error: the path "dashboard-clusterrole.yaml" does not exist
(venv) PS C:\Users\jrver\PycharmProjects\Python\k8s-dbt-bigQueryDataEngineering> kubectl apply -f dashboard-clusterrole.yaml
clusterrolebinding.rbac.authorization.k8s.io/admin-user created
(venv) PS C:\Users\jrver\PycharmProjects\Python\k8s-dbt-bigQueryDataEngineering> kubectl apply -f dashboard-secret.yaml
secret/admin-user created
(venv) PS C:\Users\jrver\PycharmProjects\Python\k8s-dbt-bigQueryDataEngineering>
```

- Once created, run the get secret command, copy and paste it into the authentication placeholder.

```
>kubectl get secret admin-user -n kubernetes-dashboard -o jsonpath="{.data.token}" | base64 -d
```



```
Terminal Local + ▾ Git Bash + ▾
Agent pid 224

jrver@MSI MINGW64 ~/PycharmProjects/Python/k8s-dbt-bigQueryDataEngineering (master)
$ kubectl get secret admin-user -n kubernetes-dashboard -o jsonpath=".data.token" | base64 -d
eyJhbGciOiJSUzI1NiIsImtpZCI6ImdqRGpkUKzsTUxr0WpxUHR0M0RrN0JhV2loMkNmhmFnUDh4ZXJrd1JHY3MifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcenZpY2VhY2NvdW50Iiwia3ViZXJuZXRLcy5pbby9zZXJ2aWNLYWNjb3VudC9uYW1lc3BhY2mVybmbV0ZXMuaw8vc2VydmljZWfjY291bnQvc2VjcmV0Lm5hbWUiOijhZG1pbii1c2VyIiwi3ViZXJuZXRLcy5pbby9zZXJ2aWNLYWNjb3VudC9zZXJ2aWNLLWFjY291bnQubmFtZSI6ImFkbWluLVzzXiilCJrdWJlcm5ldGVzLmlvL3NlcenZpY2VhY2ThmMDY5NC1k0Tc4LTQ2YmEtOWFhMC1k0WI3M2FjMmE4N2UiLCJzdWIiOijzeXN0Zw06c2VydmljZWfjY291bnQ6a3ViZXJuZXRLcy1KYXNoYm9hcmQ6YWRtaW4tdXNlcij9.FszqQ-gXvcBaTlfG0Csrayygrvuh7feRndcyLxm0T8ymIhoYnS7mm2ZGlLdDMkKfWapsA_nFxhSVXcIn27NauTMKT016khVBCDbb9RnkYI3yuz8Zd7IyDHzfq-EWz0Mjds4j5zn7HD0ByjDgtqn5vCGVn4eCuCBDksW9_b2vjfknl06J2-QDmrawjX8ysNPaRdxN3aZvRrL5jduEEInmaWvkAuyXLDJWuHwlmxV8DZGf2F4FVUWjRmdF1hnpwacCC1WcGdwQ
jrver@MSI MINGW64 ~/PycharmProjects/Python/k8s-dbt-bigQueryDataEngineering (master)
$
```

3. CONFIGURE KUBERNETES DASHBOARD

- Copy, paste the token
- Access granted and shows the full dashboard with comprehensive kubernetes, Jobs, pods, services information.

The image shows two screenshots of the Kubernetes Dashboard. The top screenshot is a login screen titled "Kubernetes Dashboard" with two radio button options: "Token" (selected) and "Kubeconfig". It includes a text input field for "Ingresar token *" and a "Iniciar sesión" button. The bottom screenshot shows the main dashboard interface at the URL `localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#/workloads?namespace=kubernetes-dashboard`. The left sidebar has sections for Cargas de trabajo, Service, Configuration y Almacenamiento, and Cluster. The main area displays three large circular metrics: Despliegues (Running: 2), Pods (Running: 2), and Replica Sets (Running: 2). Below these are two tables: "Despliegues" and "Pods", each listing two entries with columns for Nombre, Imágenes, Etiquetas, and Pod status.

Nombre	Imágenes	Etiquetas	Pods	Fecha de creación
dashboard-metrics-scraper	kubernetesui/metrics-scraper:v1.0.8	k8s-app: dashboard-metrics-scraper	1 / 1	an hour ago
kubernetes-dashboard	kubernetesui/dashboard:v2.7.0	k8s-app: kubernetes-dashboard	1 / 1	an hour ago

Nombre	Imágenes	Etiquetas	Nodo	Estado	Reinicios	Utilización de CPU (núcleos)	Utilización de memoria (octetos)	Fecha de creación
dashboard-metrics-scraper-5657497c4c-k9t8s	kubernetesui/metrics-scraper:v1.0.8	k8s-app: dashboard-metrics-scraper pod-template-hash: 5657497c4c	docker-desktop	Running	0	-	-	an hour ago
kubernetes-dashboard-78f87ddfc-fnyv	kubernetesui/dashboard:v2.7.0	k8s-app: kubernetes-dashboard pod-template-hash: 78f87ddfc	docker-desktop	Running	0	-	-	an hour ago

4. DEPLOY AIRFLOW PODS

- Add airflow to the local Helm repository

> helm repo add apache-airflow https://airflow.apache.org

- Update Helm repository to ensure it is up to date after adding airflow

>helm repo update

- Install the airflow into a Kubernetes pod

>helm install airflow apache-airflow/airflow --namespace airflow --create-namespace --debug

```
PS C:\WINDOWS\system32> helm repo add apache-airflow https://airflow.apache.org
"apache-airflow" has been added to your repositories
PS C:\WINDOWS\system32> helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "apache-airflow" chart repository
Update Complete. Happy Helming!®
PS C:\WINDOWS\system32> helm install airflow apache-airflow/airflow --namespace airflow --create-namespace --debug
install.go:214: [debug] Original chart version: ""
install.go:231: [debug] CHART PATH: C:\Users\jrvr\AppData\Local\Temp\helm\Repository\airflow-1.12.0.tgz

client.go:142: [debug] creating 1 resource(s)
client.go:486: [debug] Starting delete for "airflow-broker-url" Secret
client.go:490: [debug] Ignoring delete failure for "airflow-broker-url" /v1, Kind=Secret: secrets "airflow-broker-url" not found
wait.go:66: [debug] beginning wait for 1 resources to be deleted with timeout of 5m0s
client.go:142: [debug] creating 1 resource(s)
client.go:486: [debug] Starting delete for "airflow-fernet-key" Secret
client.go:490: [debug] Ignoring delete failure for "airflow-fernet-key" /v1, Kind=Secret: secrets "airflow-fernet-key" not found
wait.go:66: [debug] beginning wait for 1 resources to be deleted with timeout of 5m0s
client.go:142: [debug] creating 1 resource(s)
client.go:486: [debug] Starting delete for "airflow-redis-password" Secret
client.go:490: [debug] Ignoring delete failure for "airflow-redis-password" /v1, Kind=Secret: secrets "airflow-redis-password" not found
wait.go:66: [debug] beginning wait for 1 resources to be deleted with timeout of 5m0s
client.go:142: [debug] creating 1 resource(s)
client.go:142: [debug] creating 31 resource(s)
client.go:486: [debug] Starting delete for "airflow-run-airflow-migrations" Job
client.go:490: [debug] Ignoring delete failure for "airflow-run-airflow-migrations" batch/v1, Kind=Job: jobs.batch "airflow-run-airflow-migrations" not found
wait.go:66: [debug] beginning wait for 1 resources to be deleted with timeout of 5m0s
client.go:142: [debug] creating 1 resource(s)
client.go:712: [debug] Watching for changes to Job airflow-run-airflow-migrations with timeout of 5m0s
client.go:740: [debug] Add/Modify event for airflow-run-airflow-migrations: ADDED
client.go:779: [debug] airflow-run-airflow-migrations: Jobs active: 0, jobs failed: 0, jobs succeeded: 0
client.go:740: [debug] Add/Modify event for airflow-run-airflow-migrations: MODIFIED
client.go:779: [debug] airflow-run-airflow-migrations: Jobs active: 1, jobs failed: 0, jobs succeeded: 0
```

4. DEPLOY AIRFLOW PODS

- Wait until instalation is finished
- Forwarding port to enable the airflow UI on the browser by localhost 8080
 >kubectl port-forward svc/airflow-webserver 8080:8080 --namespace airflow

```
# Source: airflow/templates/webserver/webserver-ingress.yaml
#####
## Airflow Webserver Ingress
#####
# Source: airflow/templates/webserver/webserver-networkpolicy.yaml
#####
## Airflow Webserver NetworkPolicy
#####
# Source: airflow/templates/webserver/webserver-poddisruptionbudget.yaml
#####
## Airflow Webserver PodDisruptionBudget
#####
# Source: airflow/templates/workers/worker-hpa.yaml
#####
## Airflow Worker HPA
#####
# Source: airflow/templates/workers/worker-kedaautoscaler.yaml
#####
## Airflow Worker KEDA Scaler
#####
# Source: airflow/templates/workers/worker-networkpolicy.yaml
#####
## Airflow Worker NetworkPolicy
#####

NOTES:
Thank you for installing Apache Airflow 2.8.1!

Your release is named airflow.
You can now access your dashboard(s) by executing the following command(s) and visiting the corresponding port at localhost in your browser:

Airflow Webserver: kubectl port-forward svc/airflow-webserver 8080:8080 --namespace airflow
Default Webserver (Airflow UI) Login credentials:
  username: admin
  password: admin
Default Postgres connection credentials:
  username: postgres
  password: postgres
  port: 5432

You can get Fernet Key value by running the following:

echo Fernet Key: $(kubectl get secret --namespace airflow airflow-fernet-key -o jsonpath="{.data.fernet-key}" | base64 --decode)

#####
# WARNING: You should set a static webserver secret key #
#####

You are using a dynamically generated webserver secret key, which can lead to unnecessary restarts of your Airflow components.

Information on how to set a static webserver secret key can be found here:
https://airflow.apache.org/docs/helm-chart/stable/production-guide.html#webserver-secret-key
PS C:\WINDOWS\system32> kubectl port-forward svc/airflow-webserver 8080:8080 --namespace airflow
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

```
Your release is named airflow.
You can now access your dashboard(s) by executing the following command(s) and visiting the corresponding port at localhost in your browser:

Airflow Webserver: kubectl port-forward svc/airflow-webserver 8080:8080 --namespace airflow
Default Webserver (Airflow UI) Login credentials:
  username: admin
  password: admin
Default Postgres connection credentials:
  username: postgres
  password: postgres
  port: 5432

You can get Fernet Key value by running the following:

echo Fernet Key: $(kubectl get secret --namespace airflow airflow-fernet-key -o jsonpath="{.data.fernet-key}" | base64 --decode)

#####
# WARNING: You should set a static webserver secret key #
#####

You are using a dynamically generated webserver secret key, which can lead to unnecessary restarts of your Airflow components.

Information on how to set a static webserver secret key can be found here:
https://airflow.apache.org/docs/helm-chart/stable/production-guide.html#webserver-secret-key
PS C:\WINDOWS\system32> kubectl port-forward svc/airflow-webserver 8080:8080 --namespace airflow
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

4. DEPLOY AIRFLOW PODS

- Access the Airflow UI and type user and pwd in.
- see a message about keys: " Usage of a dynamic webserver secret key detected" to solve this:
>helm show values apache-airflow/airflow > values.yaml

The image consists of two vertically stacked screenshots of the Apache Airflow web interface. The top screenshot shows the 'Sign In' page, which has a 'Sign In' button and two input fields: 'Username' (containing 'admin') and 'Password' (containing 'admin'). The bottom screenshot shows the 'DAGs' page, which includes a yellow banner with the text 'Usage of a dynamic webserver secret key detected. We recommend a static webserver secret key instead. See the Helm Chart Production Guide for more details.' Below the banner, there is a table with columns for 'Owner', 'Runs', 'Schedule', 'Last Run', 'Next Run', 'Recent Tasks', 'Actions', and 'Links'. The table displays 'No results' and shows 'Showing 0-0 of 0 DAGs' at the bottom. Both screenshots are taken from a browser window with the URL 'localhost:8080' visible in the address bar.

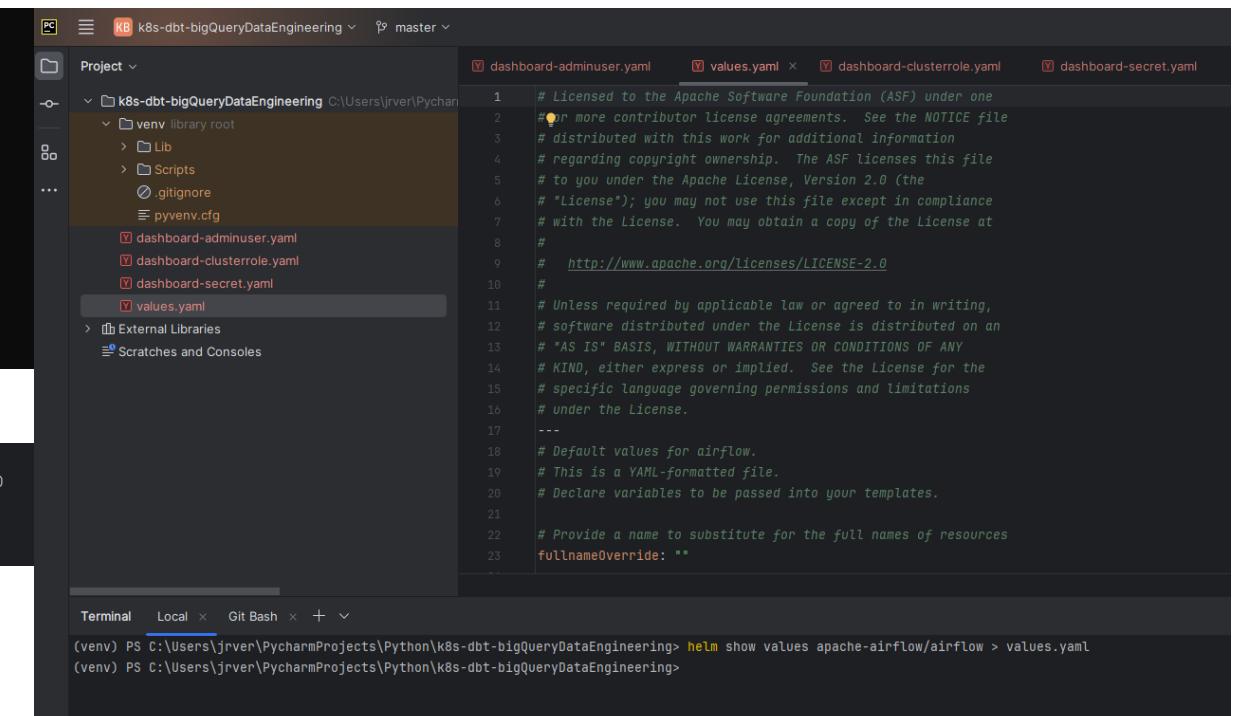
4. DEPLOY AIRFLOW PODS

- Values.yaml is created and show default information which is going to be overwritten.
 - 1. the **Fernet Key**, the command is showed once Airflow pod is installed or upgraded.

```
> echo Fernet Key: $(kubectl get secret --namespace airflow airflow-fernet-key -o jsonpath="{.data.fernet-key}" | base64 --decode)
```

- 2. **the values.yaml** configuration – see next slide

```
NOTES:  
Thank you for installing Apache Airflow 2.8.1!  
  
