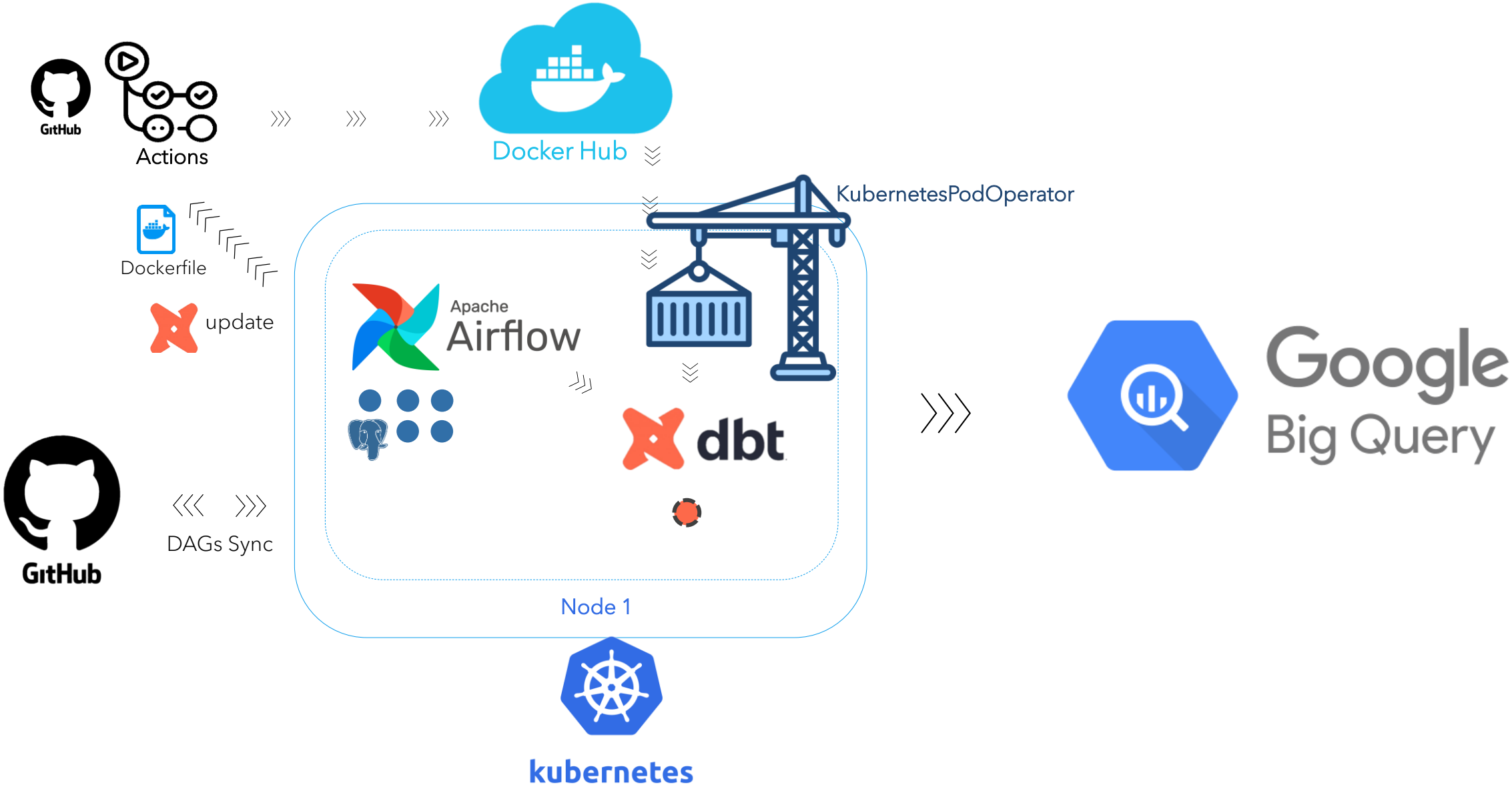
Automated BigQuery Transformations with dbt, Airflow, Kubernetes, and GitHub Actions

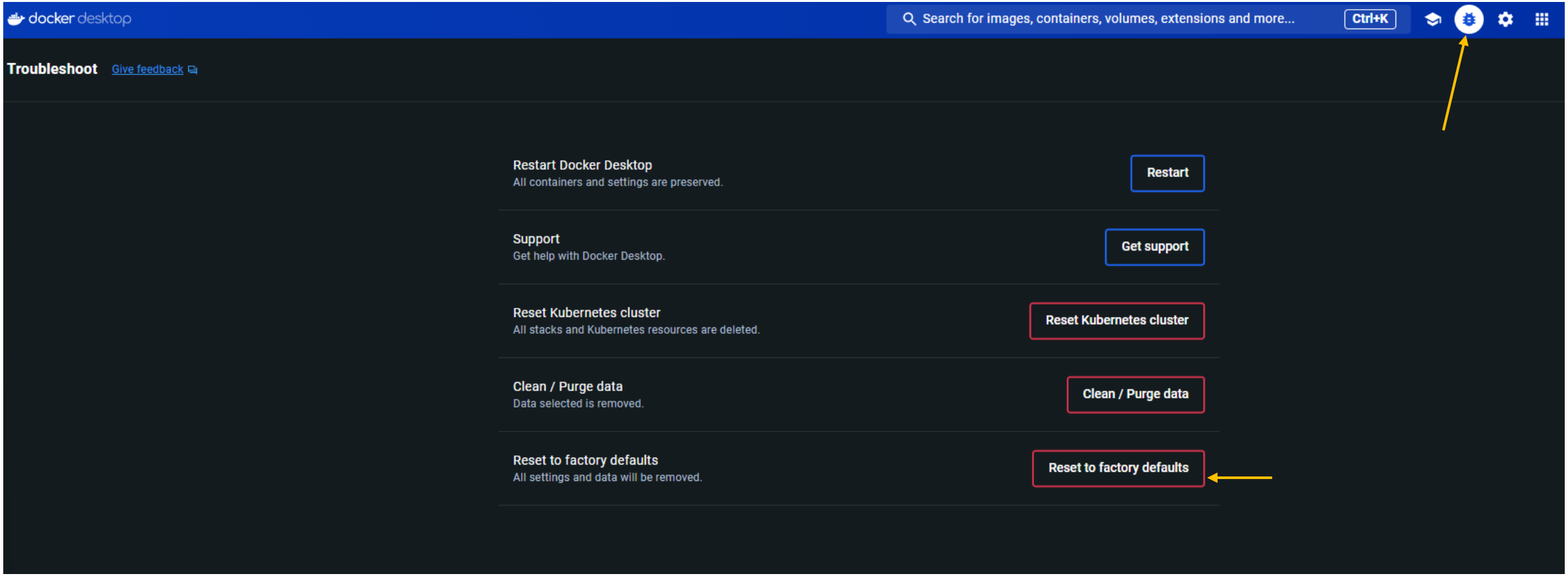


WORKFLOW

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.

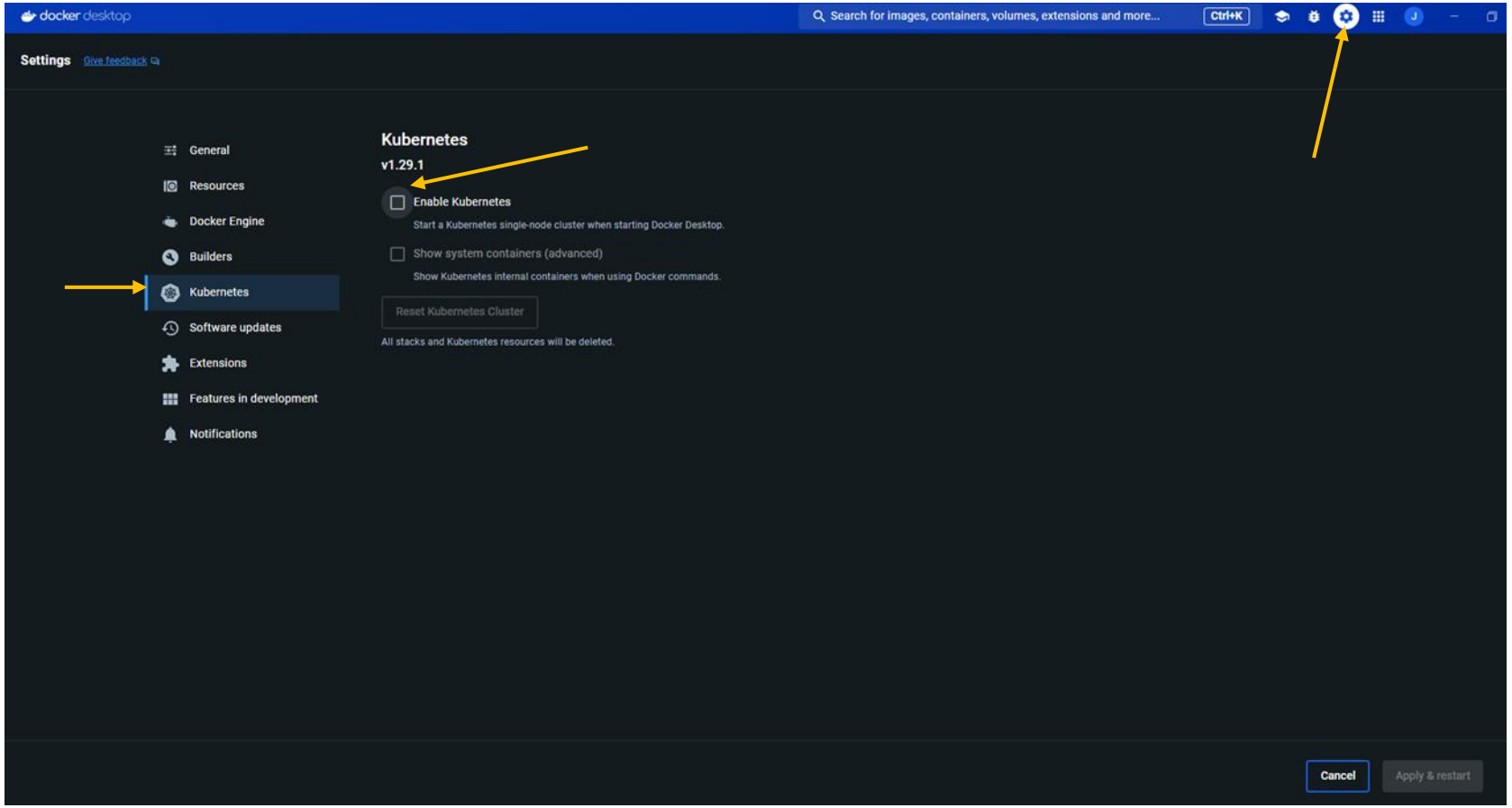
1. INITIALIZE KUBERNETES

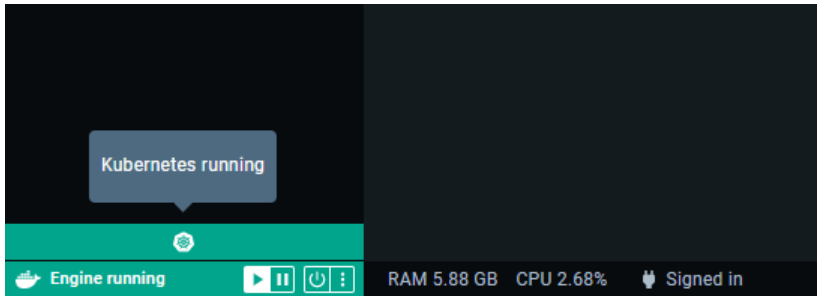
- Open Docker Desktop as Admin
- Reset to factory defaults