Your release is named airflow.  
You can now access your dashboard(s) by executing the following command(s) and visiting the corresponding port at localhost in your browser:  
  
Airflow Webserver: kubectl port-forward svc/airflow-webserver 8080:8080 --namespace airflow  
Default Webserver (Airflow UI) Login credentials:  
  username: admin  
  password: admin  
Default Postgres connection credentials:  
  username: postgres  
  password: postgres  
  port: 5432  
  
You can get Fernet Key value by running the following:  
echo Fernet Key: $(kubectl get secret --namespace airflow airflow-fernet-key -o jsonpath="{.data.fernet-key}" | base64 --decode)  
  
#####  
# WARNING: You should set a static webserver secret key #  
#####  
  
You are using a dynamically generated webserver secret key, which can lead to  
https://airflow.apache.org/docs/apache-airflow/stable/configurations.html#dynamic-webserver-secret-key
```



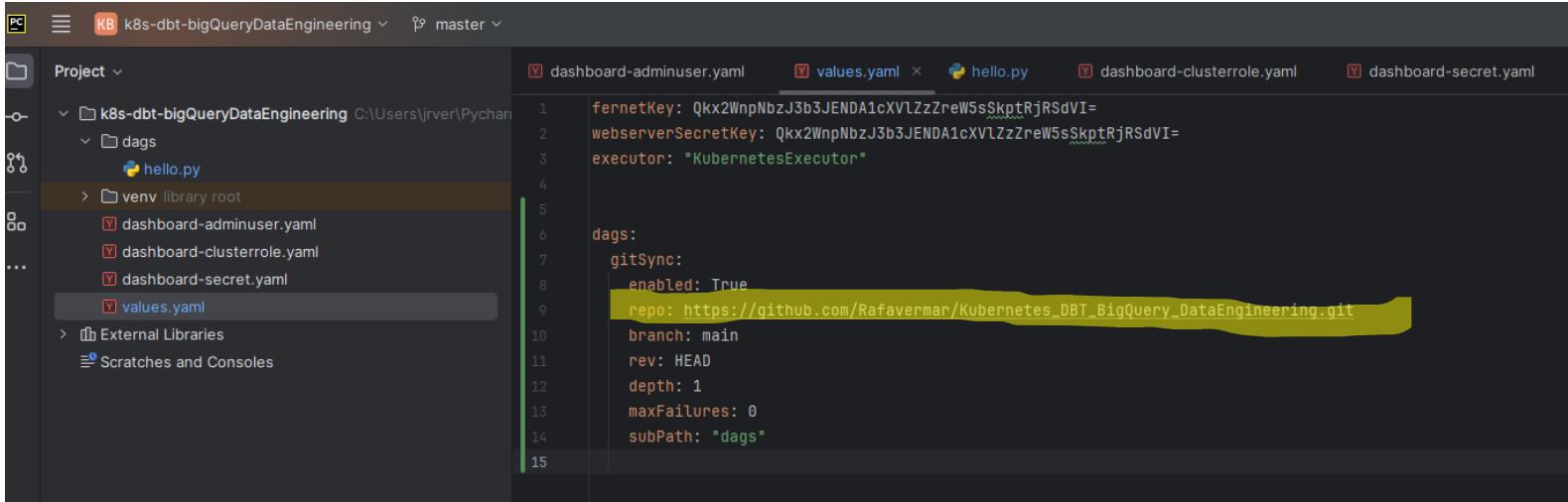
The screenshot shows a PyCharm interface with the project structure on the left and the code editor on the right. The project structure includes a 'k8s-dbt-bigQueryDataEngineering' folder containing a 'venv' folder and several configuration files: 'dashboard-adminuser.yaml', 'values.yaml', 'dashboard-clusterrole.yaml', and 'dashboard-secret.yaml'. The 'values.yaml' file is currently selected in the code editor. The code in 'values.yaml' is as follows:

```
# Licensed to the Apache Software Foundation (ASF) under one  
# or more contributor license agreements. See the NOTICE file  
# distributed with this work for additional information  
# regarding copyright ownership. The ASF licenses this file  
# to you under the Apache License, Version 2.0 (the  
# "License"); you may not use this file except in compliance  
# with the License. You may obtain a copy of the License at  
# http://www.apache.org/licenses/LICENSE-2.0  
#  
# Unless required by applicable law or agreed to in writing,  
# software distributed under the License is distributed on an  
# AS IS BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY  
# KIND, either express or implied. See the License for the  
# specific language governing permissions and limitations  
# under the License.  
---  
# Default values for airflow.  
# This is a YAML-formatted file.  
# Declare variables to be passed into your templates.  
#  
# Provide a name to substitute for the full names of resources  
fullnameOverride: ""
```

At the bottom of the PyCharm interface, there is a terminal window showing the command: `helm show values apache-airflow/airflow > values.yaml`.

4. DEPLOY AIRFLOW PODS

- Values.yaml must look like this.
- You need a github repo to be synchronized with and from where your pushed DAGs will be read by the Airflow pod.



The screenshot shows a PyCharm interface with the project 'k8s-dbt-bigQueryDataEngineering' open. The left sidebar shows files like 'hello.py', 'values.yaml', 'dashboard-adminuser.yaml', 'dashboard-clusterrole.yaml', and 'dashboard-secret.yaml'. The right pane displays the 'values.yaml' file content:

```
1 fernetKey: QKx2WnpNbzb3JENDA1cXVlZzZreW5sSkptRjRSdVI=
2 webserverSecretKey: QKx2WnpNbzb3JENDA1cXVlZzZreW5sSkptRjRSdVI=
3 executor: "KubernetesExecutor"
4
5
6 dags:
7   gitSync:
8     enabled: True
9     repo: https://github.com/Rafavermar/Kubernetes_DBT_BigQuery_DataEngineering.git
10    branch: main
11    rev: HEAD
12    depth: 1
13    maxFailures: 0
14    subPath: "dags"
15
```

- Upgrade Airflow pod:
› helm upgrade --install airflow apache-airflow/airflow --namespace airflow --create-namespace -f values.yaml

```
[venv] PS C:\Users\jrver\PycharmProjects\Python\k8s-dbt-bigQueryDataEngineering> helm upgrade --install airflow apache-airflow/airflow --namespace airflow --create-namespace -f values.yaml
Release "airflow" has been upgraded. Happy Helming!
NAME: airflow
LAST DEPLOYED: Mon Mar  4 17:23:16 2024
NAMESPACE: airflow
STATUS: deployed
REVISION: 2
TEST SUITE: None
NOTES:
Thank you for installing Apache Airflow 2.8.1!

Your release is named airflow.
You can now access your dashboard(s) by executing the following command(s) and visiting the corresponding port at localhost in your browser:

Airflow Webserver:  kubectl port-forward svc/airflow-webserver 8080:8080 --namespace airflow
Default Webserver (Airflow UI) Login credentials:
  username: admin
  password: admin
Default Postgres connection credentials:
  username: postgres
  password: postgres
  port: 5432

You can get Fernet Key value by running the following:

  echo Fernet Key: $(kubectl get secret --namespace airflow air-flow-fernet-key -o jsonpath='{.data.fernet-key}' | base64 --decode)

WARNING:
  Kubernetes workers task logs may not persist unless you configure log persistence or remote logging!
  Logging options can be found at: https://airflow.apache.org/docs/heptilic/1.10.0/manage-logs.html
  (This warning can be ignored if logging is configured with environment variables or secrets backend)
```

4. DEPLOY AIRFLOW PODS

- Forward ports again
- See Airflow UI without any message

```
Handling connection for 8080
E0304 17:25:56.876630 25052 portforward.go:409] an error occurred forwarding 8080 -> 8080: error forwarding
ce6b8bf703d2438b07af8a32fc9e8b97fa851c4a54b9256d96)
error: lost connection to pod
PS C:\WINDOWS\system32> kubectl port-forward svc/airflow-webserver 8080:8080 --namespace airflow
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

The screenshot shows the Airflow web interface at `localhost:8080/home`. The top navigation bar includes links for Artículos, APIs, AWS, BBDD, Empleo, Forecasting, GITHUB, Learning, My Web, NOISE PROJECT, OPENAI, Python, Snowflake, Spark Projects, TFM, Tsystems, Utilities, Water Project, and Historial. The user is logged in, as indicated by the profile icon and "AU". The timestamp is 16:28 UTC.

The main content area is titled "DAGs". It features a toolbar with filters: All (0), Active (0), Paused (0), Running (0), Failed (0), and a search bar for "Search DAGs". There is also an "Auto-refresh" button with a refresh icon.

The table below has columns for DAG (sorted by name), Owner, Runs, Schedule, Last Run, Next Run, Recent Tasks, Actions, and Links. A message "No results" is displayed. At the bottom, there are navigation icons for first, previous, next, and last pages, and a status message "Showing 0-0 of 0 DAGs".

4. DEPLOY AIRFLOW PODS

- In Airflow UI go to Cluster Activity to see further information about Dag and cluster performance.
- In Kubernetes Dashboard confirm the pods are running as expected.

Cargas de trabajo

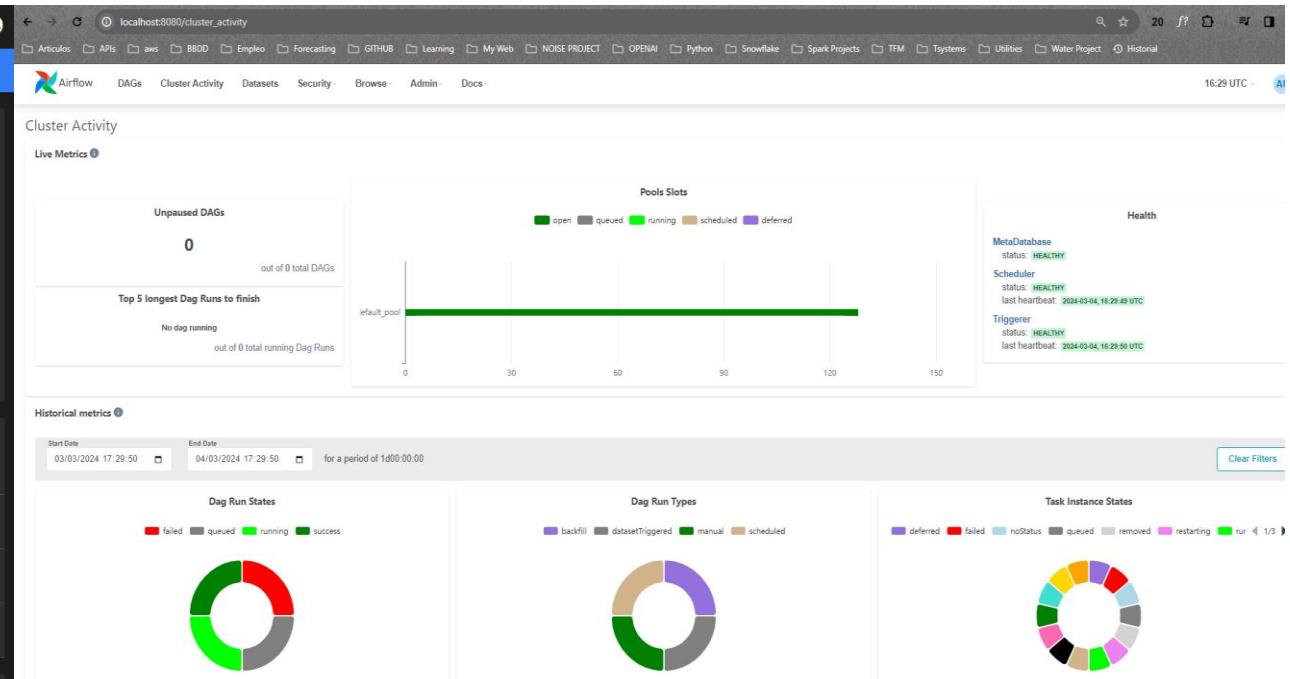
Estado de Carga de trabajo

Despliegues: Running 3

Pods: Running 7

Replica Sets: Running 3

Stateful Sets: Running 4



4. DEPLOY AIRFLOW PODS

- Create the first DAG “hello.py”.
- Push your code to github (your values.yaml should have the repo sync as showed before).
- Install airflow locally and any needed dependencies to avoid the IDE showing errors by missing packages.

```
>pip install apache-airflow
```

- The DAG **dbt_bq_final.py** needs the **KubernetesPodOperator**. Install it with this:

```
>pip install apache-airflow-providers-cncf-kubernetes
```

The screenshot shows a terminal window on the left and a GitHub browser window on the right. The terminal window displays the command to install Apache Airflow and its providers. The GitHub browser window shows the 'hello.py' DAG script in a repository named 'Kubernetes_DBT_BigQuery_DataEngineering'. The code in 'hello.py' is as follows:

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime, timedelta

default_args = {
    'owner': 'rafaelvrm',
    'start_date': datetime(2024, 03, 04),
    'catchup': False
}

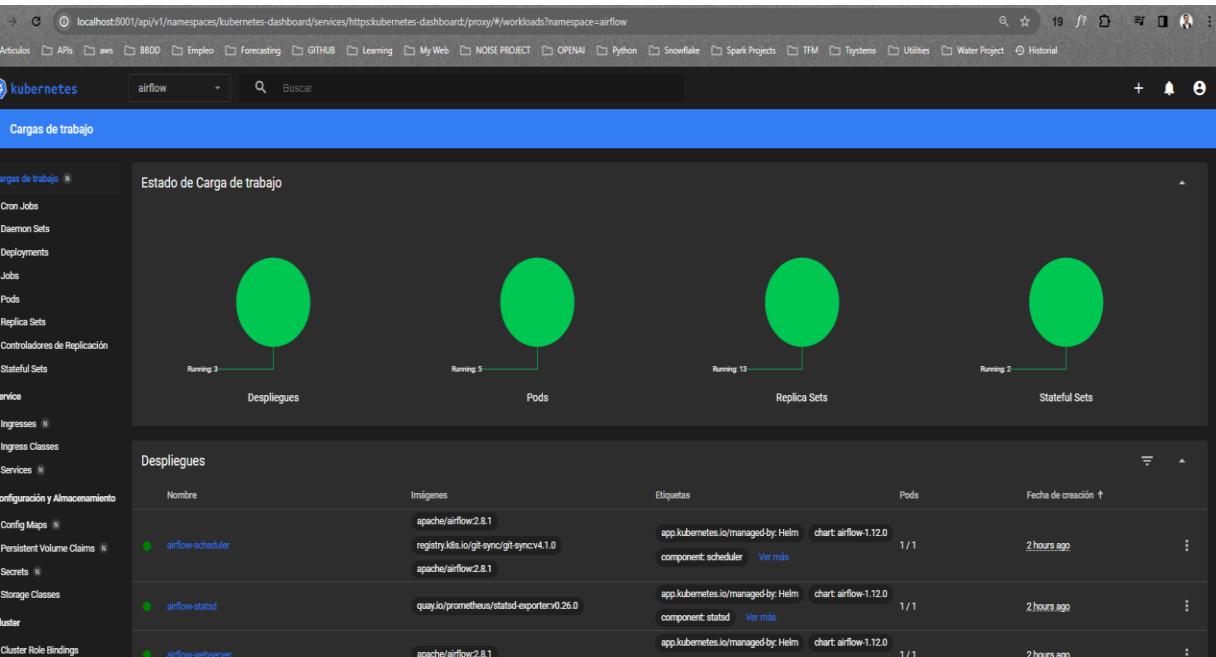
dag = DAG(
    'First_K8s',
    default_args=default_args,
    schedule=timedelta(days=1)
)

t1 = BashOperator(
    task_id='first_k8s',
    bash_command='echo "First_k8s"',
    dag=dag
)
```

4. DEPLOY AIRFLOW PODS