1. INITIALIZE KUBERNETES

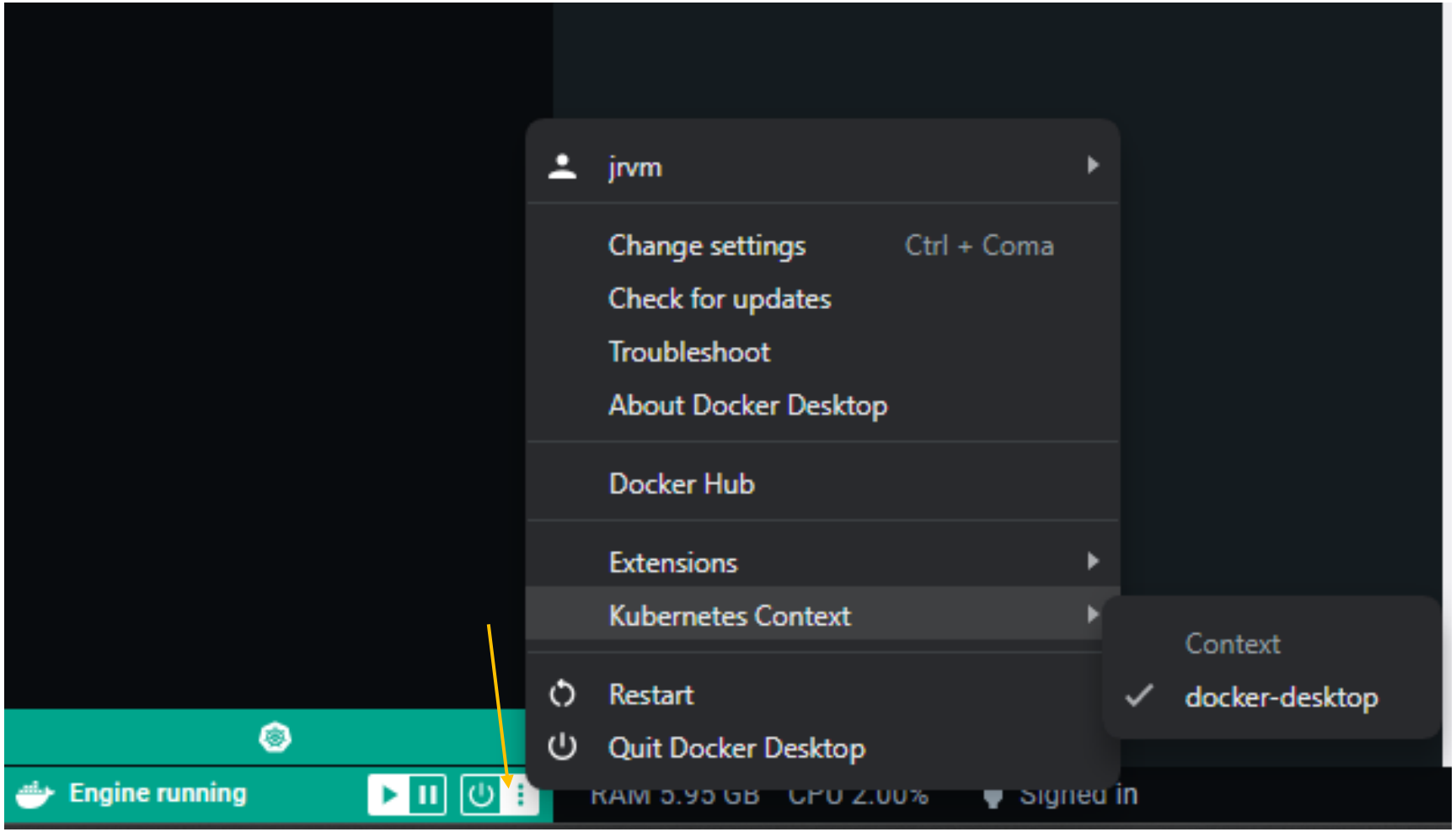
- Enable Kubernetes
- Apply & restart
- Wait until running-see bottom left





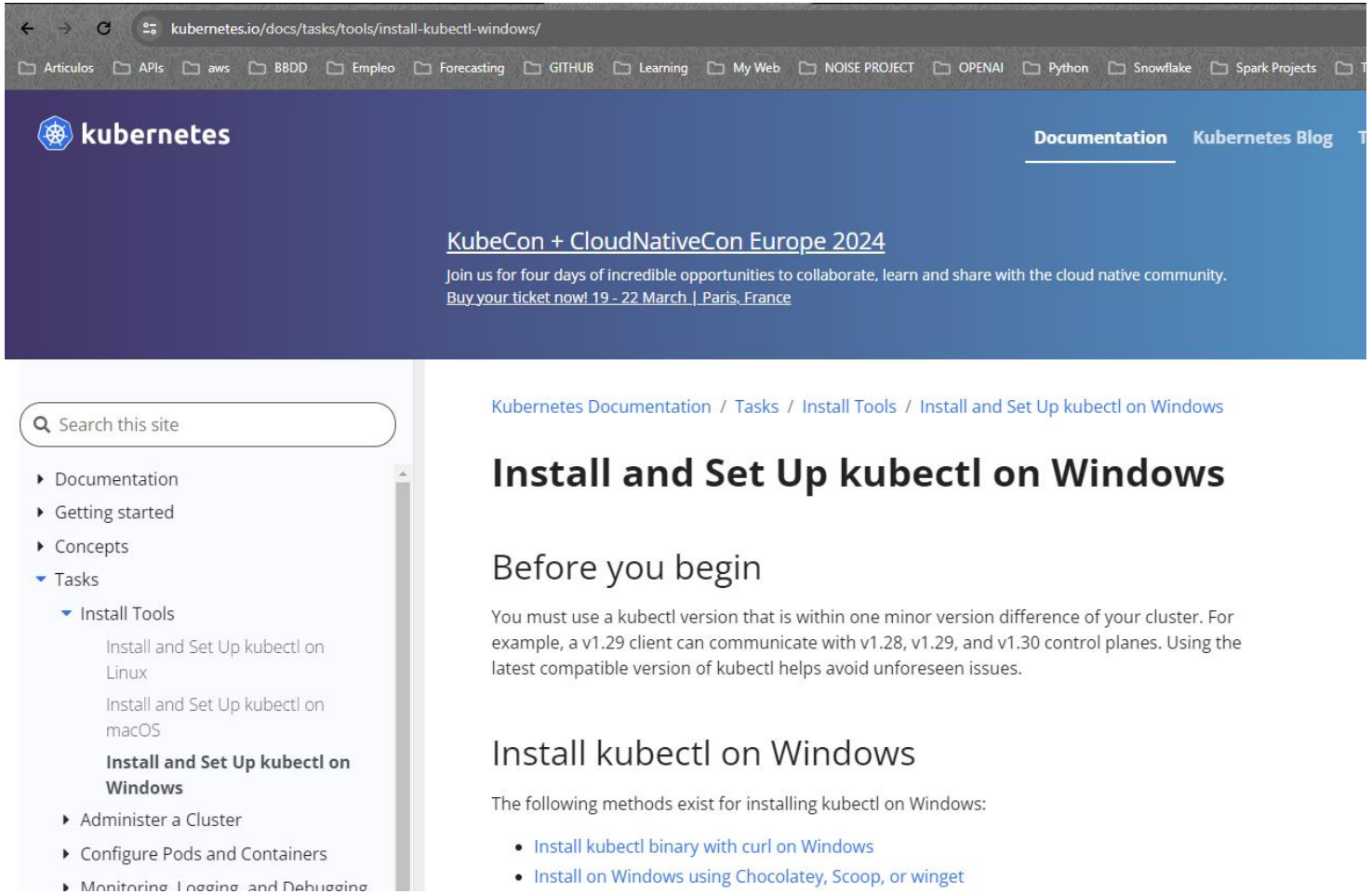
1. INITIALIZE KUBERNETES

- Ensure the kubernetes context is set as Docker-desktop



2. SET UP TOOLS

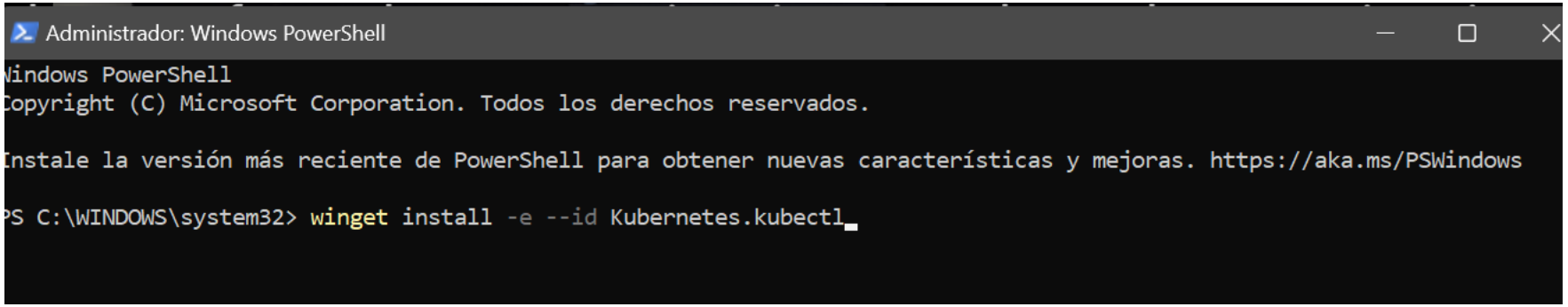
- **Kubectl**: follow the great official guide -> https://kubernetes.io/docs/tasks/tools/
- In my case I choosed Windows

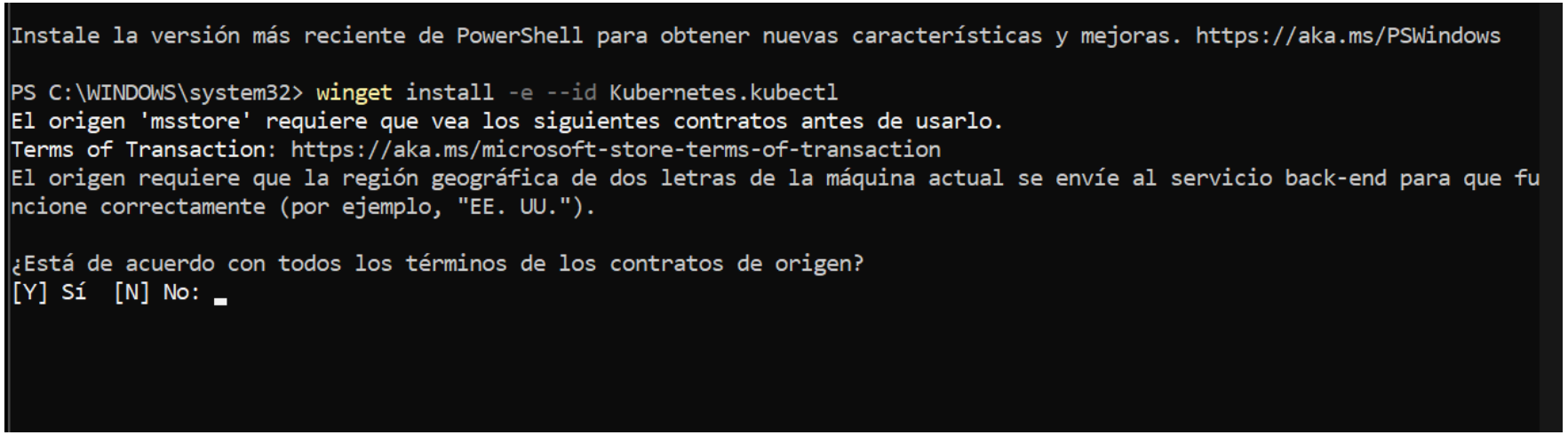


2. SET UP TOOLS

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

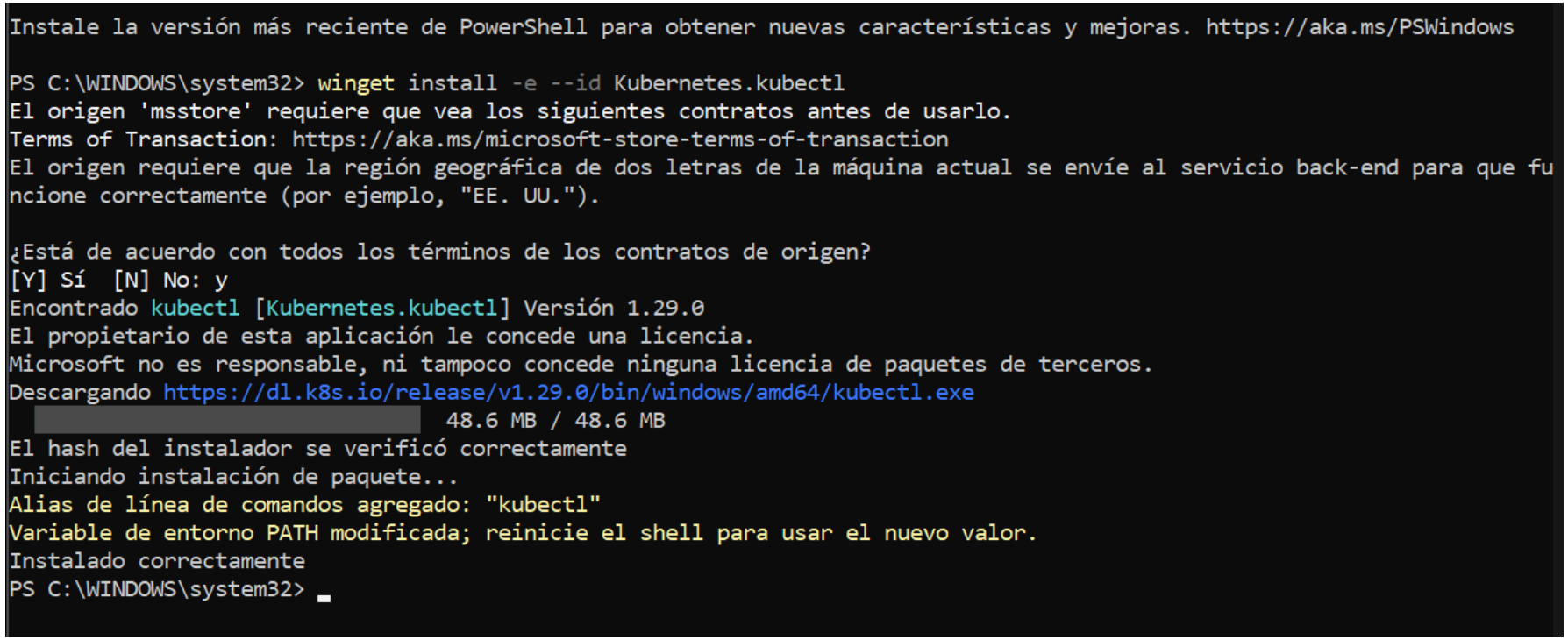
  > winget install –e –id Kubernetes.kubectl
  > Type : Y
- Wait for downloading





2. SET UP TOOLS

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



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 fu
ncione 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/

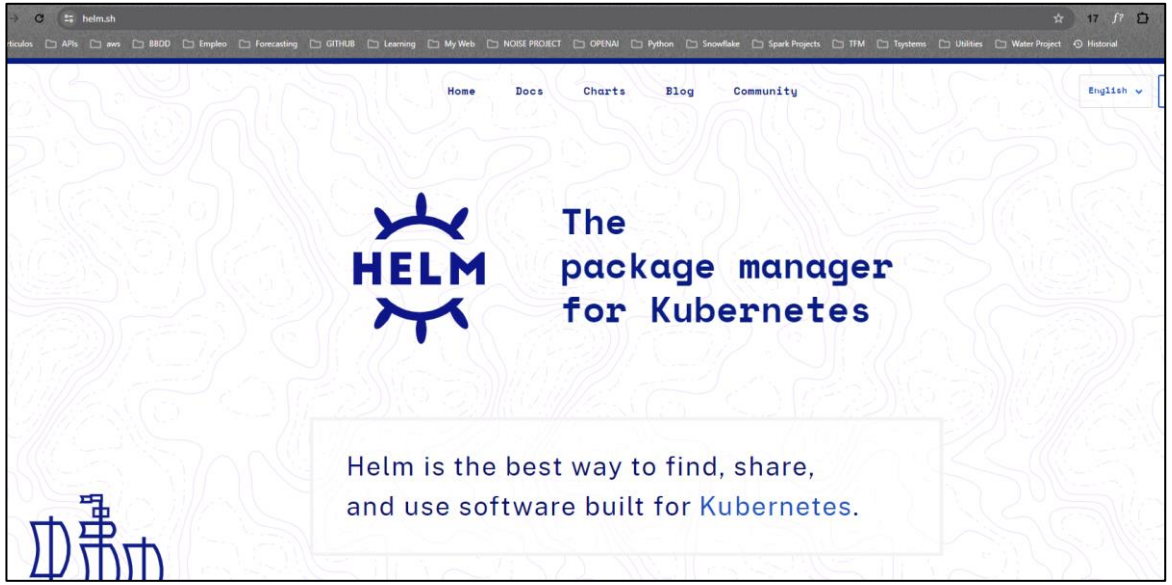
- Check kubernetes cluster info:
  > Kubectl cluster-info
- Further debug and diagnose cluster problems
  > Kubectl cluster-info dump

```
PS C:\Users\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 cluster-info dump
{
    "kind": "NodeList",
    "apiVersion": "v1",
    "metadata": {
        "resourceVersion": "3495"
    },
    "items": [
        {
            "metadata": {
                "name": "docker-desktop",
                "uid": "3ecea5f7-68e1-4f16-bc70-142d69fbd17f",
                "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**

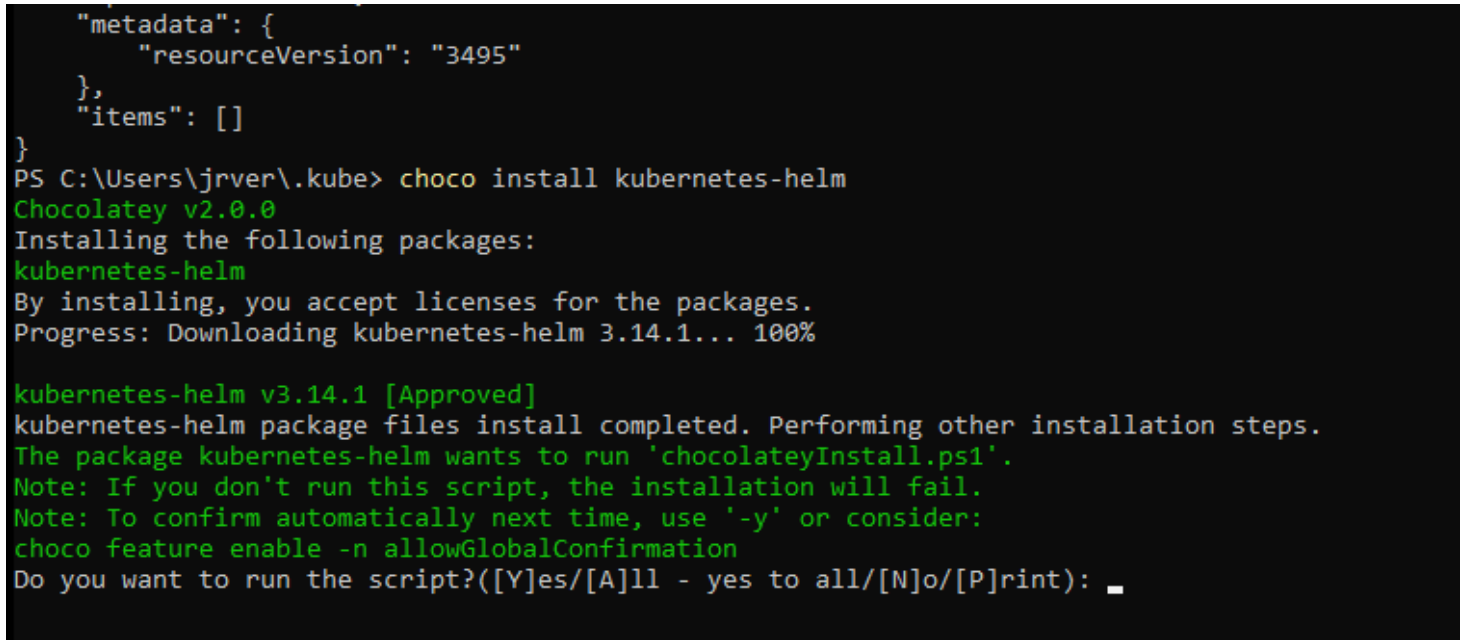


2. SET UP TOOLS

- **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



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