```
> helm upgrade --install airflow apache-airflow/airflow --namespace airflow --create-namespace -f values.yaml
```

- Check the Kubernetes Dashboard if all is up and running as expected.



The screenshot shows the Kubernetes Dashboard interface. On the left, there is a terminal window displaying the output of the Helm upgrade command. The output indicates that the "airflow" release has been upgraded successfully, with a status of "deployed" and revision 3. It also provides information about the Airflow Webserver, Default Webserver (UI), and PostgreSQL connection credentials. A warning about Fernet Key persistence is shown, along with a command to echo the Fernet Key. A note about Kubernetes workers task logs persistency is also present.

The main part of the dashboard shows the "Cargas de trabajo" (Workloads) section. It displays four large green circles representing the state of different components: "Despliegues" (Deployments) with 3 running pods, "Pods" with 5 running pods, "Replica Sets" with 13 running pods, and "Stateful Sets" with 2 running pods. Below this, the "Despliegues" table lists three entries:

Nombre	Imagenes	Etiquetas	Pods	Fecha de creación
airflow-scheduler	apache/airflow:2.8.1 registry.k8s.io/gitsync/gitsyncv4.1.0 apache/airflow:2.8.1	app.kubernetes.io/managed-by: Helm chart: airflow-1.12.0 component: scheduler	1/1	2 hours ago
airflow-sensor	quay.io/prometheus/statsd-exporter:v0.26.0	app.kubernetes.io/managed-by: Helm chart: airflow-1.12.0 component: sensor	1/1	2 hours ago
airflow-webserver	apache/airflow:2.8.1	app.kubernetes.io/managed-by: Helm chart: airflow-1.12.0	1/1	2 hours ago

4. DEPLOY AIRFLOW PODS

- Run the DAG “dbt_TEST” (First_K8s) successfully

The screenshot shows the Airflow web interface at `localhost:8080/dags/First_K8s/grid`. The top navigation bar includes links for Articulos, APIs, aws, BBDD, Empleo, Forecasting, GITHUB, Learning, My Web, NOISE PROJECT, OPENAI, Python, Snowflake, Spark Projects, and a search bar. Below the navigation is a secondary header with links for Airflow, DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs.

The main content area displays the DAG: First_K8s. A toolbar below the header offers options: Grid (selected), Graph, Calendar, Task Duration, Task Tries, Landing Times, Gantt, Details, Code, and Audit Log. The Grid view shows a single row for the DAG, with columns for Duration (00:00:21), Task (first_k8s), and Status (rvm). A date filter shows 04/03/2024 19:13:16, a run count of 25, and dropdowns for All Run Types and All Run States. Buttons for Clear Filters, deferred, failed, and queue are also present.

The right side of the screen displays the DAG Details page for First_K8s. It includes sections for DAG Runs Summary, DAG Summary, and DAG Details. Key data points include:

Category	Value
DAG Runs Summary	Total Runs Displayed: 1
DAG Runs Summary	Total running: 1
DAG Summary	Total Tasks: 2
DAG Summary	BashOperators: 2
DAG Details	Dag id: First_K8s

4. DEPLOY AIRFLOW PODS

- While running the second DAG I experienced some errors but I was not able to see the logs directly into the Airflow UI ---please do not care about the dag's names here. Just see the repo. I made name changes on the fly
- Airflow pod needs a log persistency volume either external (S3, GCP) or internal **PersistentVolumeClaim**.
- Airflow DAG (dbt_bq_final.py) runs KubernetesPodOperator pulling the docker image of dbt into a pod.
 - It needs to list of the pods inside K8s to find the good one and run dbt. To do this correctly is needed to add a **rbac.yaml** as well. Let's see all this in the next slides.

The screenshot shows the Airflow web interface at localhost:8080/home. The top navigation bar includes links for Articulos, APIs, aws, BBDD, Empleo, Forecasting, GITHUB, Learning, My Web, NOISE PROJECT, OPENAI, Python, Snowflake, Spark Projects, TFM, Systems, Utilities, Water Project, and Historial. The main header has tabs for Airflow, DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. The timestamp is 19:13 UTC. On the right, there is a user icon and a blue 'AU' button.

The main content is the 'DAGs' page. At the top, there are filters for All (2), Active (1), Paused (1), Running (0), Failed (1), and buttons for Filter DAGs by tag and Search DAGs. There is also an Auto-refresh toggle and a refresh button.

The table lists two DAGs:

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
dbt_BigQuery	rafaverama	1	1 day, 0:00:00	2024-03-04, 19:05:56	2024-03-04, 00:00:00	2		
First_K8s	rafaverama	1	1 day, 0:00:00	2024-03-04, 19:13:06	2024-03-04, 00:00:00	2		

At the bottom, there are navigation buttons for pages 1, 2, and 3, and a message indicating 'Showing 1-2 of 2 DAGs'.

5. ESTABLISH LOG PERSISTENCE FOR AIRFLOW

- To be able to see any log in Airflow UI while running or after running DAGs inside Kubernetes:
- 1. Create a **PersistentVolumeClaim.yaml** (easy way and persistence inside k8s)
- 2. Create a **rbac.yaml** (Airflow fine grained authorization to resources, verbs and ClusterRoleBinding)
- 3. helm upgrade
 > helm upgrade --install airflow apache-airflow/airflow --namespace airflow --create-namespace -f values.yaml

The screenshot shows a code editor with two tabs open: `airflow-logs-pvc.yaml` and `airflow-rbac.yaml`. The left sidebar shows a project structure for `K8s-dbt-GCP` located at `C:\Users\jrvera`, containing files like `.github`, `dags`, `dbt_bigquery_main`, `logs`, `venv`, `.gitignore`, `airflow-logs-pvc.yaml`, `airflow-rbac.yaml`, `dashboard-adminuser.yaml`, `dashboard-clusterrole.yaml`, `dashboard-secret.yaml`, `Dockerfile`, `requirements.txt`, and `values.yaml`.

`airflow-logs-pvc.yaml` content:

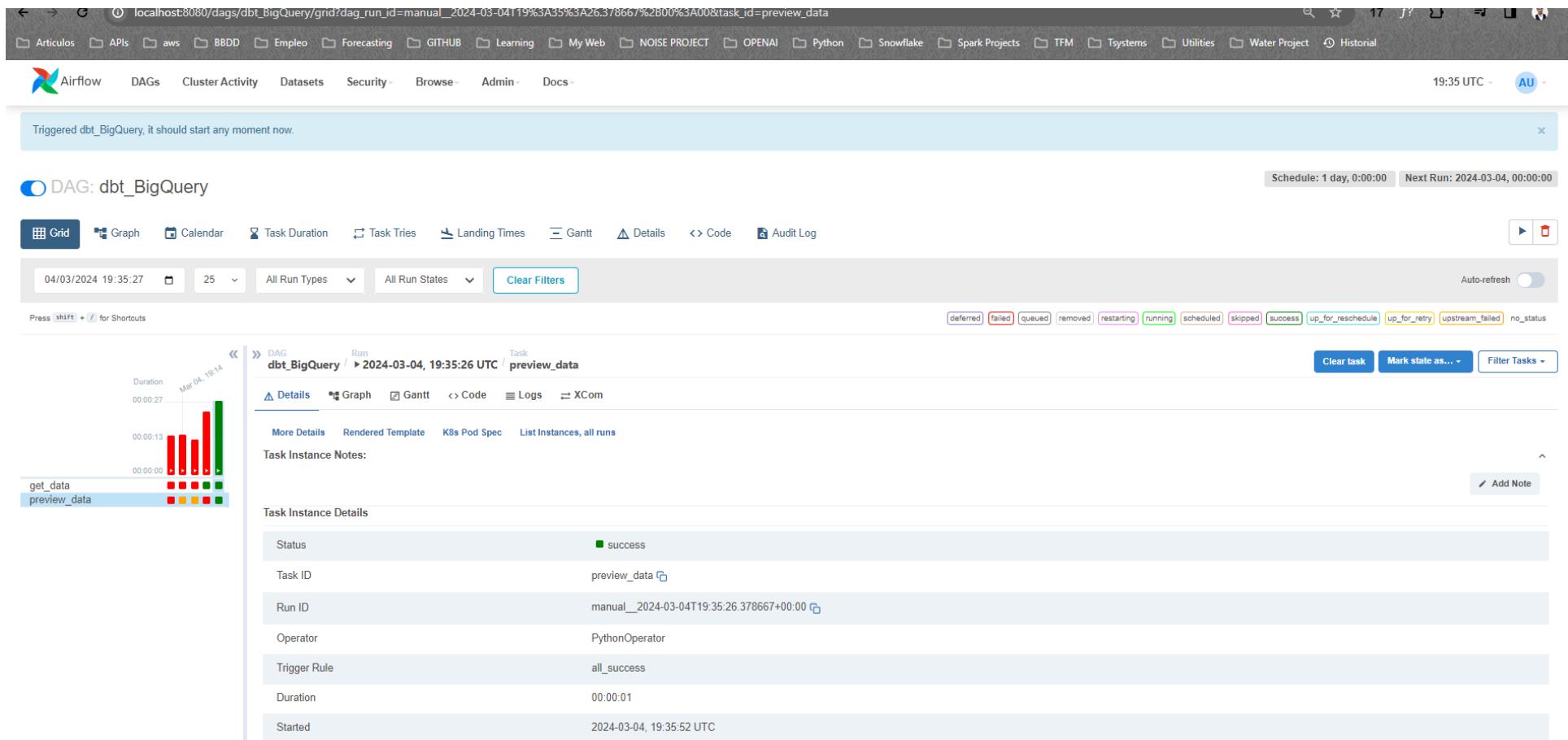
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: airflow-logs-pvc
  namespace: airflow
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: hostpath
```

`airflow-rbac.yaml` content:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: airflow-worker-role
rules:
- apiGroups: []
  resources: ["pods", "pods/log", "pods/exec"]
  verbs: ["get", "list", "watch", "create", "delete", "patch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: airflow-worker-rolebinding
subjects:
- kind: ServiceAccount
  name: airflow-worker
  namespace: airflow
roleRef:
  kind: ClusterRole
  name: airflow-worker-role
  apiGroup: rbac.authorization.k8s.io
```

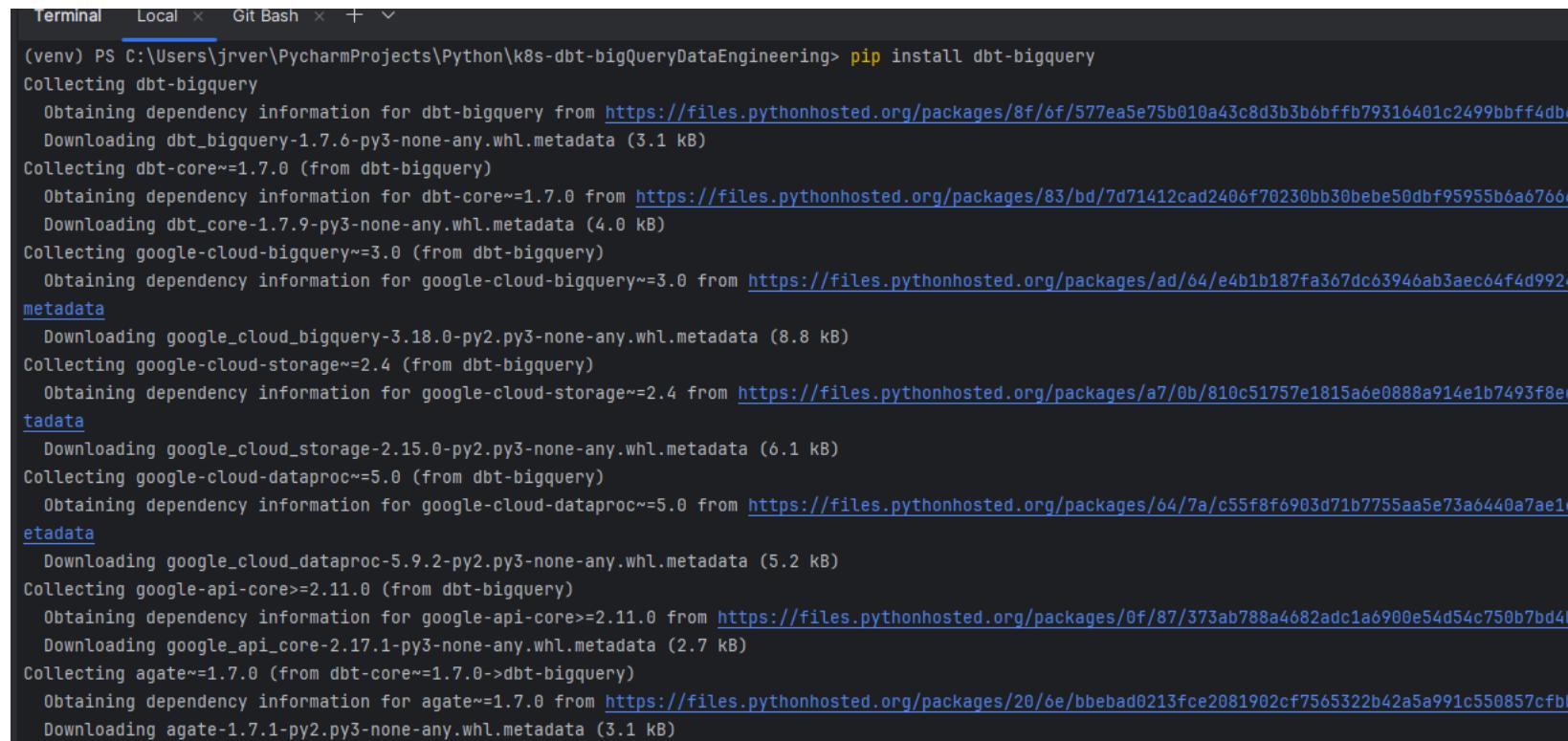
5. STABLISH LOG PERSISTENCE FOR AIRFLOW

- After setting the log persistency properly I could solve the issues faster and run the next DAG dbt_BigQuery (dbt_TEST.py)
 - Remember the DAG name in the image below was changed as I went along the rest of DAGs



6. INSTALL DBT-BIGQUERY

- Set up the dbt Project locally (later we go over dockerization) using:
 > pip install dbt-bigquery
- After instalation, in the terminal will be prompted several questions which need to answer



```
Terminal Local × Git Bash × + ▾
(venv) PS C:\Users\jrver\PycharmProjects\Python\k8s-dbt-bigQueryDataEngineering> pip install dbt-bigquery
Collecting dbt-bigquery
  Obtaining dependency information for dbt-bigquery from https://files.pythonhosted.org/packages/8f/6f/577ea5e75b010a43c8d3b3b6bffb79316401c2499bbff4db6
    Downloading dbt_bigquery-1.7.6-py3-none-any.whl.metadata (3.1 kB)
Collecting dbt-core~=1.7.0 (from dbt-bigquery)
  Obtaining dependency information for dbt-core~=1.7.0 from https://files.pythonhosted.org/packages/83/bd/7d71412cad2406f70230bb30bebe50dbf95955b6a67666
    Downloading dbt_core-1.7.9-py3-none-any.whl.metadata (4.0 kB)
Collecting google-cloud-bigquery~=3.0 (from dbt-bigquery)
  Obtaining dependency information for google-cloud-bigquery~=3.0 from https://files.pythonhosted.org/packages/ad/64/e4b1b187fa367dc63946ab3aec64f4d9924
    metadata
    Downloading google_cloud_bigquery-3.18.0-py2.py3-none-any.whl.metadata (8.8 kB)
Collecting google-cloud-storage~=2.4 (from dbt-bigquery)
  Obtaining dependency information for google-cloud-storage~=2.4 from https://files.pythonhosted.org/packages/a7/0b/810c51757e1815a6e0888a914e1b7493f8ed
    metadata
    Downloading google_cloud_storage-2.15.0-py2.py3-none-any.whl.metadata (6.1 kB)
Collecting google-cloud-dataproc~=5.0 (from dbt-bigquery)
  Obtaining dependency information for google-cloud-dataproc~=5.0 from https://files.pythonhosted.org/packages/64/7a/c55f8f6903d71b7755aa5e73a6440a7ae1e
    metadata
    Downloading google_cloud_dataproc-5.9.2-py2.py3-none-any.whl.metadata (5.2 kB)
Collecting google-api-core>=2.11.0 (from dbt-bigquery)
  Obtaining dependency information for google-api-core>=2.11.0 from https://files.pythonhosted.org/packages/0f/87/373ab788a4682adc1a6900e54d54c750b7bd4b
    Downloading google_api_core-2.17.1-py3-none-any.whl.metadata (2.7 kB)
Collecting agate~=1.7.0 (from dbt-core~=1.7.0->dbt-bigquery)
  Obtaining dependency information for agate~=1.7.0 from https://files.pythonhosted.org/packages/20/6e/bbebad0213fce2081902cf7565322b42a5a991c550857cfbb
    Downloading agate-1.7.1-py2.py3-none-any.whl.metadata (3.1 kB)
```

6. INSTALL DBT-BIGQUERY

- Prompted questions:
 - Give your **dbt Project a name**:
My advice, choose an easy one and don't change your mind after. I did so and experienced many issues.
As you can see the dbt Project folder in the images is different as in the repo. I take almost the same name for several folders and it caused incompatibilities thus I changed the Project name after.
 - Choose **database Bigquery** [1]
 - Choose **authenticacion by service_account** [2]
 - Give the path of your Google Cloud Platform Auth **.JSON** (give the path. you will get this keys in a few steps)
 - GCP Project ID** : gives one (you will create it after in GCP **with the same name**)
 - Dataset (dbt)**: give the name of your preference
 - Threads**: 1
 - Location**: 1 [us]

```
(venv) PS C:\Users\jrver\PycharmProjects\Python\k8s-dbt-bigQueryDataEngineering> dbt init dbt-BigQuery
14:41:07 Running with dbt=1.7.9
dbt-BigQuery is not a valid project name.
Enter a name for your project (letters, digits, underscore): dbt_BigQuery
14:42:18
Your new dbt project "dbt_BigQuery" was created!

For more information on how to configure the profiles.yml file,
please consult the dbt documentation here:

https://docs.getdbt.com/docs/configure-your-profile

One more thing:

Need help? Don't hesitate to reach out to us via GitHub issues or on Slack:
https://community.getdbt.com/

Happy modeling!

14:42:18 Setting up your profile.
Which database would you like to use?
[1] bigquery

(Don't see the one you want? https://docs.getdbt.com/docs/available-adapters)

Enter a number:
[1] oauth
[2] service_account
Desired authentication method option (enter a number): 2
keyfile (/path/to/bigquery/keyfile.json): /Users/jrver/.dbt/dbt-BigQuery.json
project (GCP project id): dbt-BigQuery-rvm-01
dataset (the name of your dbt dataset): dbt-BigQuery
threads (1 or more): 1
job_execution_timeout_seconds [300]:
[1] US
[2] EU
Desired location option (enter a number): 1
14:52:15 Profile dbt_BigQuery written to C:/Users/jrver/.dbt/profiles.yml using target's profile_template
```