```
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

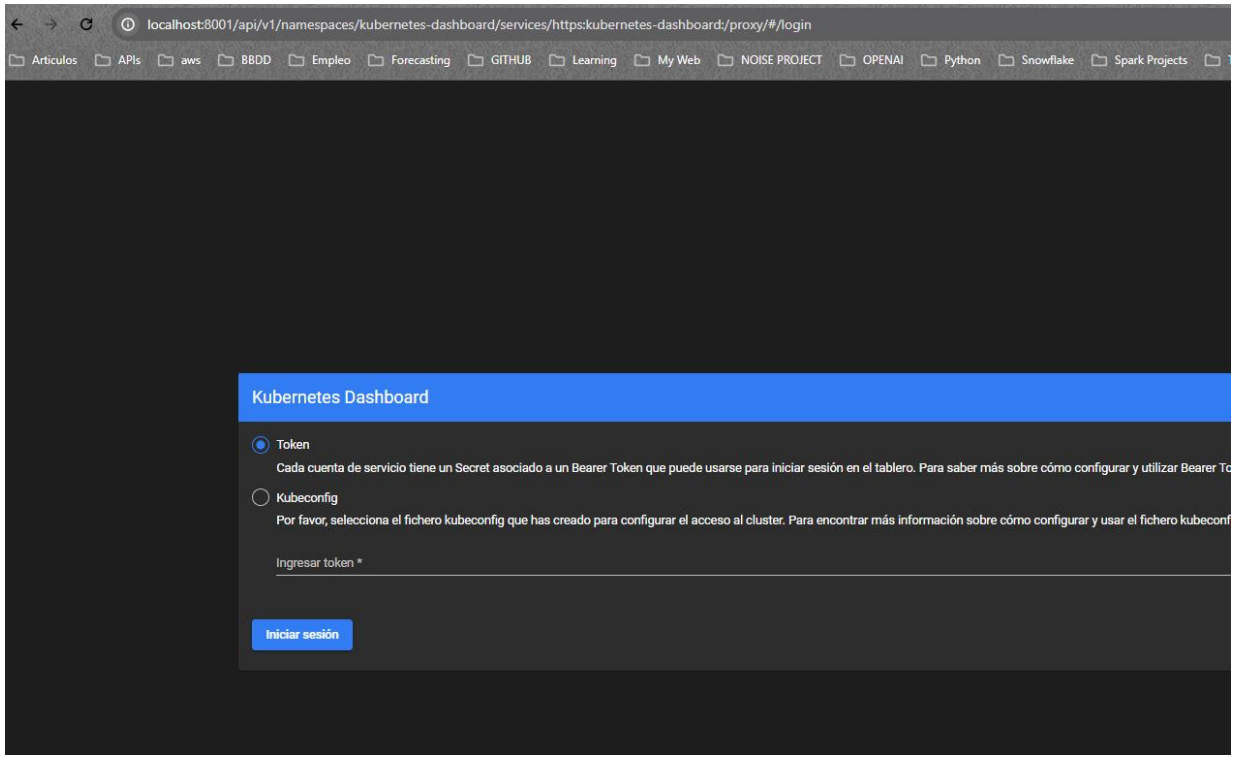
```
PS C:\Users\jrver\.kube> kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf 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 fro 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



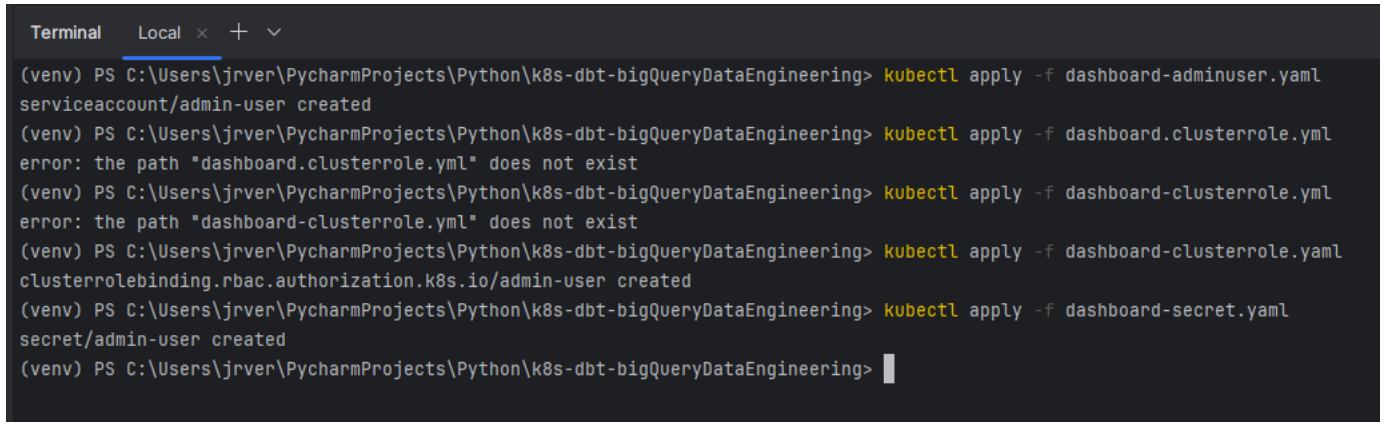
3. CONFIGURE KUBERNETES DASHBOARD

- 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

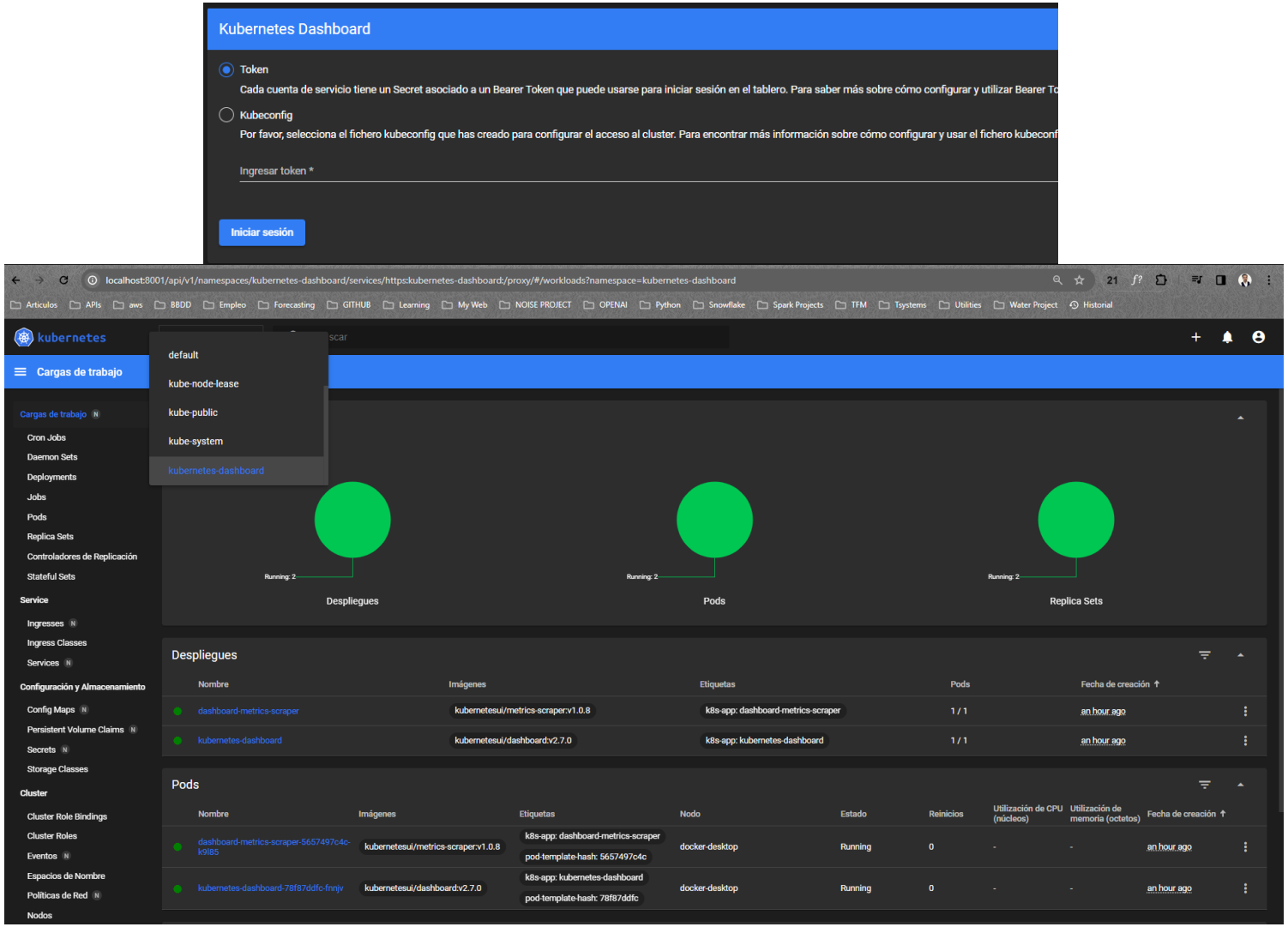


- Once created, run the get secret command, copy and paste it into the authentication placeholder.

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

3. CONFIGURE KUBERNETES DASHBOARD

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



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

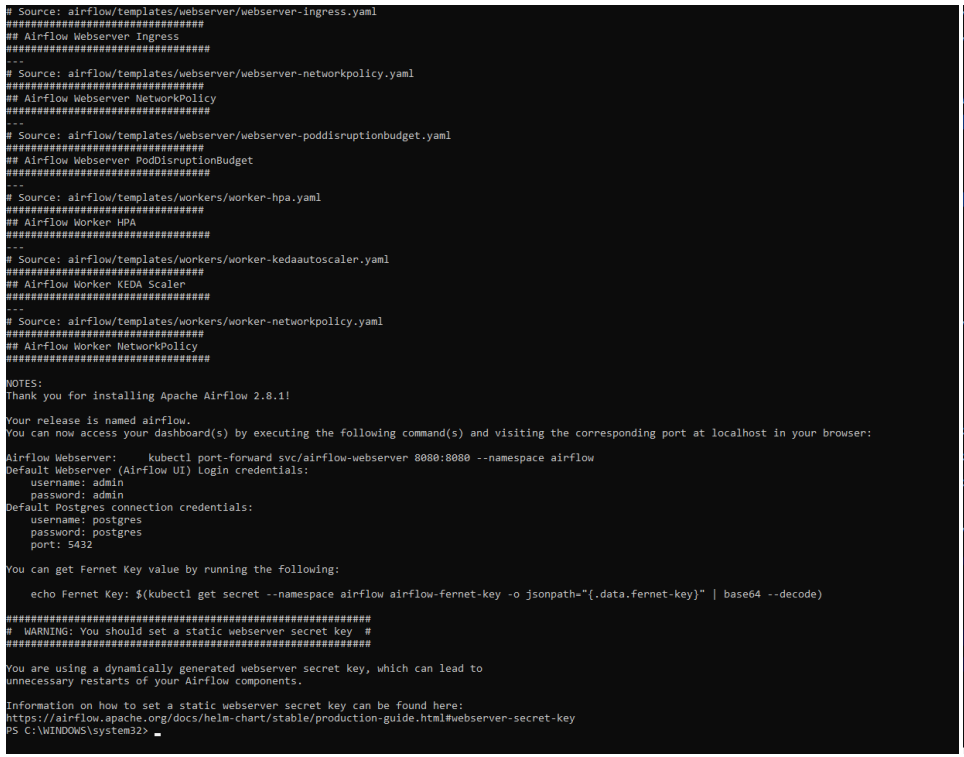
>helm install airflow apache-airflow/airflow --namespace airflow --create-namespace --debug

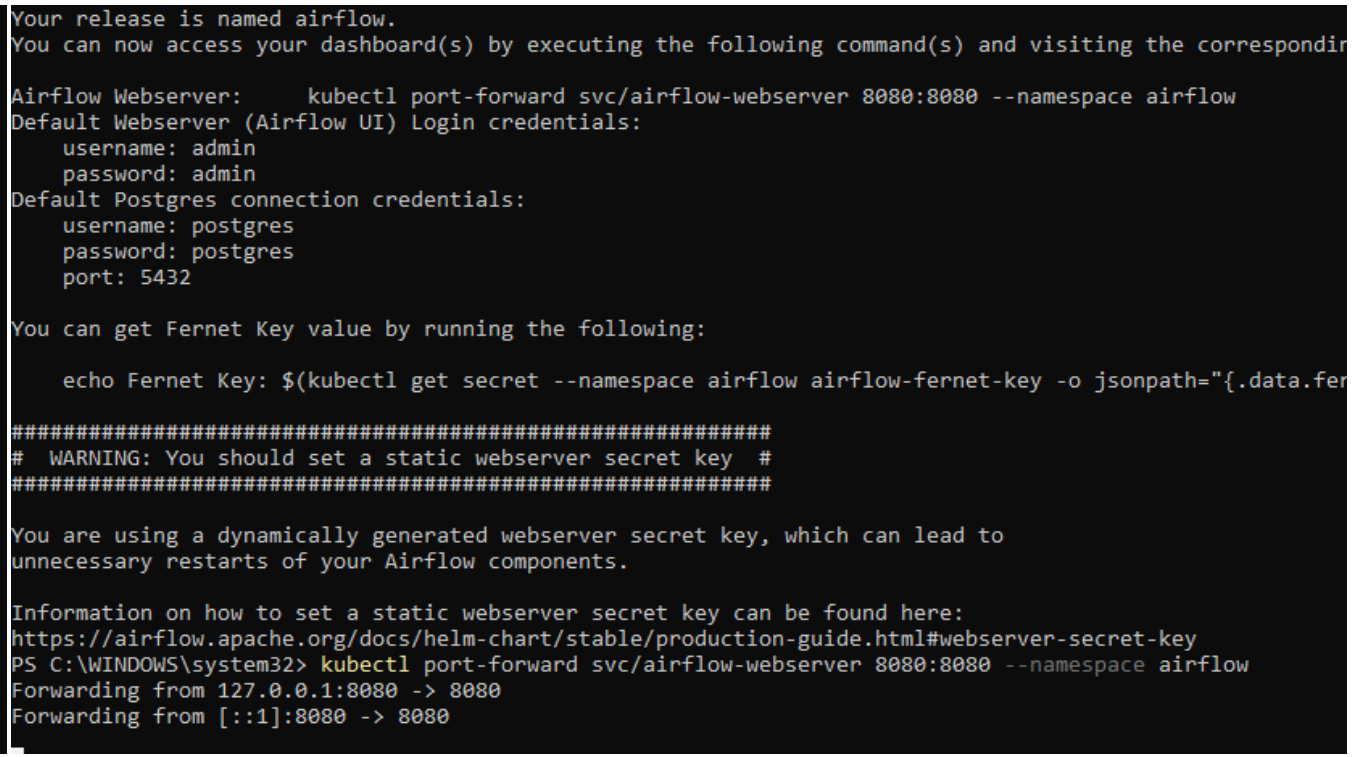
```
PS C:\WINDOWS\system32> helm repo add apache-airflow https://airflow.apache.org
"apache-airflow" has been added to your repositories
PS C:\WINDOWS\system32> helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "apache-airflow" 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\jrver\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