```
(venv) PS C:\Users\jrver\PycharmProjects\Python\k8s-dbt-bigQueryDataEngineering>
```

7. SETUP BIGQUERY-GCP

- Go to GCP-BigQuery : <https://cloud.google.com/bigquery?hl=en>
- Sign in and **Try BigQuery free**

The screenshot shows the Google Cloud BigQuery product page. The top navigation bar includes links for Overview, Solutions, Products (which is the active tab), Pricing, Resources, and Contact Us. The right side of the header features a search icon, language selection (English), sign-in options, and a prominent blue "Start free" button.

The main content area has a sidebar on the left with sections for Features (How It Works, Common Uses, Data warehouse migration, Transfer data into BigQuery, Real-time analytics, Predictive analytics, Log analytics, Marketing analytics, Data clean rooms), Pricing, Partners & Integration, and FAQ. Below this is a "Release notes" section with links to "Documentation" and a "View all releases" button.

The central content area features a green banner stating "Google is positioned furthest in vision among all leaders in the 2023 Gartner® Magic Quadrant™ for Cloud DBMS. Get the report." Below this is a "BigQuery" section with the heading "Cloud data warehouse to power your data-driven innovation". It describes BigQuery as a serverless and cost-effective enterprise data warehouse that works across clouds and scales with your data, using built-in ML/AI and BI for insights at scale. It includes a "Try BigQuery free" button and a "Contact sales" button. Text below states that new customers get \$300 in free credits and all customers get 10 GiB storage and up to 1 TiB queries free per month.

The right side of the page contains a "Product highlights" section with three items: "Access and analyze data across clouds", "Built-in machine learning for all data types using simple SQL", and "Real-time analytics with streaming and built-in BI". Below this is a "Learn BigQuery in a minute" section featuring a thumbnail of a video and a "1-minute video" link.

At the bottom, there's a footer with links to "BigQuery Studio", "BigQuery API", "BigQuery ML", "BigQuery Data Catalog", "BigQuery Storage API", and "BigQuery Job Transfer".

7. SETUP BIGQUERY-GCP

- By default BigQuery takes “Robot” Project. Let’s create your own Project.
- Click on the dropdown and a pop-up window opens “**SELECT YOUR PROJECT**”
 - “**New Project**” opens a little pop-up -> just give the Project a name
 - Create

This screenshot shows the BigQuery Studio interface. At the top, there's a navigation bar with links like Artículos, APIs, AWS, BBDD, Empleo, Forecasting, GITHUB, Learning, My Web, NOISE PROJECT, OPENAI, Python, Snowflake, Spark Projects, and TFM. Below the bar, a search bar says "Buscar (/) recursos, documentos, productos y más". A dropdown menu is open, showing "Robot" selected. The main area has a banner saying "Te damos la bienvenida a BigQuery Studio" and "Comenzar". Another banner below it says "Te damos la bienvenida a BigQuery en la consola de Cloud" and "CREAR UNA CONSULTA EN SQL".

The first part of this image shows a "Selección de proyecto" (Select Project) dialog. It has a search bar, a "PROYECTO NUEVO" button with a hand cursor, and tabs for RECENTES, DESTACADOS, and TODOS. It lists several projects: dbt-BigQuery-RVM, My First Project, Robot, and another My First Project entry. The second part shows a "Proyecto nuevo" (New Project) dialog. It displays a warning about quota limits and a "MANAGE QUOTAS" link. The "Nombre del proyecto" field is filled with "My Project 22314". The "Ubicación" field shows "Sin organización" with an "EXPLORAR" button. At the bottom are "CREAR" and "CANCELAR" buttons.

7. SETUP BIGQUERY-GCP

- Now you can see the BigQuery console of your new Project
- The name showed in the explore section isn't case sensitive, so keep the Project name simple.

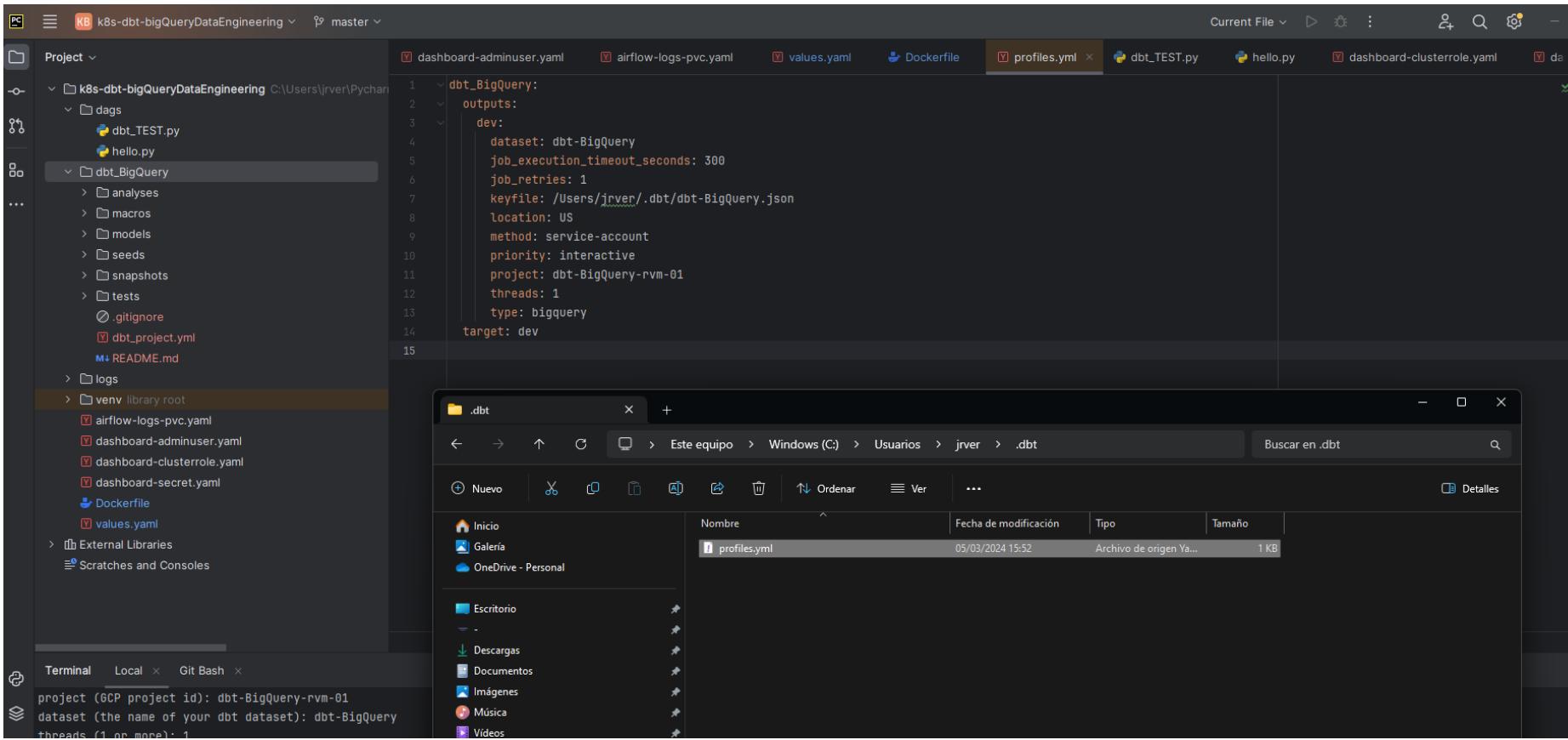
The screenshot shows the Google Cloud BigQuery Studio interface. At the top, there's a navigation bar with the 'Google Cloud' logo and a dropdown menu showing 'dbt-BigQuery-RVM'. A yellow box highlights this dropdown. To its right is a search bar with the word 'big' and a 'Buscar' button. On the far right of the top bar are several icons: a document, a refresh, a green circle with the number '1', a question mark, three dots, and a user profile.

The main area is titled 'BigQuery' and has a sub-header 'Explorador'. It features a toolbar with 'AGREGAR', a search bar ('Sin título'), and various actions like 'EJECUTAR', 'GUARDAR', 'DESCARGAR', 'COMPARТИR', 'PROGRAMACIÓN', and 'MÁS'. Below the toolbar is a message: 'Your BigQuery projects will have new capabilities after February 14, 2024. Services and roles will be enabled automatically to help with these changes.' with a 'Más información' link.

The left sidebar contains a navigation tree with sections like 'Análisis', 'BigQuery Studio' (which is selected and highlighted in blue), 'Transferencias de datos', 'Consultas programadas', 'Analytics Hub', 'Dataform', 'Centro de socios', 'Migración', 'Evaluación', 'Traducción de SQL', 'Administración', 'Supervisión', and 'Administración de capacidad'. The 'BigQuery Studio' section is expanded, showing 'dbt-bigquery-rvm' which is also highlighted with a yellow box. Under 'dbt-bigquery-rvm', there are three main categories: 'Consultas', 'Notebooks', and 'Conexiones externas', each with their respective sub-items and icons.

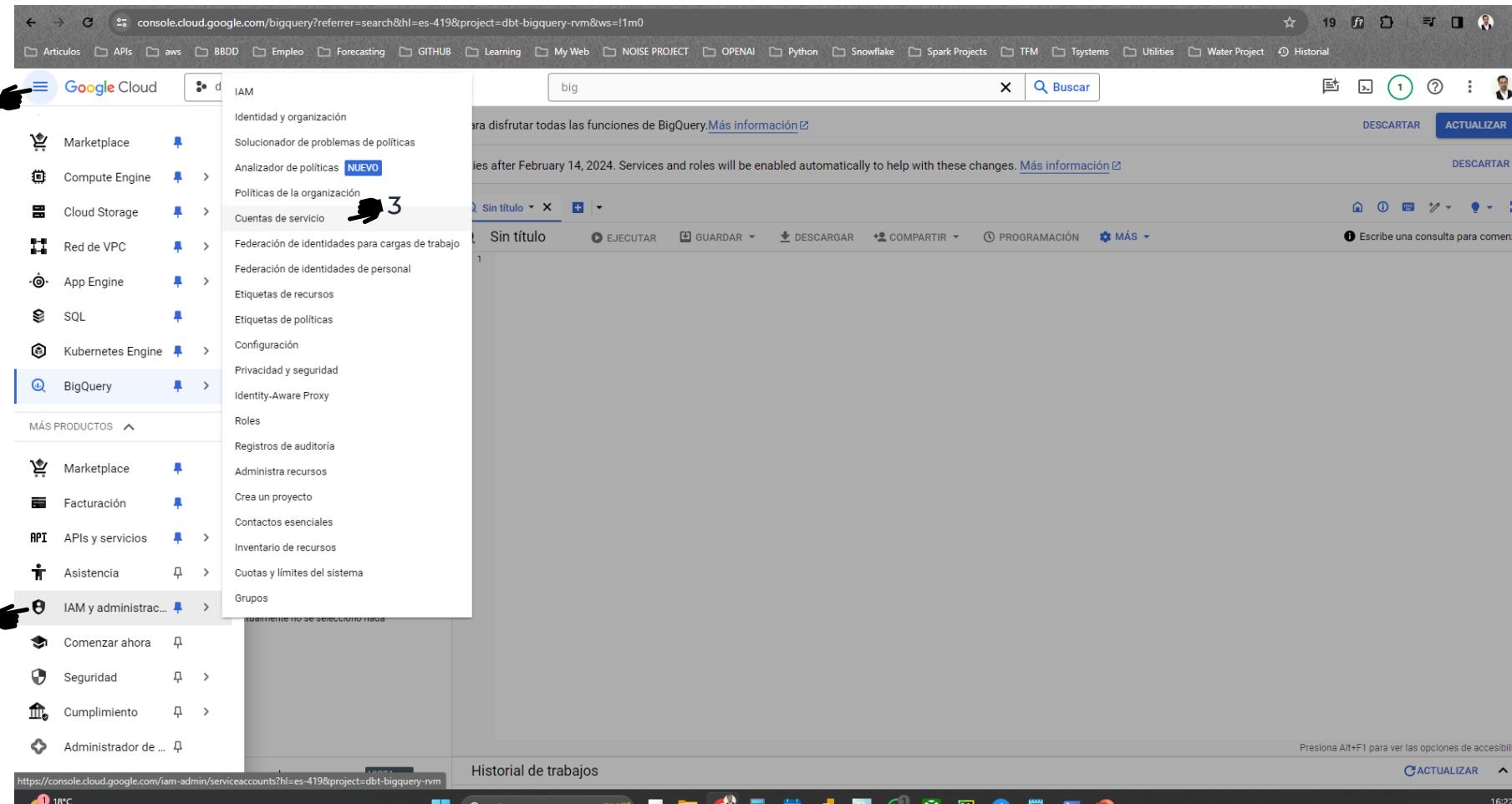
7. SETUP BIGQUERY-GCP

- We haven't yet the keyfile which is needed to be passed in **profiles.yml**
- This profiles.yml is in C:/users/your_user/.dbt
 - You can open it in pycharm just drag and dropping it.
 - I highly recommend to copy paste it inside your dbt Project directory within your env.
- Now go to Google Cloud to get the keyfile with your GCP credentials in json format.



7. SETUP BIGQUERY-GCP

- **Getting keyfile**
- 1. Click top left on the stacked lines
- 2. Go to IAM and Management
- 3. Service account



7. SETUP BIGQUERY-GCP

- **Getting keyfile**
- 4. Create a service account

The screenshot shows the Google Cloud IAM & administration service accounts interface. The left sidebar has a blue header for 'Cuentas de servicio'. The main area displays the title 'Cuentas de servicio del proyecto "dbt-BigQuery-RVM"' and a note about service accounts. A search bar at the top right contains the word 'big'. Below the search bar are buttons for '+ CREAR CUENTA DE SERVICIO', 'BORRAR', 'ADMINISTRAR ACCESO', and 'ACTUALIZAR'. A table below lists service accounts, with a note 'No hay filas para mostrar'.

Correo electrónico	Estado	Nombre ↑	Descripción	ID de clave	Fecha de creación de la clave	ID de cliente de OAuth 2	Acciones
No hay filas para mostrar							

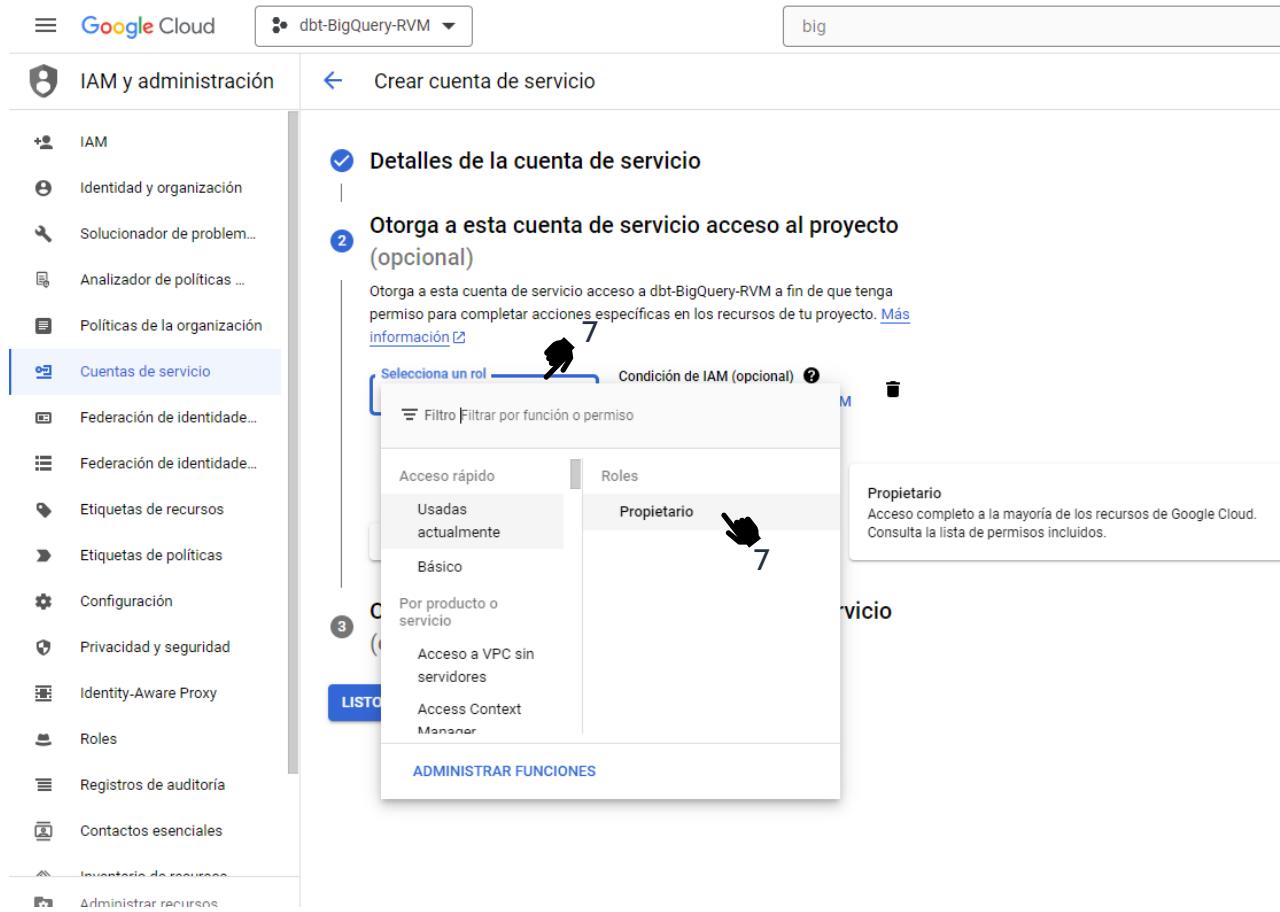
7. SETUP BIGQUERY-GCP

- **Getting keyfile**
- 5. Give your service account a name and ID
- 6. Create and continue

The screenshot shows the Google Cloud IAM & Administration interface for creating a new service account. The left sidebar lists various IAM-related options like IAM, Identity and Organization, and Service Accounts. The main panel is titled 'Crear cuenta de servicio' (Create service account).
Step 1: Detalles de la cuenta de servicio
The first section is labeled 'Detalles de la cuenta de servicio'. It contains fields for the service account's name ('Nombre de la cuenta de servicio' with value 'Rafael Vera') and ID ('ID de la cuenta de servicio' with value 'rafael-vera'). A note below says 'Mostrar nombre de esta cuenta de servicio' (Show name of this service account) and 'Dirección de correo electrónico: rafael-vera@dbt-bigquery-rvm.iam.gserviceaccount.com'. A 'CREAR Y CONTINUAR' (Create and Continue) button is at the bottom of this section.
Step 2: Otorga a esta cuenta de servicio acceso al proyecto (opcional)
This section is labeled '2' and 'Otorga a esta cuenta de servicio acceso al proyecto (opcional)'. It includes a checkbox for granting project access.
Step 3: Otorga a usuarios acceso a esta cuenta de servicio (opcional)
This section is labeled '3' and 'Otorga a usuarios acceso a esta cuenta de servicio (opcional)'. It includes a checkbox for granting user access.
Bottom Buttons
At the bottom, there are two buttons: 'LISTO' (Ready) and 'CANCELAR' (Cancel). A large hand cursor icon is positioned over the 'CREAR Y CONTINUAR' button.

7. SETUP BIGQUERY-GCP

- **Getting keyfile**
- 7. Assign permissions to the Project -> **Owner**
- Continue



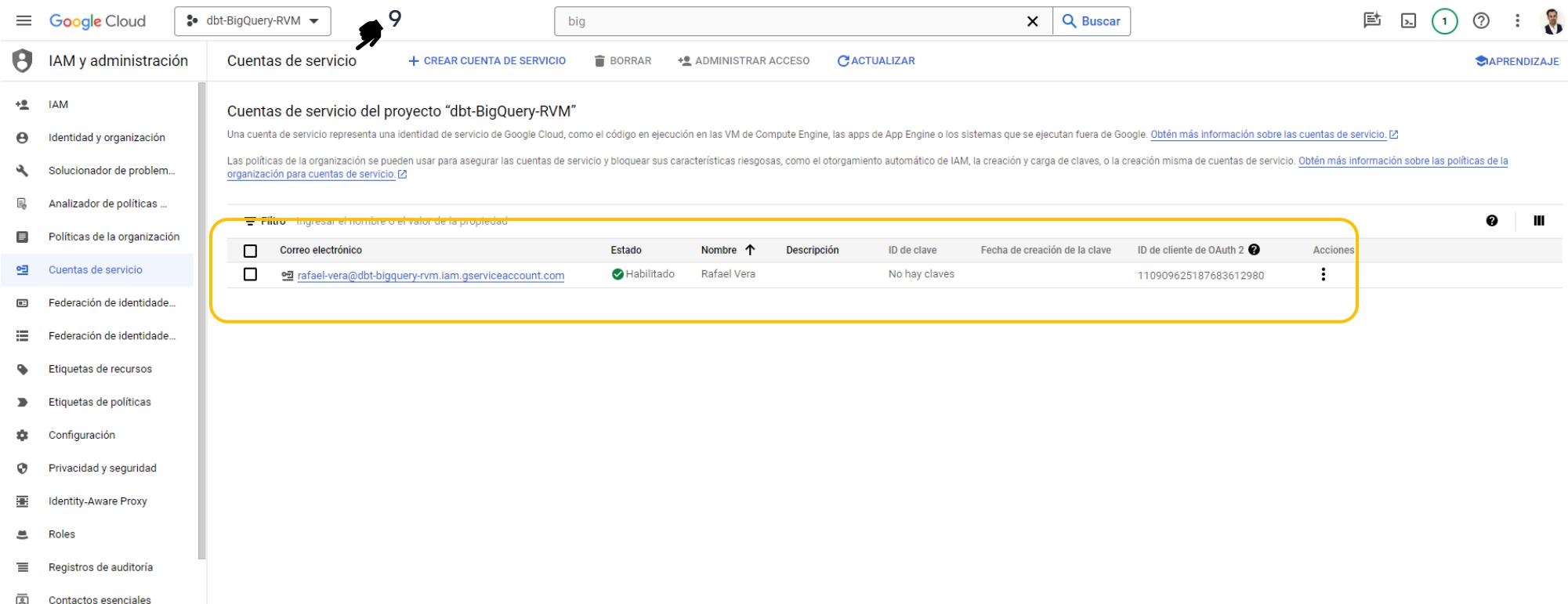
7. SETUP BIGQUERY-GCP

- **Getting keyfile**
- 8. Ready
- Left Access user assignment empty

The screenshot shows the Google Cloud IAM service account creation interface. The left sidebar lists various IAM-related options: IAM, Identidad y organización, Solucionador de problemas..., Analizador de políticas ..., Políticas de la organización, Cuentas de servicio (which is selected), Federación de identidad..., Federación de identidad..., Etiquetas de recursos, Etiquetas de políticas, Configuración, Privacidad y seguridad, Identity-Aware Proxy, Roles, and Registros de auditoría. The main panel is titled "Crear cuenta de servicio" and contains three sections: "Detalles de la cuenta de servicio" (checked), "Otorga a esta cuenta de servicio acceso al proyecto (opcional)" (checked), and "Otorga a usuarios acceso a esta cuenta de servicio (opcional)". The third section includes a note: "Grant access to users or groups that need to perform actions as this service account." Below these sections are two input fields: "Función de los usuarios de cuentas de servicio" and "Función de los administradores de cuentas de servicio". At the bottom are "LISTO" and "CANCELAR" buttons, with a hand cursor pointing at the "LISTO" button. A small number "8" is located at the bottom center of the main panel.

7. SETUP BIGQUERY-GCP

- **Getting keyfile**
- Service Account for the project successfully created
- 9. Go to service Accounts

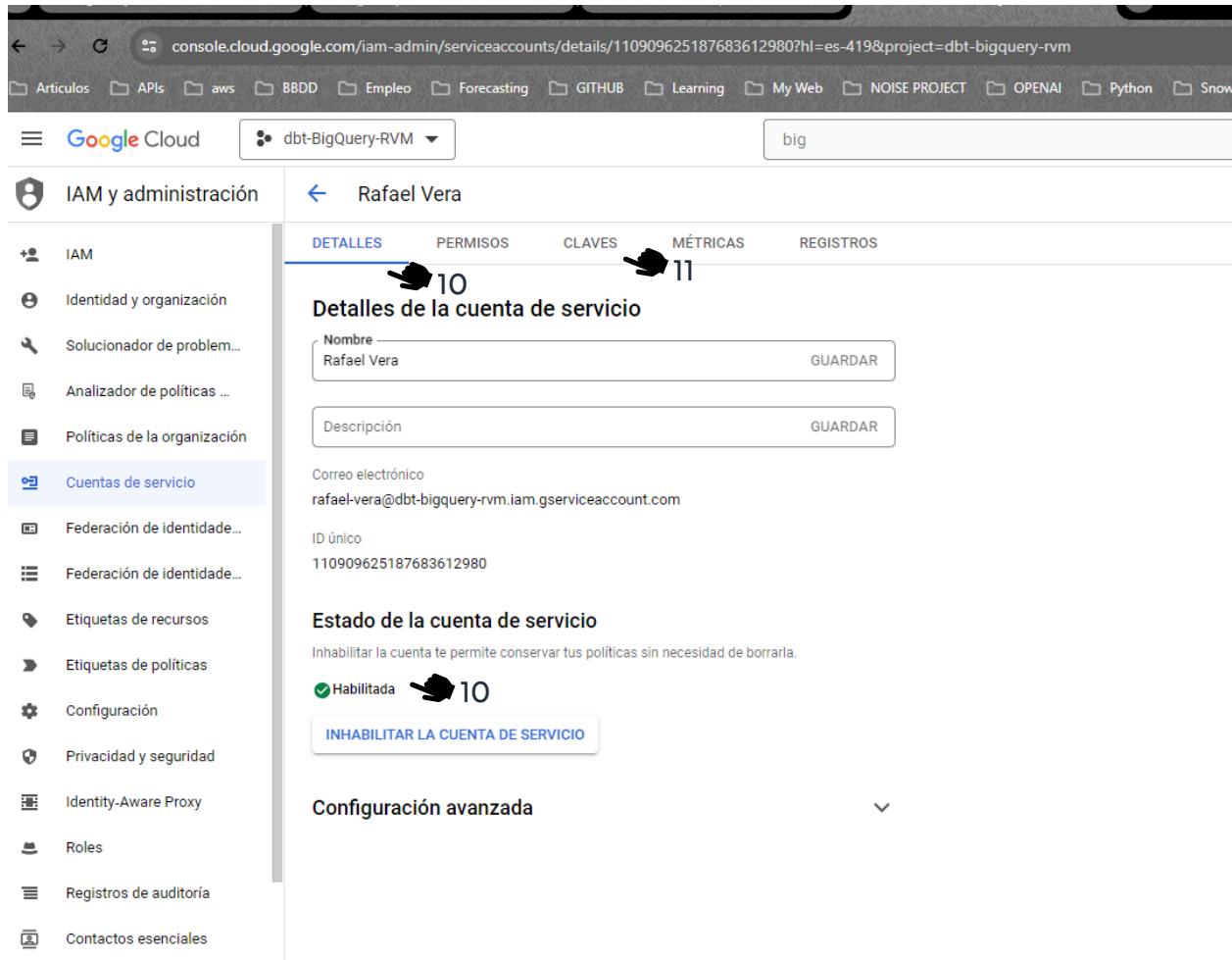


The screenshot shows the Google Cloud IAM & administration service accounts page. The left sidebar is titled "IAM y administración" and includes options like IAM, Identidad y organización, Solucionador de problemas, Analizador de políticas, Políticas de la organización, Cuentas de servicio, Federación de identidades, Etiquetas de recursos, Etiquetas de políticas, Configuración, Privacidad y seguridad, Identity-Aware Proxy, Roles, Registros de auditoría, and Contactos esenciales. The main content area is titled "Cuentas de servicio del proyecto ‘dbt-BigQuery-RVM’". It contains a table with one row, highlighted by a yellow box. The table columns are: Correo electrónico (rafael-vera@dbt-bigquery-rvm.iam.gserviceaccount.com), Estado (Habilitado), Nombre (Rafael Vera), Descripción (No hay claves), Fecha de creación de la clave (110909625187683612980), and Acciones (three dots). A search bar at the top right shows the word "big".

Correo electrónico	Estado	Nombre	Descripción	Fecha de creación de la clave	Acciones
rafael-vera@dbt-bigquery-rvm.iam.gserviceaccount.com	Habilitado	Rafael Vera	No hay claves	110909625187683612980	⋮

7. SETUP BIGQUERY-GCP

- **Getting keyfile**
- 10. See Service Accounts details – check the Service Account Status : **Enabled**
- 11. Keys



7. SETUP BIGQUERY-GCP

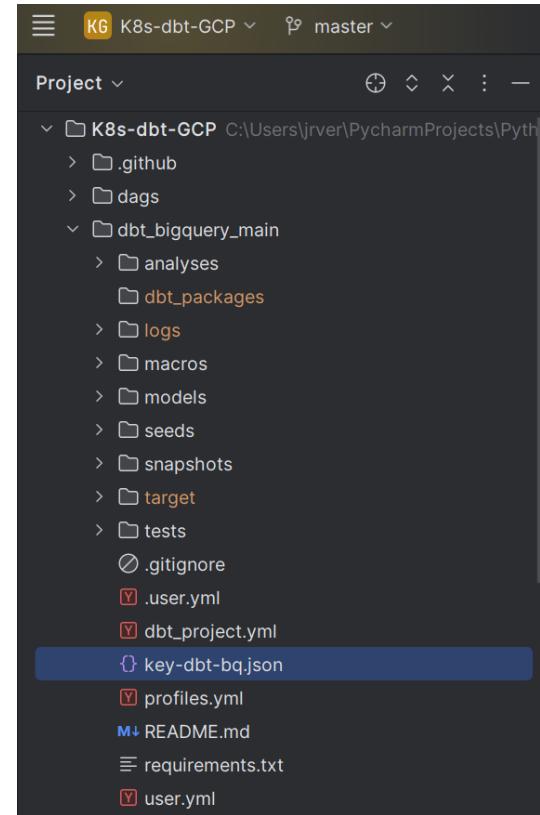
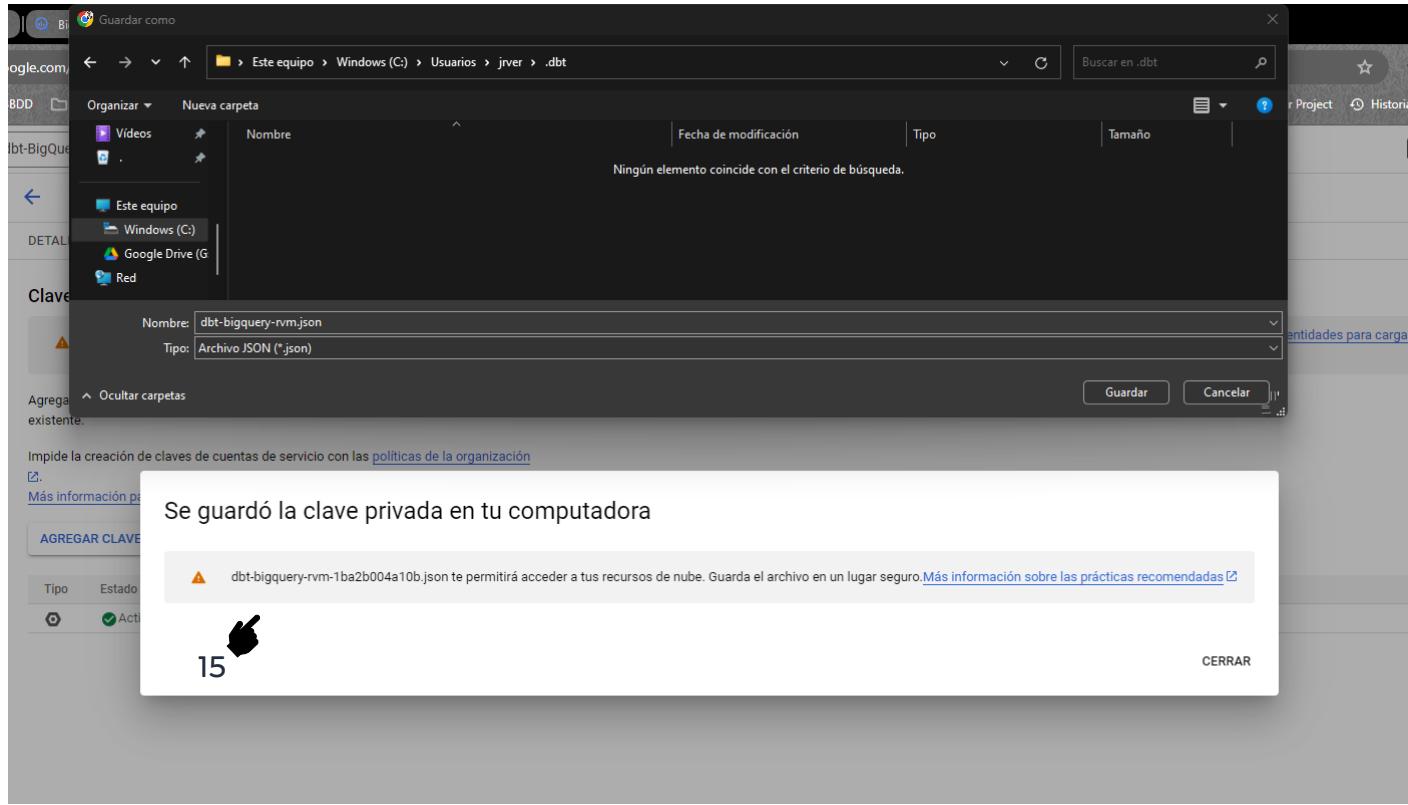
- **Getting keyfile**
- 12. Create a new Key -> pop up shows up
- 13. Add new key -> Type JSON
- 14. Create key -> pops up a window to select the downloading path into your computer

The screenshot shows the Google Cloud IAM & administration interface. On the left sidebar, under 'Cuentas de servicio', the 'AGREGAR CLAVE' button is highlighted with a black box and labeled '12'. Below it are two options: 'Crear clave nueva' and 'Subir clave existente'.

The screenshot shows a modal dialog titled 'Crear clave privada para "Rafael Vera"'. It contains a warning message about service account keys being a security risk if compromised. It also includes instructions for creating a new key or uploading an existing certificate. A dropdown menu 'AGREGAR CLAVE' is visible at the top left of the dialog. The 'Tipo de clave' section shows two options: 'JSON' (selected) and 'P12'. The 'JSON' option is labeled 'Recomendado'. At the bottom right of the dialog, there are 'CANCELAR' and 'CREAR' buttons, with the 'CREAR' button highlighted with a black box and labeled '14'.

7. SETUP BIGQUERY-GCP

- **Getting keyfile**
- 15. The private keyfile is successfully downloaded
 - At the first time I pointed to C:/users/.dbt but I recommend you save it inside your dbt Project folder within your environment together with profiles.yml



7. SETUP BIGQUERY-GCP

- **Keyfile downloaded**
- **Confirm the correct path and BigQuery Project name in profiles.yml**
 - This keyfile path must point exactly where you GCP-privateKey is
 - dbt will use this profiles.yml to get and set connection parameter with BigQuery
 - You can explore the json to see what is inside
- **Confirm the Project parameter match exactly your Project name in BigQuery**
 - As you can see I mistaked at the first try and as I run in pycharm terminal “dbt debug” I received and error.

The image consists of two side-by-side screenshots. The left screenshot shows the Google Cloud BigQuery interface with a project named 'dbt-BigQuery-rvm'. A yellow arrow points from the 'keyfile' entry in the 'outputs' section of the configuration to the 'dbt-biggquery-rvm' project in the list. Another yellow arrow points from the 'project' entry in the configuration to the same project in the list. The right screenshot shows a code editor with a JSON file named 'key-dbt-bq.json'. The JSON content is as follows:

```
{  
  "type": "service_account",  
  "project_id": "dbt-bigquery",  
  "private_key_id": "1ba2b004a...",  
  "private_key": "-----BEGIN P...",  
  "client_email": "rafael-vera...",  
  "client_id": "1109096251876...",  
  "auth_uri": "https://account...",  
  "token_uri": "https://oauth...",  
  "auth_provider_x509_cert_url": "...",  
  "client_x509_cert_url": "...",  
  "universe_domain": "googleads..."}
```

7. SETUP BIGQUERY-GCP

- **Keyfile downloaded**
- **Profiles.yml doble checked**
- It's time to know if dbt is able to connect to bigquery:
 - Run the following command on IDE terminal, pointing to your dbt Project directory inside the enviroment

> dbt debug

- We have not yet created any dbt model, there are only the bydefault examples into models folder
- After debugging you can see all connections succeded
 - That means that the configuration is correctly setup

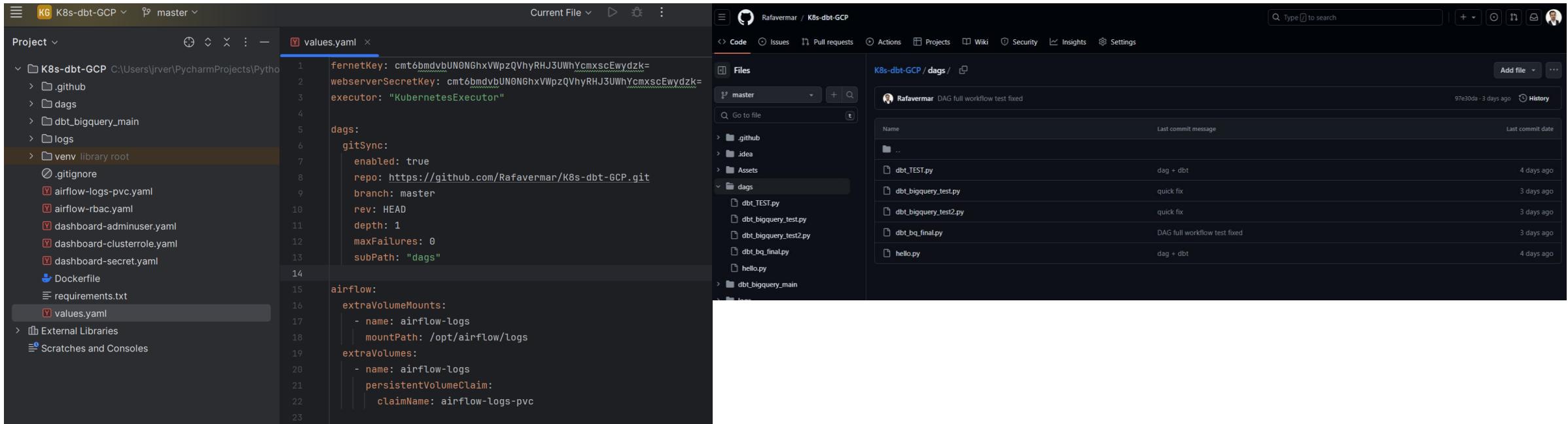
The screenshot shows the PyCharm IDE interface with the 'k8s-db-t BigQueryDataEngineering' project open. In the left sidebar, the 'dbt' directory contains 'dbt_TEST.py', 'hello.py', and 'ad_t BigQuery'. The 'ad_t BigQuery' directory contains 'models', 'analyses', and 'logs'. In the top right, several configuration files are listed: 'dashboard-adminuser.yaml', 'airflow-logs-pvc.yaml', 'values.yaml', and 'Dockerfile'. The terminal window at the bottom displays the output of the 'dbt debug' command. The log starts with 'profiles.yml file [W found and valid]', followed by 'Required dependencies:', and then a detailed connection configuration for 'Connection'. The connection details include method: service-account, database: dbt-bigquery-rvm, execution_project: dbt-bigquery-rvm, scheme: dbt-BigQuery, location: US, priority: interactive, and various timeout and retry parameters. The log concludes with 'Registered adapter: bigquery@1.7.0', 'Connection test: [Ok connection ok]', and 'All checks passed!'. The terminal also shows the path '(venv) PS C:\Users\jrvr\PycharmProjects\Python\k8s-db-t BigQueryDataEngineering\dbt_BigQuery>'.

```
15:33:30 profiles.yml file [W found and valid]
15:33:30 dbt-project.yml file [W found and valid]
15:33:30 Required dependencies:
15:33:30 - git [ok found]

15:33:30 Connection:
15:33:30   method: service-account
15:33:30   database: dbt-bigquery-rvm
15:33:30   execution_project: dbt-bigquery-rvm
15:33:30   scheme: dbt-BigQuery
15:33:30   location: US
15:33:30   priority: interactive
15:33:30   maximum_bytes_billed: None
15:33:30   impersonate_service_account: None
15:33:30   job_retry_deadline_seconds: None
15:33:30   job_retries: 1
15:33:30   job_creation_timeout_seconds: None
15:33:30   max_parallelism: 100
15:33:30   timeout_seconds: 300
15:33:30   keyfile: /Users/jrvr/.dbt/dbt-BigQuery.json
15:33:30   timeout_seconds: 300
15:33:30   client_id: None
15:33:30   token_url: None
15:33:30   dataproc_region: None
15:33:30   dataproc_cluster_name: None
15:33:30   gcs_bucket: None
15:33:30   dataproc_batch: None
15:33:30 Registered adapter: bigquery@1.7.0
15:33:32 Connection test: [Ok connection ok]
15:33:32 All checks passed!
(venv) PS C:\Users\jrvr\PycharmProjects\Python\k8s-db-t BigQueryDataEngineering\dbt_BigQuery>
```

8. SYNCHRONIZE DAGs WITH GITHUB

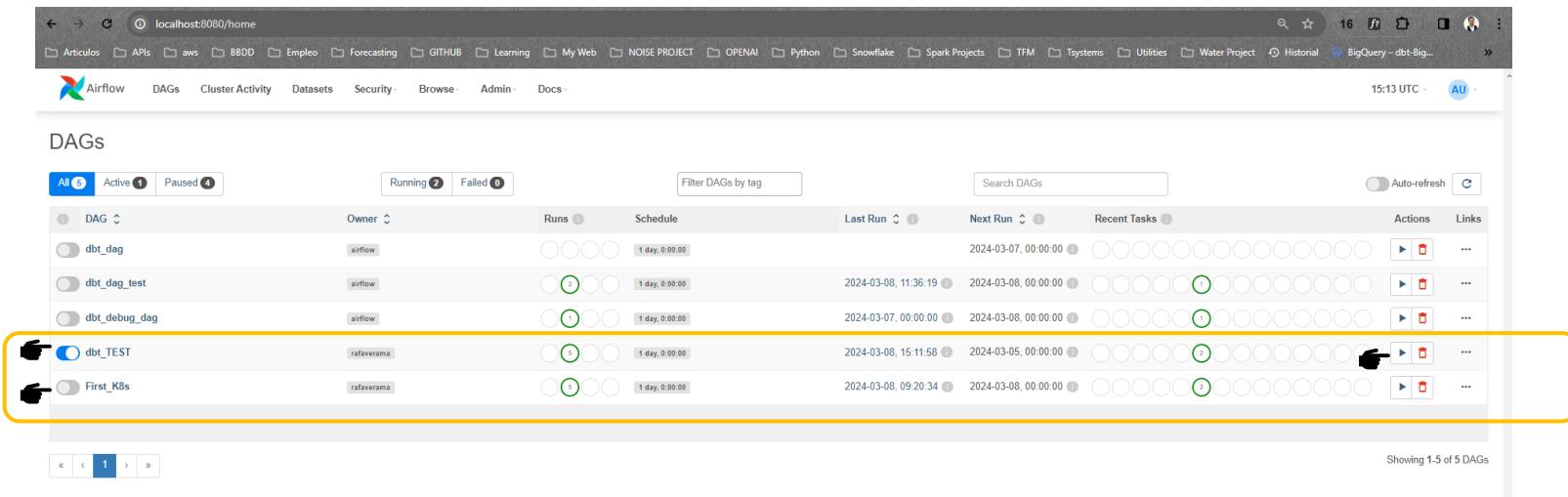
- Before continue with dbt better try some Airflow DAGs tests
- And make sure your Airflow Pod un Kubernetes is in synchronization with your Github repository
- **1. Ensure the values.yaml contains the gitSync configuration and points to your repo**
- **2. Commit and push the changes and DAGs and see the sync DAGs in Airflow UI**
- **3. Launch DAG : hello.py and dbt_Test**



8. SYNCHRONIZE DAGS WITH GITHUB

- **3. Launch DAG : hello.py and dbt_Test**

- Let's see both DAGs in Airflow UI (K8s pod on localhost and still forwarding ports)
- Note: the image below shows all the successfully run DAG. You will see only DAG names and no runs
- hello.py and dbt_Test are DAGs which don't involve dbt into the orchestrated process (just for try airflow)
 - **hello.py** -> performs a simple BashOperator tasks to print something in bash
 - **dbt_Test** -> performs two simple tasks: get a .csv from my public repo, create a DF selecting few columns and preview this data
- **To do this, in Airflow UI: Activate the DAG (change to blue color) and click run on the right**



8. SYNCHRONIZE DAGS WITH GITHUB

- **4. Create a DAG as the image, commit and push it to Github**
 - Note: Dbt_BigQuery_SAmplesModels.py DOESNOT EXIST in the repo anymore.
 - It was an intermediate DAG for testing purposes
 - This DAG orchestrate the first dbt workflow just running “dbt run” through BashOperator.
 - The dbt models run are those bydefault with the dbt installation and Airflow K8s pod run the command in our environment.

The screenshot shows a code editor interface with a dark theme. On the left is a file tree for a project named "k8s-dbt-bigQueryDataEngineering". The "dags" directory contains three files: "dbt_BIGQuery_SampleModels.py", "dbt_TEST.py", and "hello.py". The "dbt_BIGQuery" directory contains sub-directories "analyses", "logs", "macros", "models", "seeds", "snapshots" (which contains ".gitkeep"), "tests", ".gitignore", "dbt_project.yml", and "README.md". The "logs" and "venv" directories are also shown. The main editor area displays the content of "dbt_BIGQuery_SampleModels.py". The code defines a DAG named "dbt_dag" with default arguments, a schedule interval of one day, and a catchup=False setting. It uses a BashOperator to run "dbt run" with specific profile and project settings. The terminal at the bottom shows the output of running the DAG, including adapter registration and connection tests.

```
from airflow.operators.bash import BashOperator
from datetime import datetime, timedelta

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(year=2024, month=3, day=4),
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(
    dag_id='dbt_dag',
    default_args=default_args,
    schedule_interval=timedelta(days=1),
    catchup=False,
)

dbt_run = BashOperator(
    task_id='dbt_run',
    bash_command='dbt run --profiles-dir /opt/airflow/dbt --project-dir /opt/airflow/dbt',
    dag=dag,
)
```

Terminal Local × Git Bash ×

15:33:30 Registered adapter: bigquery=1.7.6
15:33:32 Connection test: [OK connection ok]

15:33:32 All checks passed!

8. SYNCHRONIZE DAGS WITH GITHUB

- **5. Watch how the triggered Airflow-Pod DAG run our dbt Project and sent the model to BigQuery**
 - Thanks to values.yaml (Airflow-k8s-pod) and profiles.yml (dbt Project) mainly.
 - Note that:
 - my_first_dbt_model is materialized as table
 - my_second_dbt_model is materialized as view → later go come back to this but using CI/CD

The screenshot shows the Google Cloud BigQuery interface. The top navigation bar includes the Google Cloud logo, a dropdown for 'dbt-BigQuery-RVM', a search bar, and a 'Buscar' button. On the left, there's a sidebar with various icons. The main content area has a title 'ZONA DE PRUEBAS' with a sub-instruction 'Configura la facturación para disfrutar todas las funciones de BigQuery.' and a link 'Más información'. Below this, a message states: 'Tus proyectos de BigQuery tendrán nuevas capacidades a partir del 14 de febrero de 2024. Los servicios y los roles se habilitarán automáticamente para ayudar con estos cambios.' with another 'Más información' link. The central part of the screen displays the schema for a model named 'my_first_dbt_model'. The schema table has columns: Nombre del campo, Tipo, Modo, Clave, Intercalación, Valor predeterminado, Etiquetas de políticas, and Descripción. A row for the 'id' field is shown, with 'id' as the name, 'INTEGER' as the type, and 'NULLABLE' as the mode. At the bottom of the schema view, there are buttons for 'EDITAR ESQUEMA' and 'VER POLÍTICAS DE ACCESO DE FILA'. A yellow box highlights the 'my_first_dbt_model' entry in the project tree on the left.

8. SYNCHRONIZE DAGS WITH GITHUB

- **Troubleshooting**
- Probably somewhen you get the following error, running `dbt run` either into your terminal or through DAGs

```
    return func(*args, **kwargs)
File "/usr/local/lib/python3.9/site-packages/dbt/cli/requirements.py", line 271, in wrapper
    ctx.obj["manifest"] = parse_manifest(
File "/usr/local/lib/python3.9/site-packages/dbt/parser/manifest.py", line 1797, in parse_manifest
    manifest = ManifestLoader.get_full_manifest(
File "/usr/local/lib/python3.9/site-packages/dbt/parser/manifest.py", line 318, in get_full_manifest
    manifest = loader.load()
File "/usr/local/lib/python3.9/site-packages/dbt/parser/manifest.py", line 443, in load
    self.load_and_parse_macros(project_parser_files)
File "/usr/local/lib/python3.9/site-packages/dbt/parser/manifest.py", line 617, in load_and_parse_macros
    block = FileBlock(self.manifest.files[file_id])
KeyError: 'dbt_bigquery://macros/adapters.sql'
```

- Just run this commands to make something similar to “clean install” to remove logs , files in target folder and artifacts that may generate issues by running dbt:

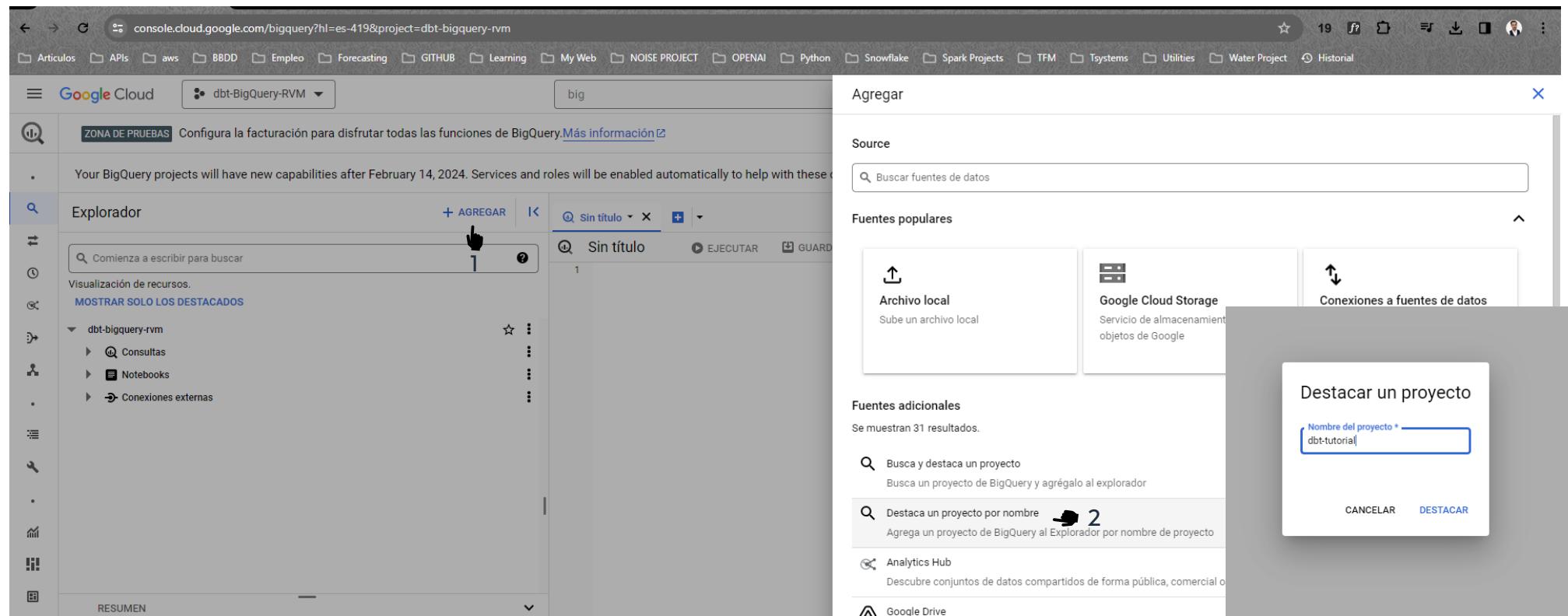
```
> dbt clean
> dbt deps
```

- After that, rerun dbt

```
- models.dbt_bigquery.example
14:49:20 Found 2 models, 4 tests, 0 sources, 0 exposures, 0 metrics, 454 macros, 0 groups, 0 semantic models
14:49:20
14:49:23 Concurrency: 1 threads (target='dev')
14:49:23
14:49:23 1 of 2 START sql table model dbt_bigquery_main.my_first_dbt_model ..... [RUN]
14:49:26 1 of 2 OK created sql table model dbt_bigquery_main.my_first_dbt_model ..... [CREATE TABLE (2.0 rows, 0 processed) in 3.28s]
14:49:26 2 of 2 START sql view model dbt_bigquery_main.my_second_dbt_model ..... [RUN]
14:49:27 2 of 2 OK created sql view model dbt_bigquery_main.my_second_dbt_model ..... [CREATE VIEW (0 processed) in 1.41s]
14:49:27
14:49:27 Finished running 1 table model, 1 view model in 0 hours 0 minutes and 7.66 seconds (7.66s).
14:49:27
14:49:27 Completed successfully
```

9. CREATE DBT MODEL AND ITS DAG

- 1. ADD new Project by name inside our existing BigQuery Project
- 2. Search by name: **dbt-tutorial**



9. CREATE DBT MODEL AND ITS DAG

- **3.** See this Project added and marked as favourite
- This data are offered from Google for free to practice with dbt-BigQuery
- We are extracting this data from **jaffle_shop_customers** and **jaffle_shop_orders** tables to use them in our dbt model and creating a new table named: customers
- **4. Create the model customers.sql**

The screenshot shows the Google Cloud BigQuery interface. In the 'Explorador' (Explorer) view, a yellow box highlights the 'jaffle_shop' dataset. Inside 'jaffle_shop', there are two tables: 'customers' and 'orders'. Other datasets like 'data_prep' and 'stripe' are also visible. The top navigation bar shows 'Google Cloud' and 'dbt-BigQuery-RVM'.

The screenshot shows a code editor with the file 'customers.sql' open. A yellow box highlights the first part of the code where the 'customers' table is defined. The code uses a WITH clause to select from 'jaffle_shop_customers' and 'jaffle_shop_orders' tables. The code editor interface includes a sidebar with project files and a status bar at the bottom.