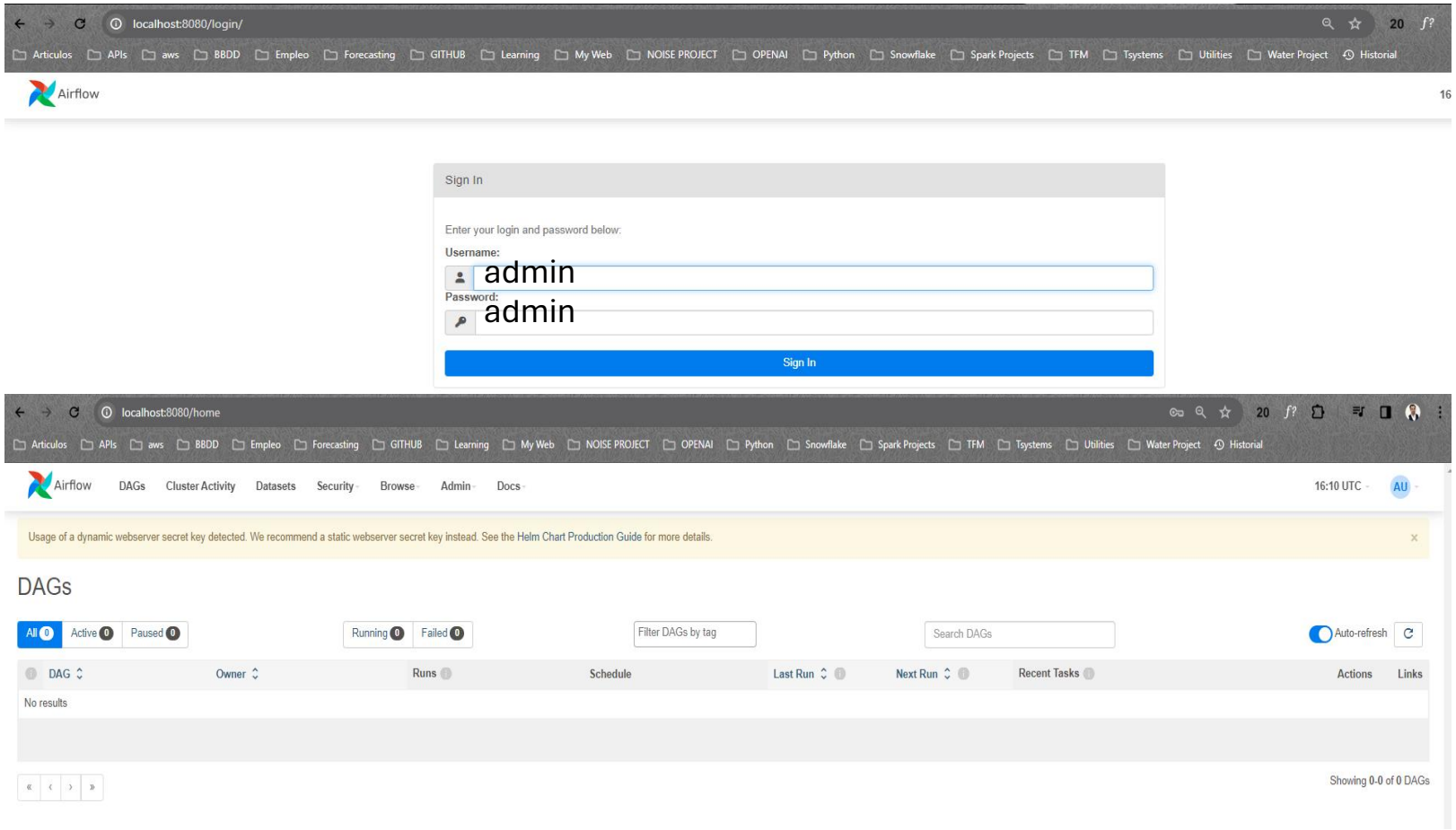




4. DEPLOY AIRFLOW PODS

- 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

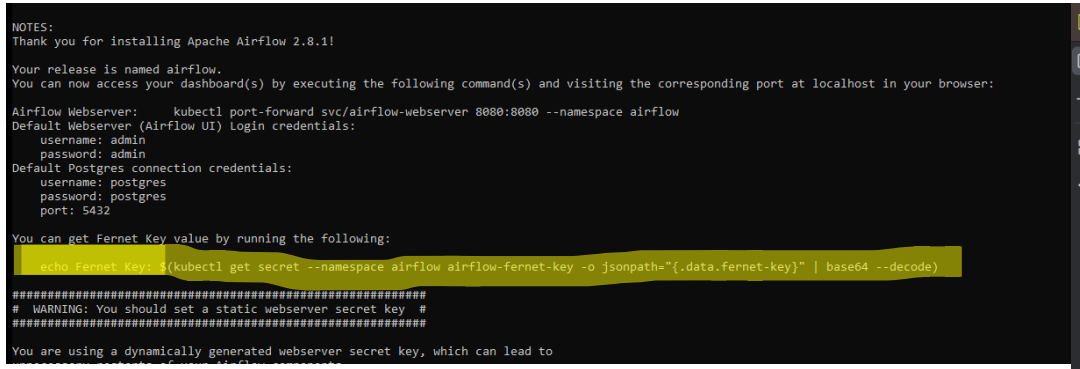


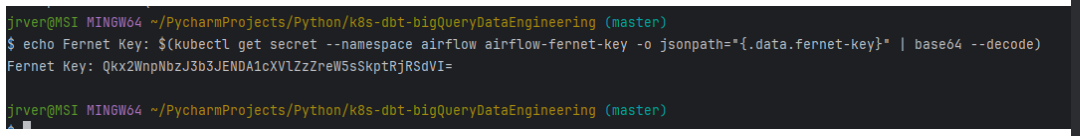
4. DEPLOY AIRFLOW PODS

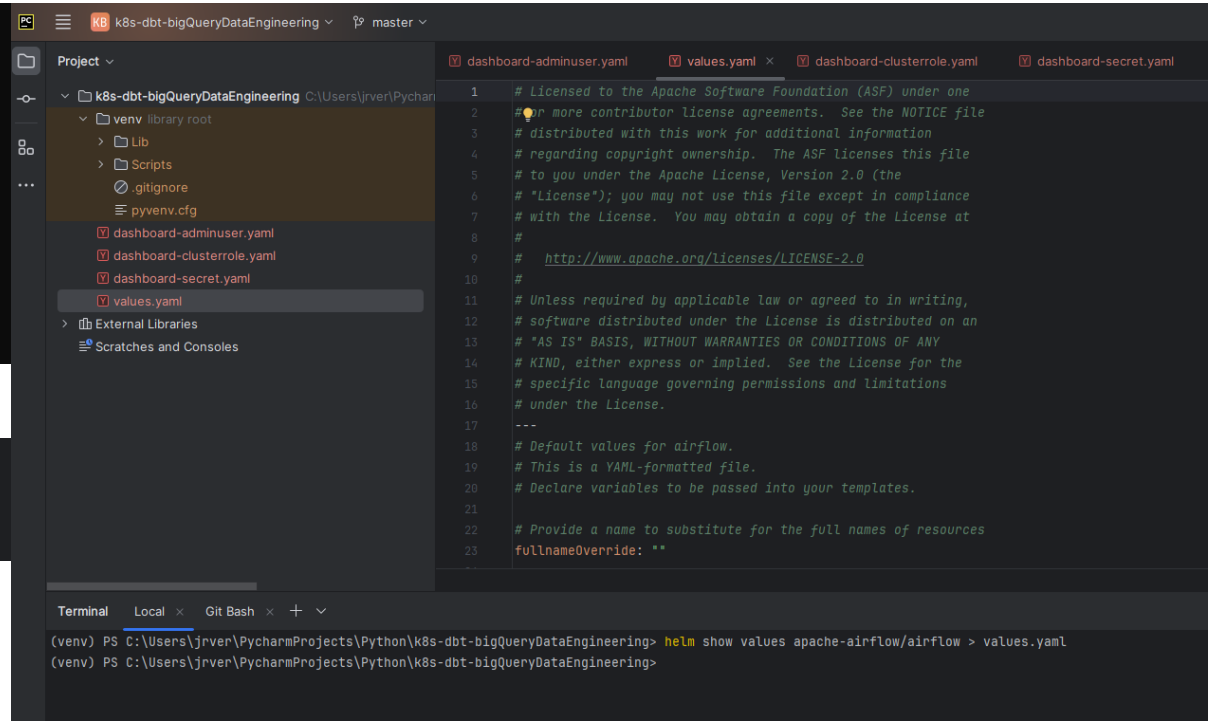
- Values.yaml is created and show default information which is going to be overwritten.
  - 1. the **Fernet Key**, the command is showed once Airflow pod is installed or upgraded.

    > 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

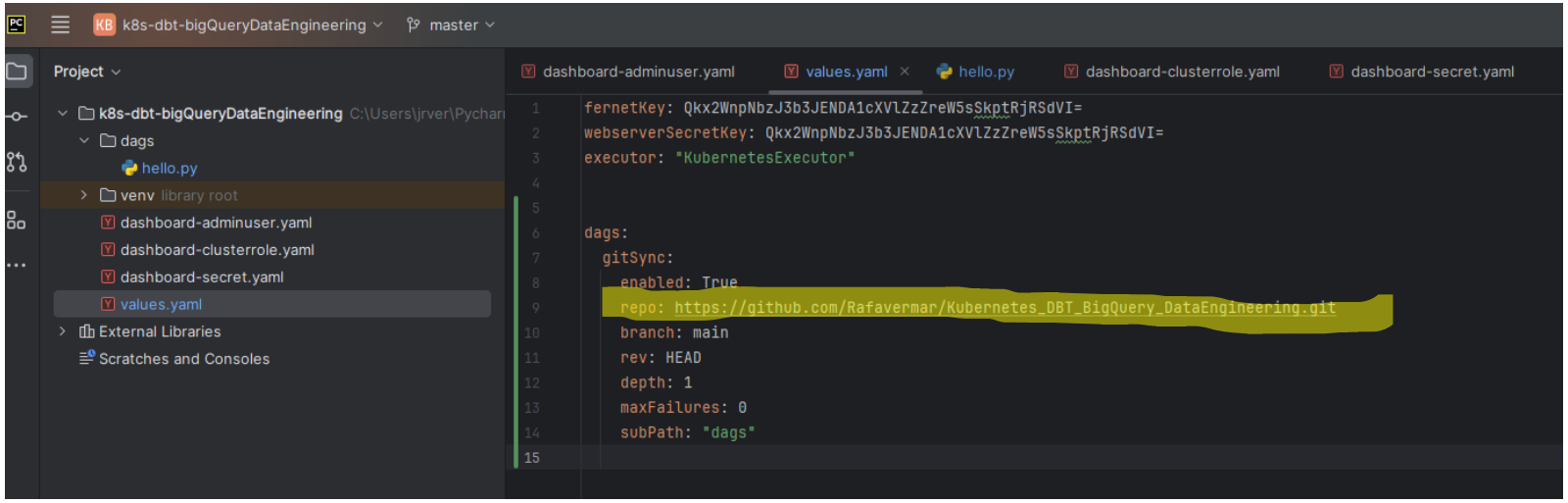






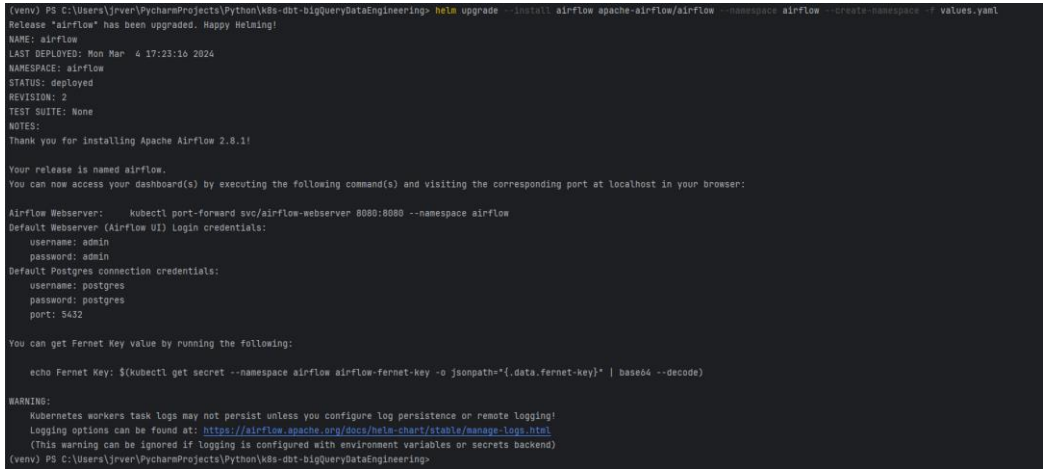
4. DEPLOY AIRFLOW PODS

- 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.



- Upgrade Airflow pod:

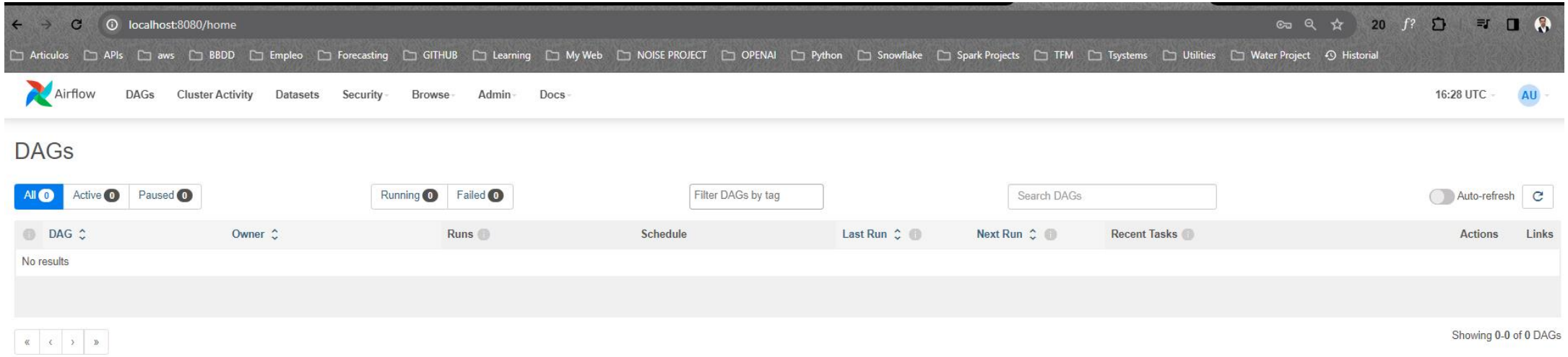
> helm upgrade --install airflow apache-airflow/airflow --namespace airflow --create-namespace -f values.yaml