```
with customers as (
    select id,
           first_name,
           last_name
      from `dbt-tutorial`.jaffle_shop.customers
),
orders as (
    select id,
           user_id,
           order_date,
           status
      from `dbt-tutorial`.jaffle_shop.orders
),
customer_orders as (
    SELECT
        user_id,
        min(order_date) as first_order,
        max(order_date) as most_recent_order,
        count(id) as number_of_orders
    from orders
    group by user_id
),
final as (
    select
        customers.id,
        customers.first_name,
        customers.last_name,
        customer_orders.first_order,
        customer_orders.most_recent_order,
        customer_orders.number_of_orders
    from customers
    left join customer_orders
    on customers.id = customer_orders.user_id
)
select * from final
```

9. CREATE DBT MODEL AND ITS DAG

- **5.** ReRun the DAG (created in section 8.4 if you want to test step by step)
- **6.** See how the DAG worked and the model was pushed into BigQuery here the customer table was materialized as view.

We will materialize it as table in the next steps but dockerizing the dbt Project and applying CI/CD You can even write a query to see the data in customers table view.

The screenshot shows the Google Cloud BigQuery schema browser. The left sidebar shows a tree structure of datasets and tables. A red box highlights the 'dbt-bigquery-rvm' dataset, which contains a 'dbt_bigquery_main' table that has a 'customers' table. The main panel displays the schema for the 'customers' table, which includes columns: id (INTEGER, NULLABLE), first_name (STRING, NULLABLE), last_name (STRING, NULLABLE), first_order (DATE, NULLABLE), most_recent_order (DATE, NULLABLE), and number_of_orders (INTEGER, NULLABLE). There are buttons for 'EDITAR ESQUEMA' (Edit Schema) and 'CONSULTA' (Query).

The screenshot shows the Google Cloud BigQuery results page. The top bar indicates a query named 'Sin título 2'. The results table shows 12 rows of data from the 'customers' table. The columns are: Fila (Row), id, first_name, last_name, first_order, most_recent_order, and number_of_orders. The data includes various names like Anna, Mildred, Maria, Adam, Harry, Jacqueline, Benjamin, Scott, Anne, Alan, Phillip, and Jimmy, along with their respective order details.

Fila	id	first_name	last_name	first_order	most_recent_order	number_of_orders
1	20	Anna	A.	2018-01-23	2018-01-23	1
2	23	Mildred	A.	null	null	null
3	40	Maria	A.	2018-01-17	2018-01-17	1
4	59	Adam	A.	2018-01-15	2018-01-15	1
5	74	Harry	A.	null	null	null
6	96	Jacqueline	A.	null	null	null
7	27	Benjamin	B.	2018-02-21	2018-04-04	2
8	45	Scott	B.	null	null	null
9	53	Anne	B.	2018-01-12	2018-03-11	2
10	73	Alan	B.	null	null	null
11	87	Phillip	B.	null	null	null
12	4	Jimmy	C	null	null	null

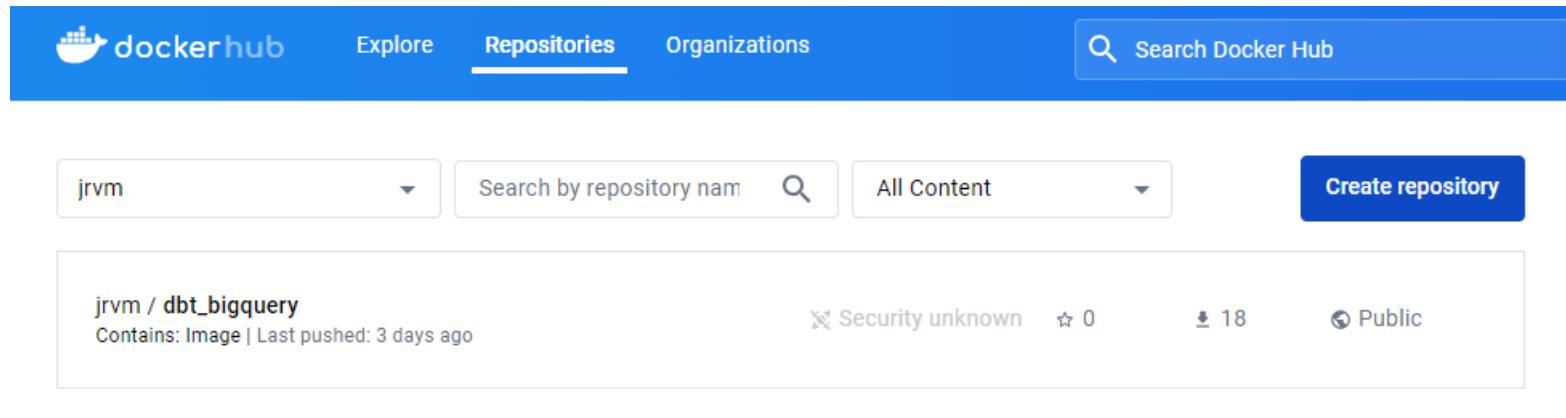
9. CREATE DBT MODEL AND ITS DAG

- At this point you can start creating all the DAGs from the repo which dag_id match the image below
- You can start creating one by one or all together (let you decide) but don't forget to **commit and push**.
- The goal now is to **implement** in all the DAGs related with dbt the **KubernetesPodOperator** controller.
- dbt_bigquery_test.py (**dbt_dag_test**) -> executes a simple directory listing command
- dbt_bigquery_test2.py (**dbt_debug_dag**) -> executes dbt debug in this dockerized dbt Project.
- dbt_bq_final.py (**dbt_full_workflow**) -> executes subsequently dbt run, dbt seed, dbt test
- Before running this DAGS we need to dockerize the dbt Project and set the CI/CD with **Github actions**

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks
dbt_dag_test	airflow	2	1 day, 0:00:00	2024-03-08, 11:36:19	2024-03-08, 00:00:00	1
dbt_debug_dag	airflow	1	1 day, 0:00:00	2024-03-07, 00:00:00	2024-03-08, 00:00:00	1
dbt_full_workflow	airflow	3	1 day, 0:00:00	2024-03-08, 20:46:34	2024-03-08, 00:00:00	3
dbt_TEST	rafaverama	5	1 day, 0:00:00	2024-03-08, 15:11:58	2024-03-08, 00:00:00	2
First_K8s	rafaverama	5	1 day, 0:00:00	2024-03-08, 09:20:34	2024-03-08, 00:00:00	2

10. SETUP DOCKER HUB

- Before creating de Docker image from the dockerfile, ensure you have an active Docker hub and a repository created in there, where the Docker image will be pushed with GitHub Actions.
- The process is pretty straightforward:
Access/ create the account -> <https://hub.docker.com/>
Create a repository. That's it.



11. DOCKERIZE THE DBT PROJECT

- **1.** Save all the dependencias needed into a requirements.txt
> Pip freeze -> requirements.txt
- **2.** Create the dockerfile
Please be carefull in which directory you copy the Project inside the image.
In my case, I copied it directly in the root directory with the same name of origin.
- **3.** Keep the dbt model ready: **customers.sql**
- **4.** Create the **docker-build-and-push.yaml** (Github Actions workflow)
- **5.** Save your secrets as environment variables in your Github repository (otherwise Github Actions fail)
- **6. Commit and push** from local to github
- **7. It should trigger the CI/CD deployment building the image and pushing it to dockerhub**

2

```
FROM python:3.9.13

# Update and install system packages
RUN apt-get update -y && \
    apt-get install -no-install-recommends -y -q git libpq-dev && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

# Set working directory
WORKDIR /dbt_bigquery_main

# Copy requirements and install DBT
COPY requirements.txt .
RUN pip install -U pip && \
    pip install -r requirements.txt

# Add your dbt project to the Docker image
COPY dbt_bigquery_main .

# Run dbt clean and dbt deps
RUN dbt clean && \
    dbt deps --project-dir .
```

3

```
with customers as (
    select id,
           first_name,
           last_name
      from `dbt-tutorial`.jaffle_shop.customers
),
orders as (
    select id,
           user_id,
           order_date,
           status
      from `dbt-tutorial`.jaffle_shop.orders
),
customer_orders as (
    SELECT
        user_id,
        min(order_date) as first_order,
        max(order_date) as most_recent_order,
        count(id) as number_of_orders
       from orders
      group by user_id
),
final as (
    select
        customers.id,
        customer.first_name,
        customers.last_name,
        customer_orders.first_order,
        customer_orders.most_recent_order,
        customer_orders.number_of_orders
       from customers
      left join customer_orders
         on customers.id = customer_orders.user_id
)
select
    customers.id,
    customer.first_name,
    customers.last_name,
    customer_orders.first_order,
    customer_orders.most_recent_order,
    customer_orders.number_of_orders
   from customers
      left join customer_orders
         on customers.id = customer_orders.user_id
```

4

```
name: Build and Push Docker
on:
  push:
    branches:
      - main
  paths:
    - 'dbt_bigquery_main/**'

jobs:
  build-and-push:
    runs-on: Ubuntu-20.04
    steps:
      - uses: actions/checkout@v2

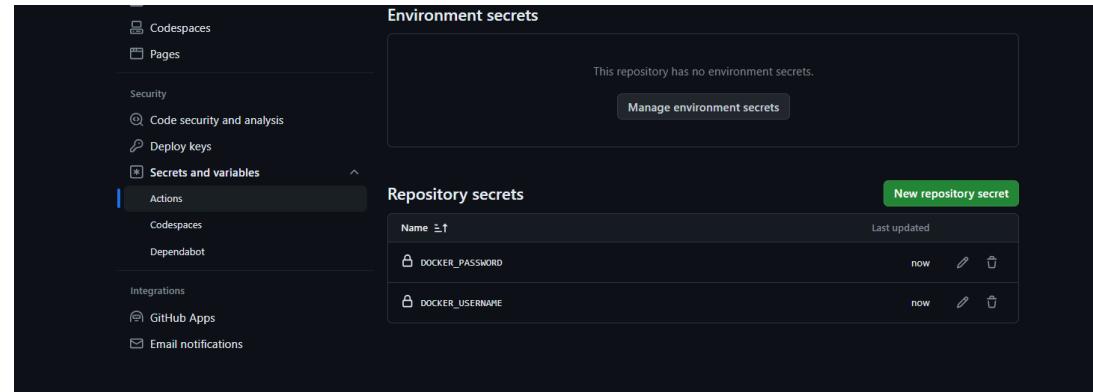
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1

      - name: Log in to Docker Hub
        uses: docker/login-action@v1
        with:
          username: ${secrets.DOCKER_USERNAME}
          password: ${secrets.DOCKER_PASSWORD}

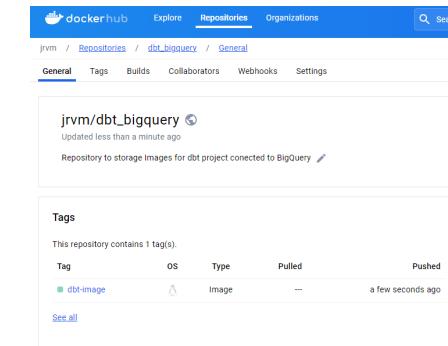
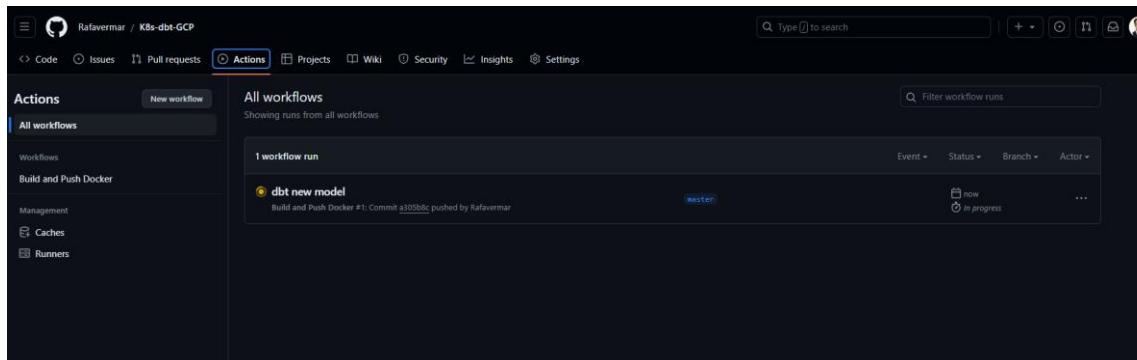
      - name: Build and push Docker image
        uses: docker/build-push-action@v2
        with:
          context: .
          push: true
          tags: jrvs/dbt_bigquery:dbt-image
```

11. DOCKERIZE THE DBT PROJECT

- **5.** Save your secrets as environment variables in your Github repository (otherwise Github Actions fail)



- **7. It should trigger the CI/CD deployment building the image and pushing it to dockerhub**
if the Action does not start, just make a tiny change into the README.md (inside dbt Project folder) and commit + push again.
This Action is configured to trigger when there are any changes inside the dbt Project folder only.



12. ORCHESTRATE CI/CD WITH GITHUB ACTIONS

- Before running the DAG this time, you can delete the models created before in BigQuery. Thus you can see that all is working as expected.
- If you run now the Airflow DAG from the UI, the image will be pulled from Docker hub by the KubernetesPodOperator into a temporary Pod inside K8s and Airflow will find this pod and run the defined dbt commands.
- Finally you can see the example models and the new model as a view in BigQuery.
- This customers view can be explored again with sql querying

The screenshot shows the Google Cloud BigQuery Explorer interface. The left sidebar lists projects: 'dbt-bigquery-rvm' (selected), 'my_first_dbt_model', 'my_second_dbt_model', and 'dbt-tutorial'. Under 'dbt-bigquery-rvm', there are 'Consultas', 'Notebooks', 'Conexiones externas', and a folder named 'dbt_bigquery_main' which contains the 'customers' table. The main panel displays the schema for the 'customers' table:

Nombre del campo	Tipo	Modo	Clave	Intercalación
id	INTEGER	NULLABLE	-	-
first_name	STRING	NULLABLE	-	-
last_name	STRING	NULLABLE	-	-
first_order	DATE	NULLABLE	-	-
most_recent_order	DATE	NULLABLE	-	-
number_of_orders	INTEGER	NULLABLE	-	-

At the bottom right of the main panel, there is a blue button labeled 'EDITAR ESQUEMA'.

The screenshot shows the Google Cloud BigQuery Explorer interface. The left sidebar lists projects: 'dbt-bigquery-rvm' (selected), 'my_first_dbt_model', 'my_second_dbt_model', and 'dbt-tutorial'. Under 'dbt-bigquery-rvm', there are 'Consultas', 'Notebooks', 'Conexiones externas', and a folder named 'dbt_bigquery_main' which contains the 'customers' table. The main panel shows a query in the editor:

```
SELECT * FROM `dbt-bigquery-rvm.dbt_bigquery_main.customers` LIMIT 1000
```

Below the editor, the results of the query are displayed in a table:

Fila	id	first_name	last_name	first_order	most_recent_order	number_of_orders
1	20	Anna	A.	2018-01-23	2018-01-23	1
2	23	Mildred	A.	null	null	null
3	40	Maria	A.	2018-01-17	2018-01-17	1
4	59	Adam	A.	2018-01-15	2018-01-15	1
5	74	Harry	A.	null	null	null
6	96	Jacqueline	A.	null	null	null
7	27	Benjamin	B.	2018-02-21	2018-04-04	2
8	45	Scott	B.	null	null	null
9	53	Anne	B.	2018-01-12	2018-03-11	2
10	73	Alan	B.	null	null	null
11	87	Philip	B.	null	null	null
12	4	Jimmv	C.	null	null	null

12. ORCHESTRATE CI/CD WITH GITHUB ACTIONS

- Let's make changes inside dbt Project to keep triggering the deployed Action.
- You can change the table materialization as table instead of as view.
- As made before:

commit + push -> Action is triggered -> Docker image is build and pushed to Docker Hub -> Run Airflow Dag

- The customers table is persisted in Bigquery as a table.

The screenshot shows the Google Cloud BigQuery interface. On the left, there's a sidebar with project navigation and a search bar. The main area displays the 'customers' table from the 'dbt_bigquery_main' dataset. The table has columns: id, first_name, last_name, first_order, most_recent_order, and number_of_orders. The data shows 15 rows of customer information, such as Mildred, Jacqueline, Harry, Adam, Maria, Anna, Alan, Phillip, Scott, Anne, Benjamin, Lillian, Jimmy, Jason, and Gloria.

The screenshot shows a GitHub Actions workflow configuration file named 'dbt-build-and-push.yml'. The file defines a workflow with steps for building dbt models, pushing them to Docker Hub, and running an Airflow DAG. It includes configurations for profiles, model paths, and specific dbt models like 'customers' and 'seed'. The file also specifies clean targets and logs.