4. DEPLOY AIRFLOW PODS

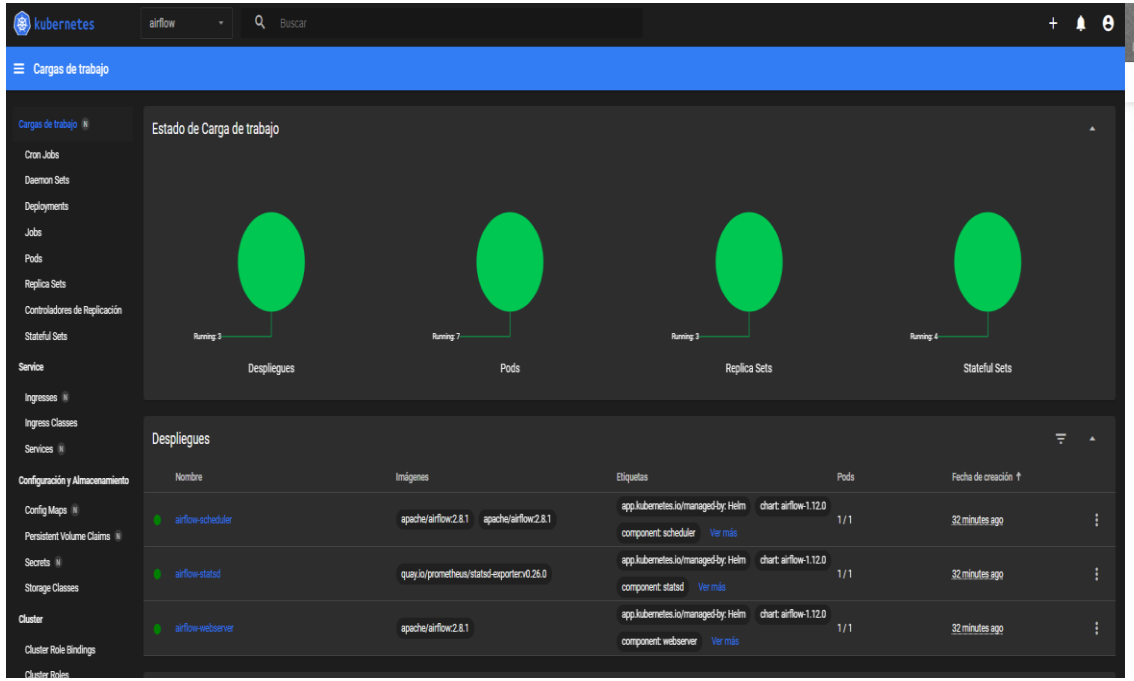
- Forward ports again
- See Airflow UI without any message

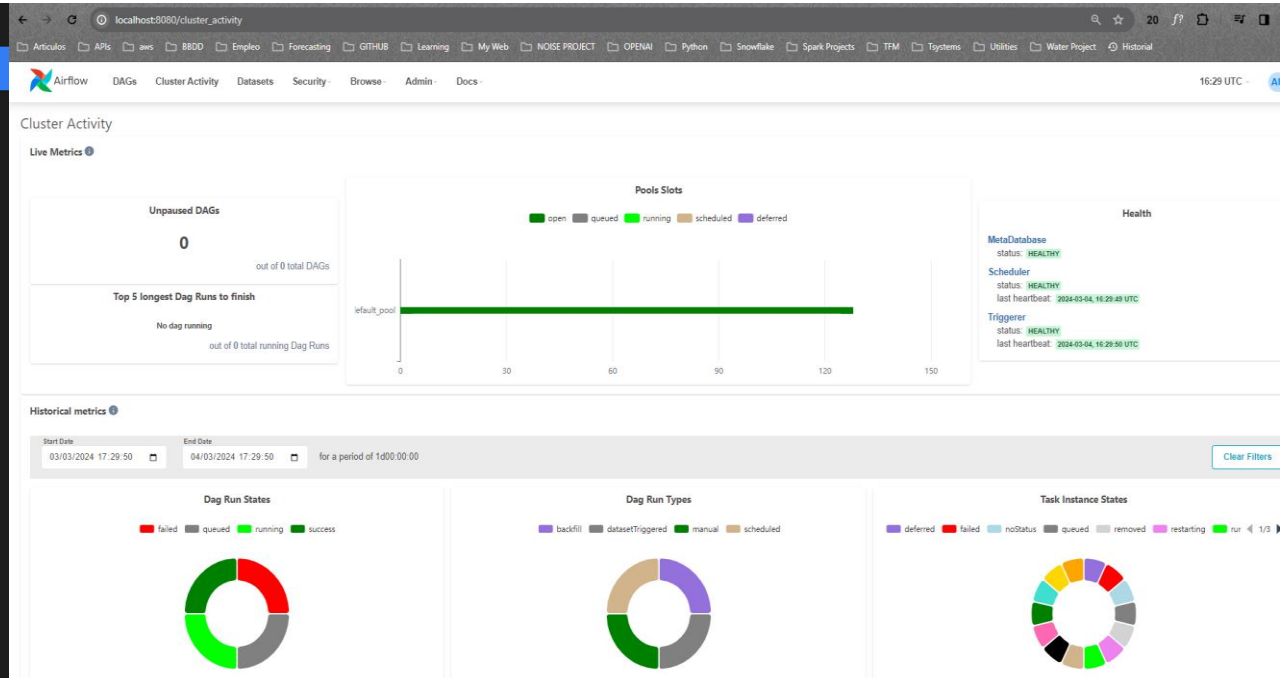
```
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
E0304 17:25:56.876630   25052 portforward.go:409] an error occurred forwarding 8080 -> 8080: error forwarding
ce6b8f703d2438b07af8a32fc9e8b97fa851c4a54b9256d96)
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
```



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.





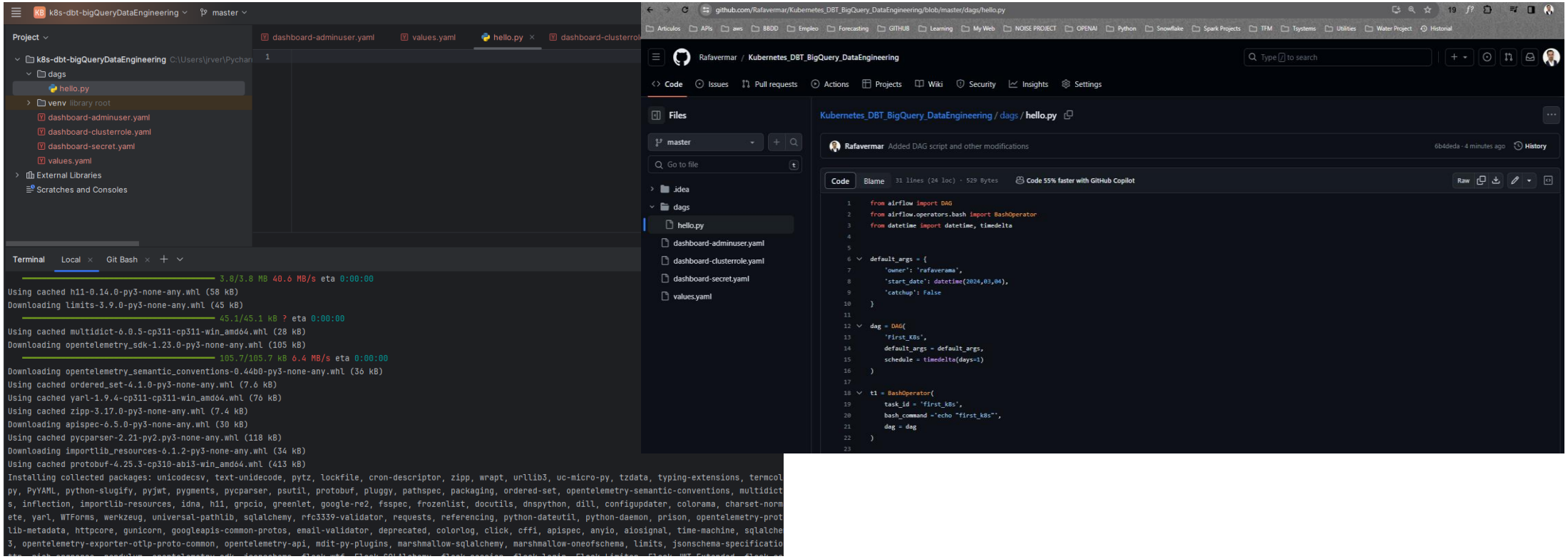
4. DEPLOY AIRFLOW PODS

- 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:

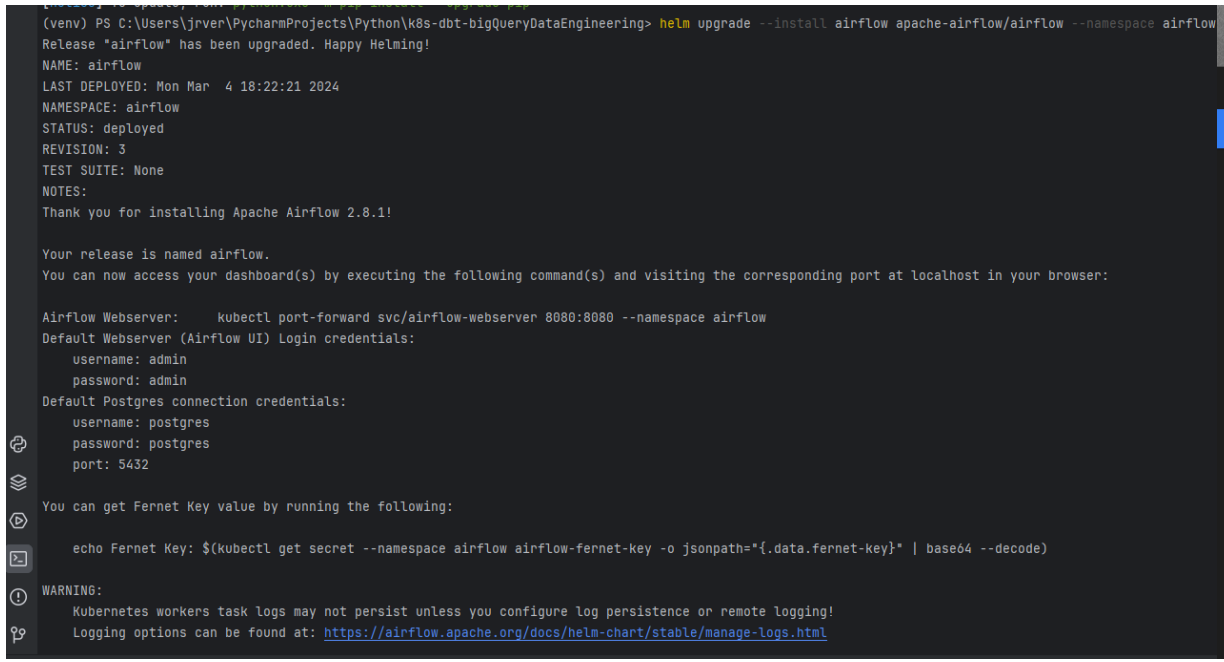
```
>pip install apache-airflow-providers-cncf-kubernetes
```