```
version: '1.0.0'
config-version: 2

# This setting configures which "profile" dbt uses for this project.
profile: "dbt_bigquery_main"

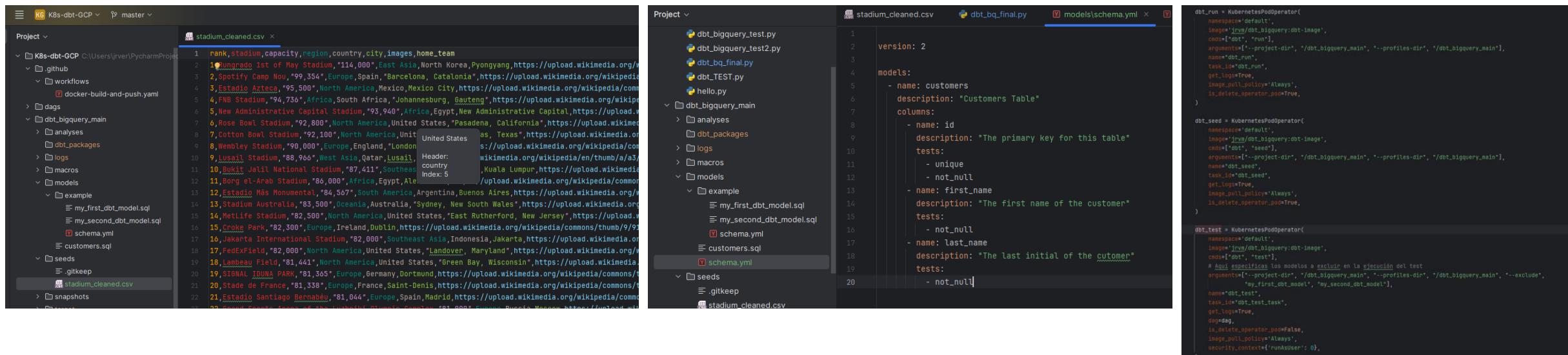
# These configurations specify where dbt should look for different types of files.
# The "model-paths" config, for example, states that models in this project can be found in the "models/" directory. You probably won't need to change these!
model-paths: ["models"]
analysis-paths: ["analyses"]
test-paths: ["tests"]
seed-paths: ["seeds"]
macro-paths: ["macros"]
snapshot-paths: ["snapshots"]

clean-targets: # directories to be removed by 'dbt clean'
- "target"
- "dbt_packages"

# Configuring models
# Full documentation: https://docs.getdbt.com/docs/configuring-models
# In this example config, we tell dbt to build all models in the example/ directory as views. These settings can be overridden in the individual model files using the `{{ config(...) }}` macro.
models:
  dbt_bigquery:
    # Config indicated by + and applies to all files under models/example/
    example:
      +materialized: view
      materialized: table
```

12. ORCHESTRATE CI/CD WITH GITHUB ACTIONS

- Now you can use the same DAG before or just jump into the **dag_full_workflow**
- To make this **dbt_bq_final.py** (dag_full_workflow) works:
 1. Add the **.csv** in seed folder → to make the dbt seed command from the DAG Works
 2. Create a new **schema.yml** in models folders and write some test → to make dbt test command work
 3. Create the **dbt_bq_final.py** DAG
 4. Commit and push to Github
 5. Observe the progression and success of the Action in Github
 6. Observe the updated image in Docker Hub
 5. Run the DAG
 6. Observe the progress in Airflow UI
 7. Observe the results in Big Query



The screenshot shows a PyCharm interface with three main panes:

- Project View:** Shows the file structure of the K8s-dbt-GCP project. It includes a 'seeds' directory containing 'stadium_cleaned.csv', a 'models' directory with 'customers' and 'schema.yml' files, and a 'dags' directory.
- Code Editor:** Displays the **dbt_bq_final.py** file. The code defines a DAG with tasks for running dbt commands. It includes sections for `version`, `models`, and `seed` (which points to `stadium_cleaned.csv`). The `models` section specifies a schema for the 'customers' table and includes test definitions for columns like `id`, `first_name`, and `last_name`.
- Terminal:** Shows the output of the dbt command, indicating successful runs for various models like `customers` and `schema`.

12. ORCHESTRATE CI/CD WITH GITHUB ACTIONS

Actions New workflow

All workflows
Showing runs from all workflows

8 workflow runs

Workflow	Branch	Event	Status	Duration	...
DAG seed data	master	Build and Push Docker #8: Commit 7c137ad pushed by Rafavermar	15 hours ago	2m 05s	...
seed	master	Build and Push Docker #7: Commit 8b4c5ba pushed by Rafavermar	16 hours ago	2m 13s	...
quick fix materialized as table	master	Build and Push Docker #6: Commit 4a120ec pushed by Rafavermar	16 hours ago	2m 5s	...
quick fix customers.sql	master	Build and Push Docker #5: Commit 5794d78 pushed by Rafavermar	16 hours ago	1m 59s	...
quick fix `dbt-tutorial`	master	Build and Push Docker #4: Commit f3e77e9 pushed by Rafavermar	17 hours ago	1m 57s	...
quick fix `dbt-tutorial`	master	Build and Push Docker #3: Commit 710a8b6 pushed by Rafavermar	17 hours ago	1m 59s	...
quick fix	master	Build and Push Docker #2: Commit fdf079c pushed by Rafavermar	17 hours ago	1m 58s	...
dbt new model	master	Build and Push Docker #1: Commit a305b8c pushed by Rafavermar	17 hours ago	2m 32s	...

dockerhub Explore **Repositories** Organizations Search

jrvm / Repositories / dbt_bigquery / General

General Tags Builds Collaborators Webhooks Settings

jrvm/dbt_bigquery

Updated less than a minute ago

Repository to storage Images for dbt project conected to BigQuery

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
dbt-image		Image	---	a few seconds ago

[See all](#)

13. RUN DAG FULL WORKFLOW

Airflow DAGs Cluster Activity Datasets Security Browse Admin Docs 20:48 UTC AU

DAG: dbt_full_workflow A DAG to run dbt tasks including run, seed, and test

Schedule: 1 day, 0:00:00 Next Run ID: 2024-03-08, 00:00:00

Grid Graph Calendar Task Duration Task Tries Landing Times Gantt Details Code Audit Log

08/03/2024 20:46:39 25 All Run Types All Run States Clear Filters Auto-refresh

Press shift + / for Shortcuts

Duration: 00:06:04

dbt_run 00:03:02

dbt_seed 00:00:00

dbt_test_task

DAG dbt_full_workflow Run 2024-03-08, 20:46:34 UTC Task dbt_test_task

Details Graph Gantt Code Logs XCom

More Details Rendered Template K8s Pod Spec List Instances, all runs

Task Instance Notes: Add Note

Task Instance Details

Status	success
Task ID	dbt_test_task
Run ID	manual_2024-03-08T20:46:34.882346+00:00
Operator	KubernetesPodOperator
Trigger Rule	all_success
Duration	00:00:24
Started	2024-03-08, 20:48:14 UTC
Ended	2024-03-08, 20:48:38 UTC

Clear task Mark state as... Filter Tasks

13. RUN DAG FULL WORKFLOW

The screenshot shows the Airflow UI for a DAG named `dbt_full_workflow`. The current run is dated `2024-03-08, 20:46:34 UTC` and is focused on the task `dbt_test_task`. The `Logs` tab is selected, displaying the log output for attempt 1.

The log output for `dbt_test_task` shows the following sequence of events:

- Initial setup and adapter registration.
- Execution of four tests:
 - `test not_null_customers_first_name`: Started at 20:48:28 UTC, completed successfully at 20:48:29 UTC.
 - `test not_null_customers_id`: Started at 20:48:29 UTC, completed successfully at 20:48:30 UTC.
 - `test unique_customers_id`: Started at 20:48:30 UTC, completed successfully at 20:48:31 UTC.
 - `test not_null_customers_last_name`: Started at 20:48:31 UTC, completed successfully at 20:48:32 UTC.
- Summary message indicating all 4 tests were passed successfully.
- Final completion message at 20:48:33 UTC.

A yellow box highlights the log output for `dbt_test_task`, specifically the test results and their completion times.

```
[2024-03-08, 20:48:27 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:22 Running with dbt=1.7.9
[2024-03-08, 20:48:28 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:24 Registered adapter: bigquery=1.7.6
[2024-03-08, 20:48:28 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:24 Unable to do partial parsing because saved manifest not found. Starting full parse.
[2024-03-08, 20:48:28 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:26 Found 3 models, 1 seed, 8 tests, 0 sources, 0 exposures, 0 metrics, 454 macros, 0 groups, 0 semantic models
[2024-03-08, 20:48:28 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:26
[2024-03-08, 20:48:27 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:27 Concurrency: 1 threads (target= dev )
[2024-03-08, 20:48:27 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:27
[2024-03-08, 20:48:28 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:28 1 of 4 START test not_null_customers_first_name ..... [RUN]
[2024-03-08, 20:48:28 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:28 1 of 4 PASS not_null_customers_first_name ..... [32mPASS[0m in 1.57s]
[2024-03-08, 20:48:29 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:29 2 of 4 START test not_null_customers_id ..... [RUN]
[2024-03-08, 20:48:29 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:29 2 of 4 PASS not_null_customers_id ..... [32mPASS[0m in 1.63s]
[2024-03-08, 20:48:30 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:30 3 of 4 START test not_null_customers_last_name ..... [RUN]
[2024-03-08, 20:48:30 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:30 3 of 4 PASS not_null_customers_last_name ..... [32mPASS[0m in 1.66s]
[2024-03-08, 20:48:31 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:31 4 of 4 START test unique_customers_id ..... [RUN]
[2024-03-08, 20:48:31 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:31 4 of 4 PASS unique_customers_id ..... [32mPASS[0m in 1.70s]
[2024-03-08, 20:48:32 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:32
[2024-03-08, 20:48:32 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:32 Finished running 4 tests in 0 hours 0 minutes and 7.24 seconds (7.24s).
[2024-03-08, 20:48:33 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:33
[2024-03-08, 20:48:33 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:33 ⌂[32mCompleted successfully[0m
[2024-03-08, 20:48:33 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:33
[2024-03-08, 20:48:34 UTC] {pod_manager.py:483} INFO - [base] ⌂[0m20:48:34 Done. PASS=4 WARN=0 ERROR=0 SKIP=0 TOTAL=4
[2024-03-08, 20:48:34 UTC] {pod_manager.py:511} WARNING - Pod dbt-test-m44c15dq log read interrupted but container base still running
[2024-03-08, 20:48:34 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:34 4 of 4 PASS unique_customers_id ..... [32mPASS[0m in 1.70s]
[2024-03-08, 20:48:35 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:35
[2024-03-08, 20:48:35 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:35 Finished running 4 tests in 0 hours 0 minutes and 7.24 seconds (7.24s).
[2024-03-08, 20:48:36 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:36
[2024-03-08, 20:48:36 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:36 ⌂[32mCompleted successfully[0m
[2024-03-08, 20:48:36 UTC] {pod_manager.py:466} INFO - [base] ⌂[0m20:48:36
[2024-03-08, 20:48:37 UTC] {pod_manager.py:483} INFO - [base] ⌂[0m20:48:37 Done. PASS=4 WARN=0 ERROR=0 SKIP=0 TOTAL=4
[2024-03-08, 20:48:38 UTC] {pod_manager.py:616} INFO - Pod dbt-test-m44c15dq has phase Running
[2024-03-08, 20:48:38 UTC] {pod.py:912} INFO - Skipping deleting pod: dbt-test-m44c15dq
```

13. RUN DAG FULL WORKFLOW

Airflow DAGs Cluster Activity Datasets Security Browse Admin Docs

DAGs

All 5 Active 1 Paused 4 Running 0 Failed 0 Filter DAGs by tag Search DAGs

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks
dbt_dag_test	airflow	2	1 day, 0:00:00	2024-03-08, 11:36:19	2024-03-08, 00:00:00	1
dbt_debug_dag	airflow	1	1 day, 0:00:00	2024-03-07, 00:00:00	2024-03-08, 00:00:00	1
dbt_full_workflow	airflow	3	1 day, 0:00:00	2024-03-08, 20:46:34	2024-03-08, 00:00:00	3
dbt_TEST	rafaverama	5	1 day, 0:00:00	2024-03-08, 15:11:58	2024-03-08, 00:00:00	2
First_K8s	rafaverama	5	1 day, 0:00:00	2024-03-08, 09:20:34	2024-03-08, 00:00:00	2

« < 1 > »

13. RUN DAG FULL WORKFLOW

- See all the tables materialized as Table included the seeded .csv

The screenshot shows the Google Cloud BigQuery interface. On the left, there's a sidebar with various icons and a search bar. The main area has a header with 'Google Cloud' and 'dbt-BigQuery-RVM'. A search bar at the top right says 'Buscar (/) recursos, documentos, productos y más'. The main content area shows a table titled 'stadium_cleaned' with the following data:

Fila	rank	stadium	capacity	region	country	city
1	233	Stade du 26 mars	50000	Africa	Mali	Bamako
2	5	New Administrative Capital Sta...	93940	Africa	Egypt	New Ad...
3	11	Borg el-Arab Stadium	86000	Africa	Egypt	Alexand...

The table has tabs for 'ESQUEMA', 'DETALLES', 'VISTA PREVIA', 'LINAJE', 'PERFIL DE DATOS', and 'CALIDAD DE LOS DATOS'. There are also buttons for 'CONSULTA', 'COMPARTE', 'COPIAR', 'INSTANTÁNEA', 'BORRAR', and 'EXPORTAR'.

13. RUN DAG FULL WORKFLOW

The screenshot shows the Google Cloud BigQuery schema editor interface. The top navigation bar includes the Google Cloud logo, a dropdown for the project 'dbt-BigQuery-RVM', and a search bar. The main workspace displays the 'stadium_cleaned' table schema under the 'stadium_cleaned' dataset. The schema consists of eight columns:

Nombre del campo	Tipo	Modo	Clave	Intercalación	Valor predeterminado	Etiquetas de políticas	Descripción
rank	INTEGER	NULLABLE	-	-	-	-	-
stadium	STRING	NULLABLE	-	-	-	-	-
capacity	INTEGER	NULLABLE	-	-	-	-	-
region	STRING	NULLABLE	-	-	-	-	-
country	STRING	NULLABLE	-	-	-	-	-
city	STRING	NULLABLE	-	-	-	-	-
images	STRING	NULLABLE	-	-	-	-	-
home_team	STRING	NULLABLE	-	-	-	-	-

Below the schema, there are buttons for 'EDITAR ESQUEMA' and 'VER POLÍTICAS DE ACCESO DE FILA'. The left sidebar shows the project structure, including datasets like 'dbt-bigquery-rvm' and 'dbt-tutorial', and specific tables like 'customers', 'my_first_dbt_model', and 'stadium_cleaned'. A summary section at the bottom provides details about the last modification and location.

DBT AUTOMATED DOCUMENTATION GENERATION

- Note: The documentation shows up on localhost 8080, the same port of Airflow K8s pod UI.
 - > dbt docs generate
 - > dbt docs serve

The screenshot shows the PyCharm IDE interface with the following details:

- Project View:** The left sidebar displays the project structure under "dbt_bigquery_main". It includes sections for workflows, dags, analyses, dbt_packages, logs, macros, models, example (with files my_first_dbt_model.sql, my_second_dbt_model.sql, schema.yml), seeds, and seeds.
- Terminal:** The bottom pane shows a PowerShell session. The user runs "cd .\dbt_bigquery_main" and "dbt docs generate". The output indicates 3 models, 1 seed, 8 tests, 0 sources, 0 exposures, 0 metrics, 454 macros, 0 groups, 0 semantic models, and 1 thread target. A catalog is built and written to target\catalog.json.
- Status Bar:** The top bar shows "KG K8s-dbt-GCP" and "master".

The screenshot displays the dbt UI across three browser tabs, illustrating the tool's capabilities for managing data models.

- Overview Tab:** Shows the project structure under "dbt_bigquery_main". The "models" folder contains "customers" and "example". The "seeds" folder contains "stadium_cleaned".
- customers Table Details Tab:** Provides detailed information about the "customers" table.
 - Details Section:** Includes table metadata: TAGS (untagged), OWNER (dbt), TYPE (table), PACKAGE (dbt_bigquery_main), LANGUAGE (sql), RELATION (dbt-bigquery-rvm.dbt_bigquery_main.customers), ACCESS (protected), VERSION (1), and CONTRACT (Not Enforced). It also shows # ROWS (100) and APPROXIMATE SIZE (3 KB).
 - Description Section:** Contains a brief description: "Customers Table".
 - Columns Section:** Lists the columns and their properties:

COLUMN	TYPE	DESCRIPTION	CONSTRAINTS	TESTS	MORE?
id	INT64	The primary key for this table	UN	N	>
first_name	STRING	The first name of the customer	N	N	>
last_name	STRING	The last initial of the customer	N	N	>
first_order	DATE				
most_recent_order	DATE				
number_of_orders	INT64				
- stadium_cleaned Table Details Tab:** Provides detailed information about the "stadium_cleaned" table.
 - Overview Section:** Shows the table's details: TAGS (untagged), OWNER (dbt), TYPE (table), PACKAGE (dbt_bigquery_rvm), LANGUAGE (sql), RELATION (dbt-bigquery-rvm.dbt_bigquery_main.stadium_cleaned), ACCESS (protected), VERSION (1), and CONTRACT (Not Enforced). It also shows # ROWS (278) and APPROXIMATE SIZE (54 KB).
 - Description Section:** Contains a brief description: "This seed is not currently documented".
 - Columns Section:** Lists the columns and their properties:

COLUMN	TYPE	DESCRIPTION	CONSTRAINTS	TESTS	MORE?
region	INT64				
state	STRING				
city	STRING				
zip	STRING				
name	STRING				
 - Code Section:** Displays the source code for the table.