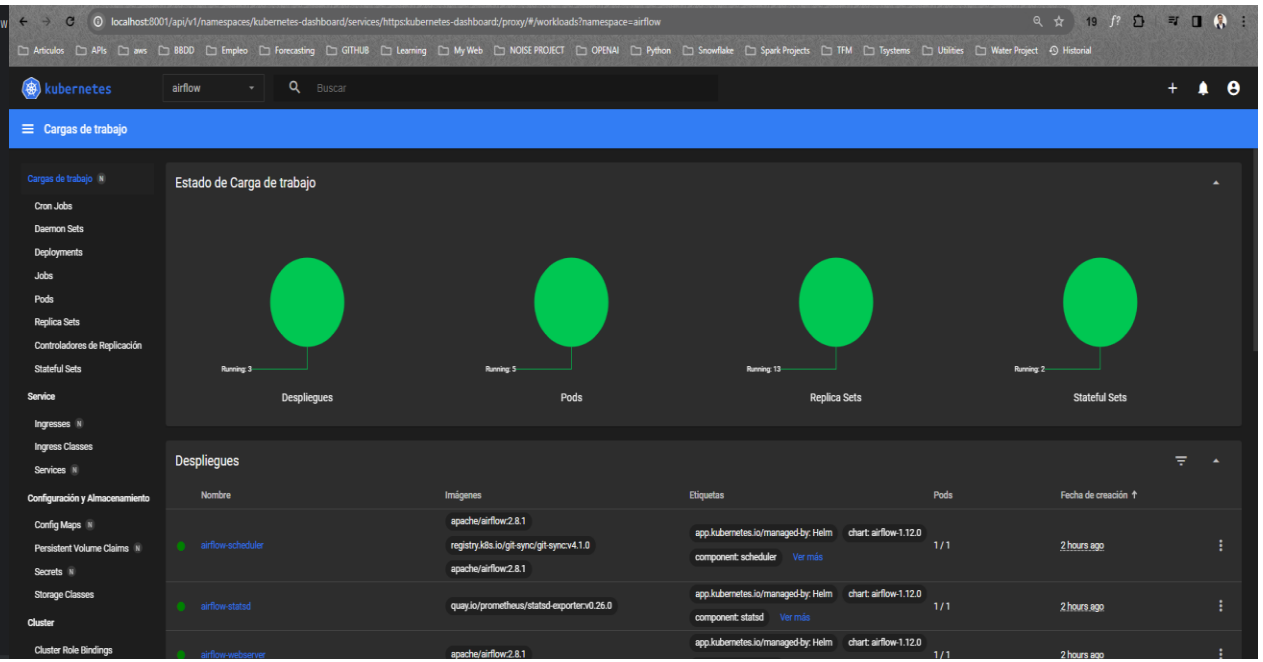


4. DEPLOY AIRFLOW PODS

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

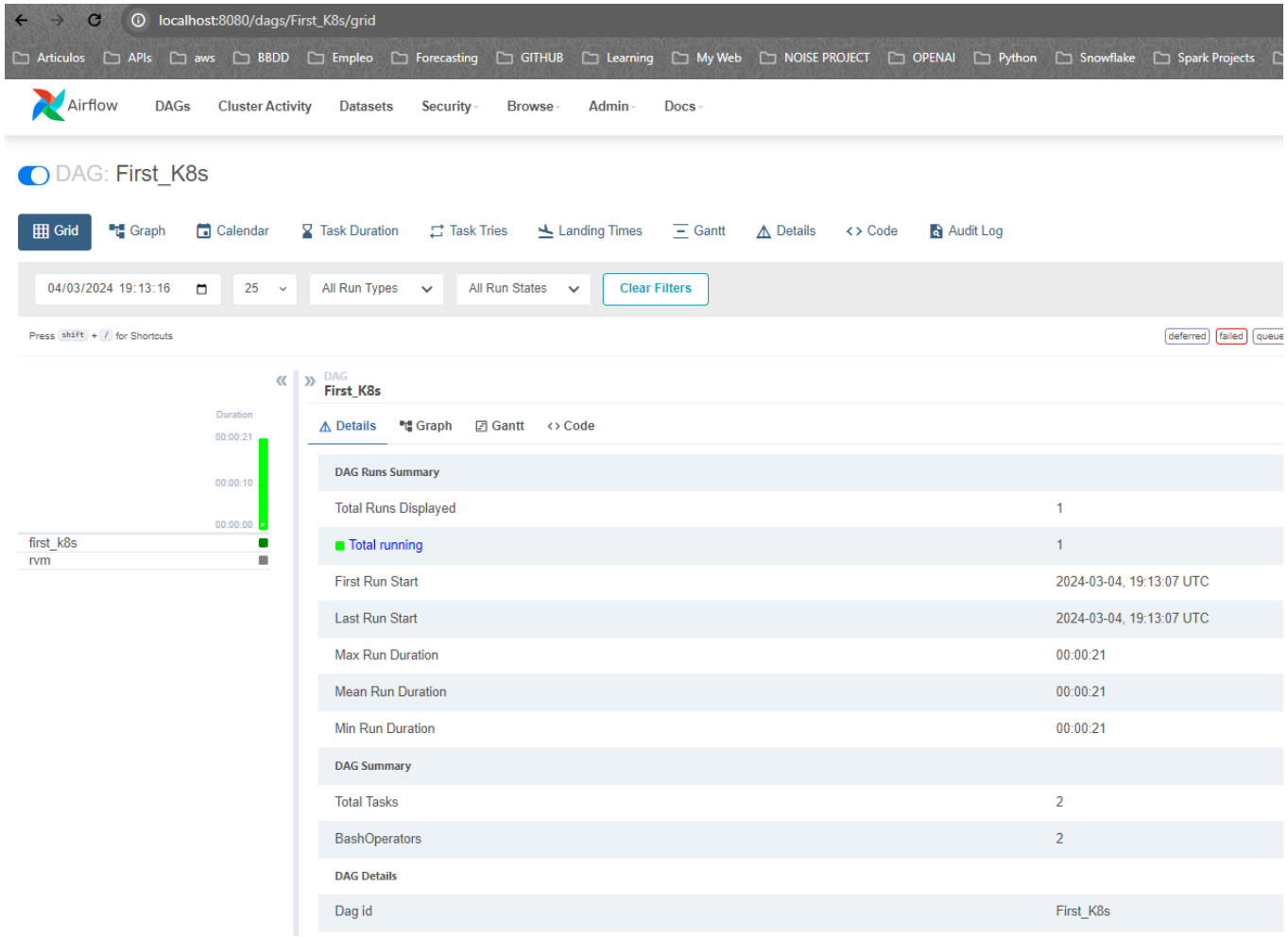
- Check the Kubernetes Dashboard if all is up and running as expected.





4. DEPLOY AIRFLOW PODS

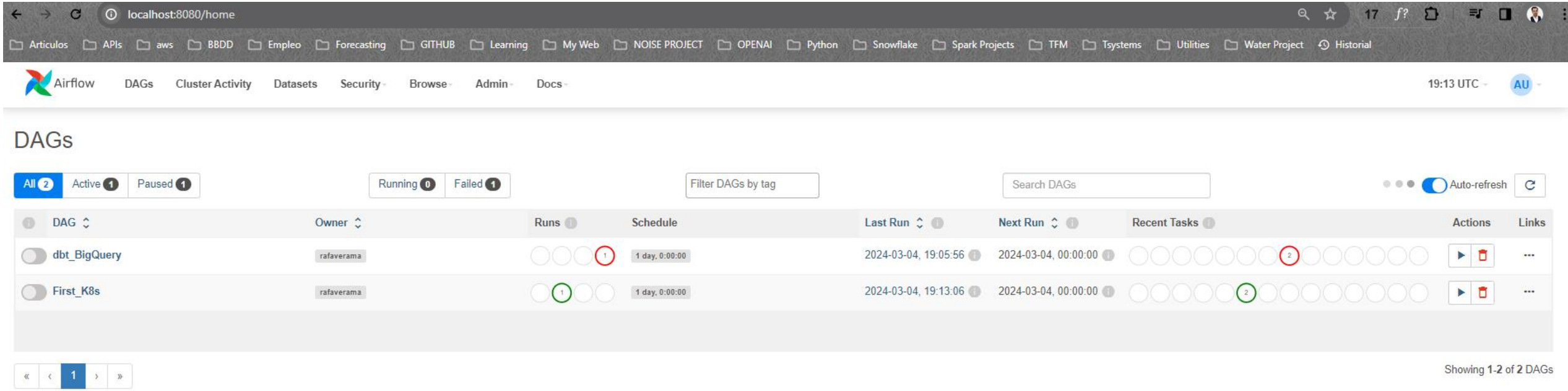
- Run the DAG “dbt_TEST” (First_K8s) successfully



4. DEPLOY AIRFLOW PODS

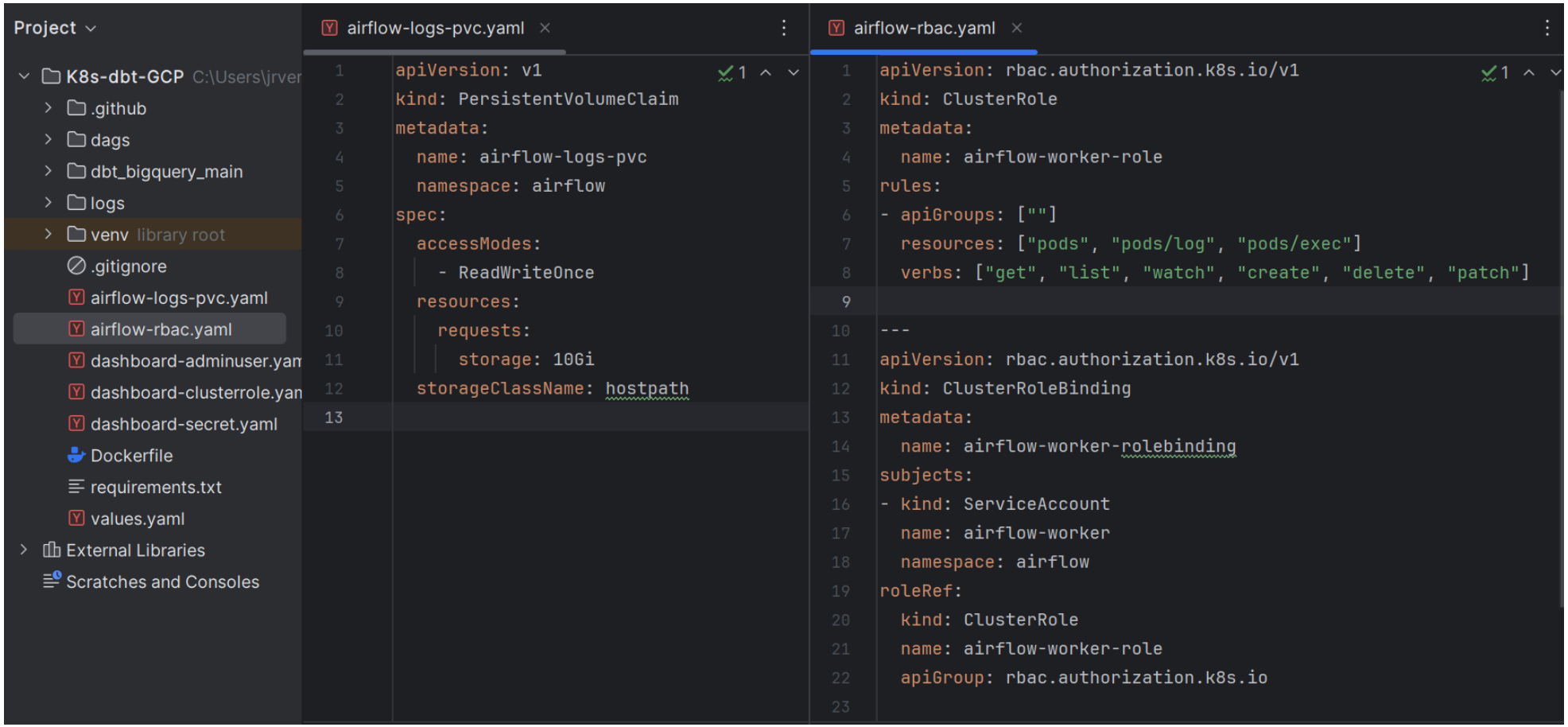
- While running the second DAG I experienced some errors but I was not able to see the logs directly into the Airflow UI ---please do not care about the dag's names here. Just see the repo. I made name changes on the fly

- 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.



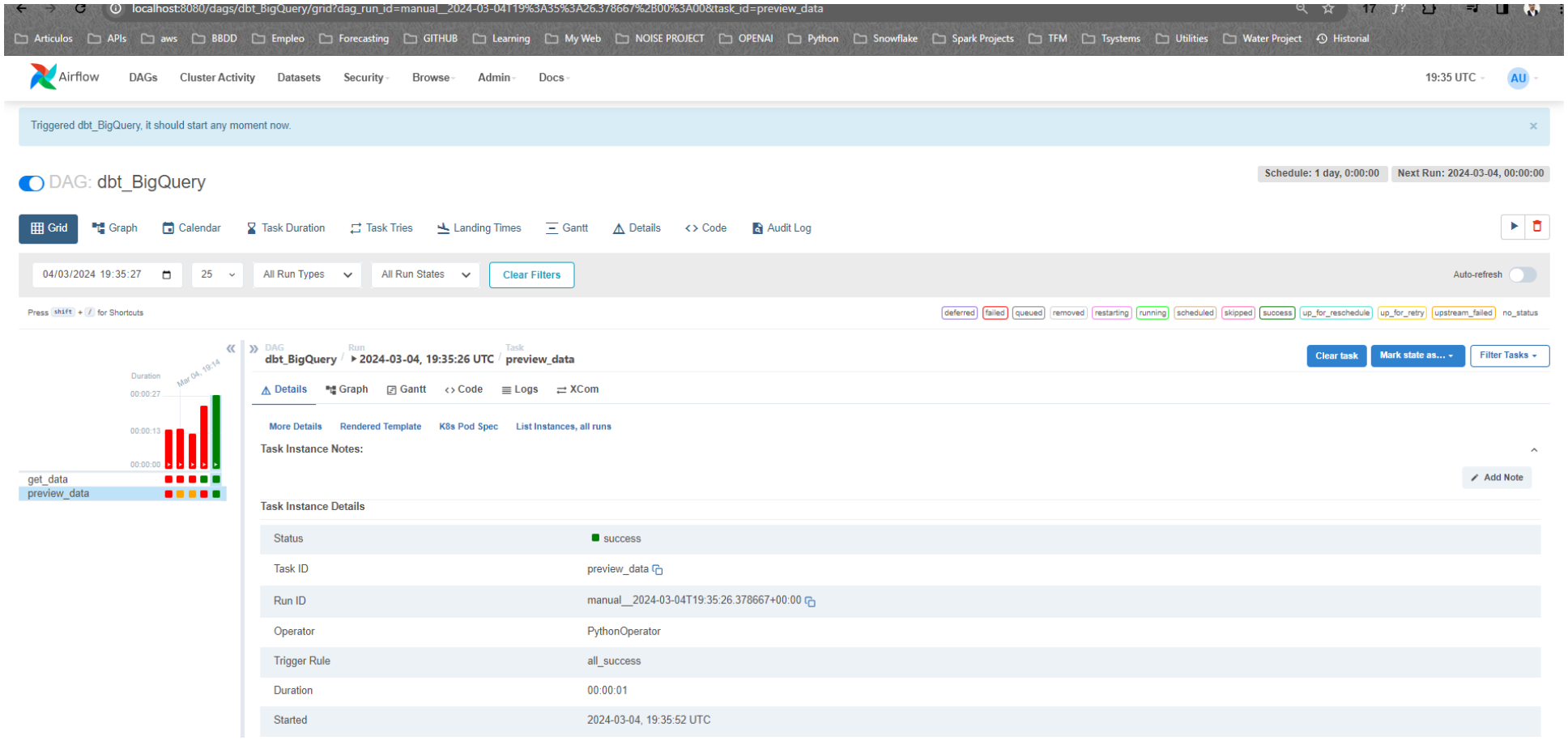
5. ESTABLISH LOG PERSISTENCE FOR AIRFLOW

- 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



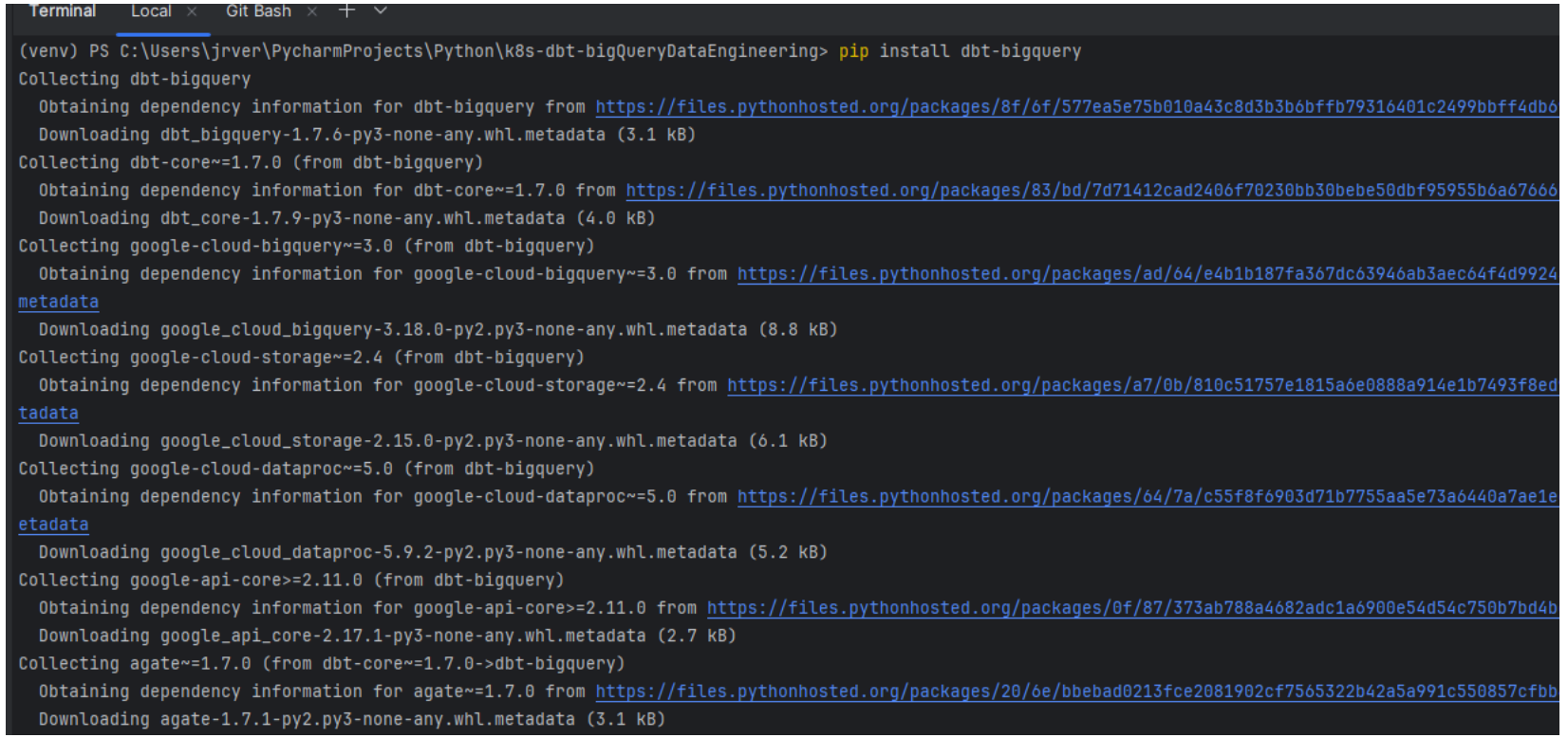
5. STABLISH LOG PERSISTENCE FOR AIRFLOW

- 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



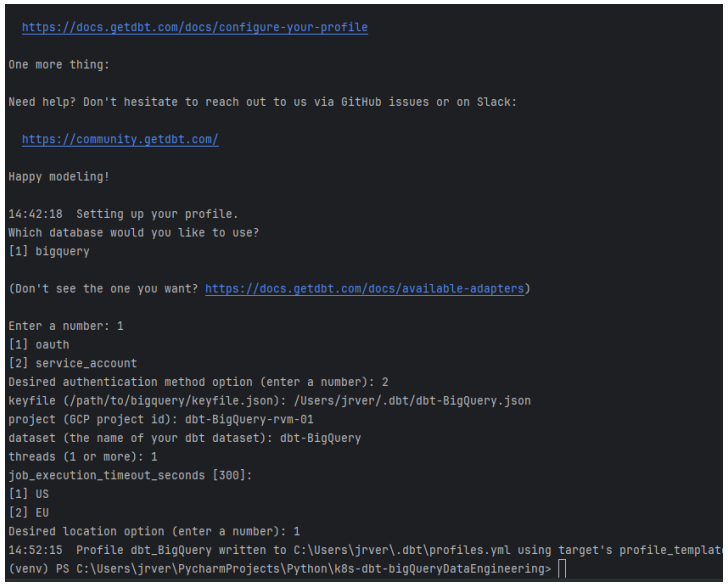
6. INSTALL DBT-BIGQUERY

- 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



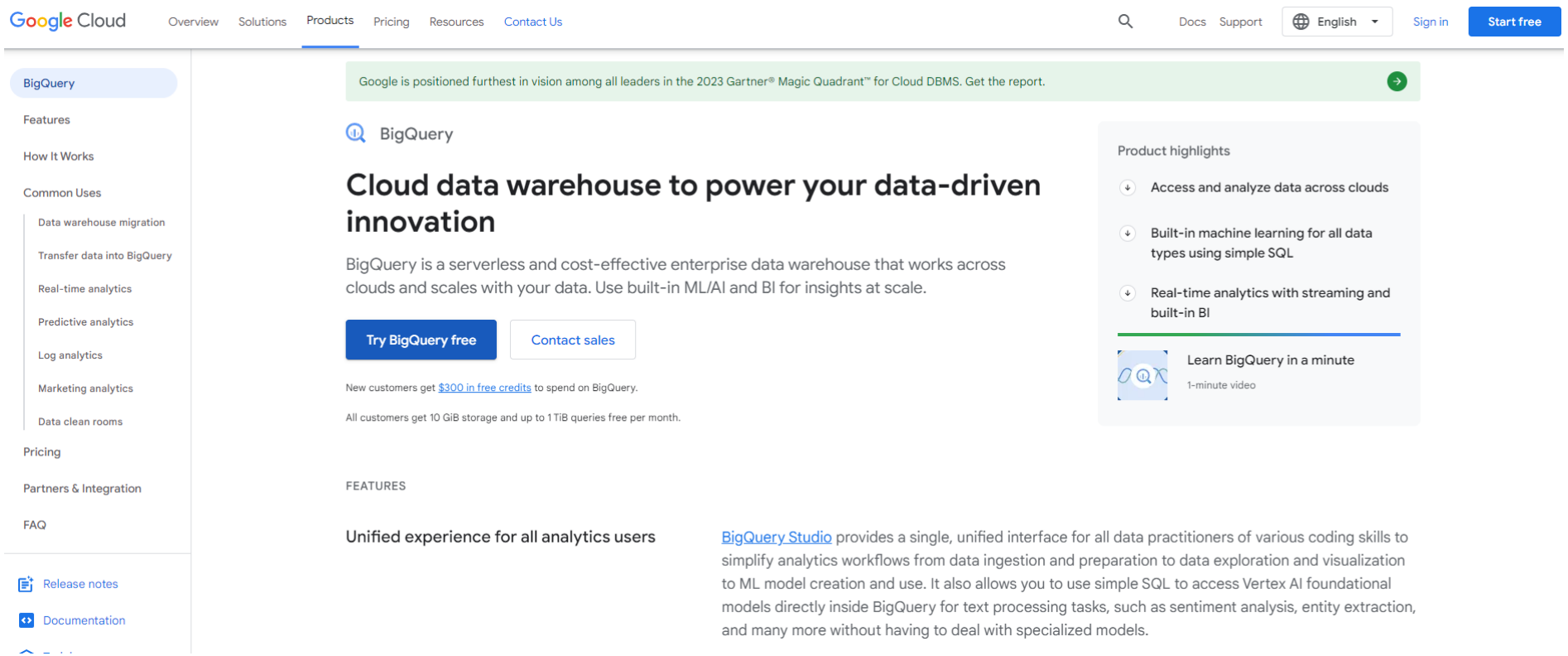
6. INSTALL DBT-BIGQUERY

- 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]



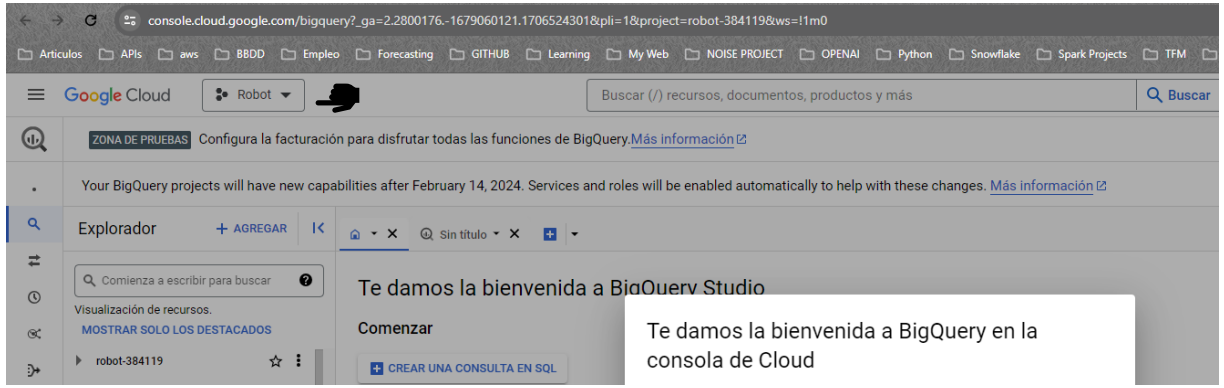
7. SETUP BIGQUERY-GCP

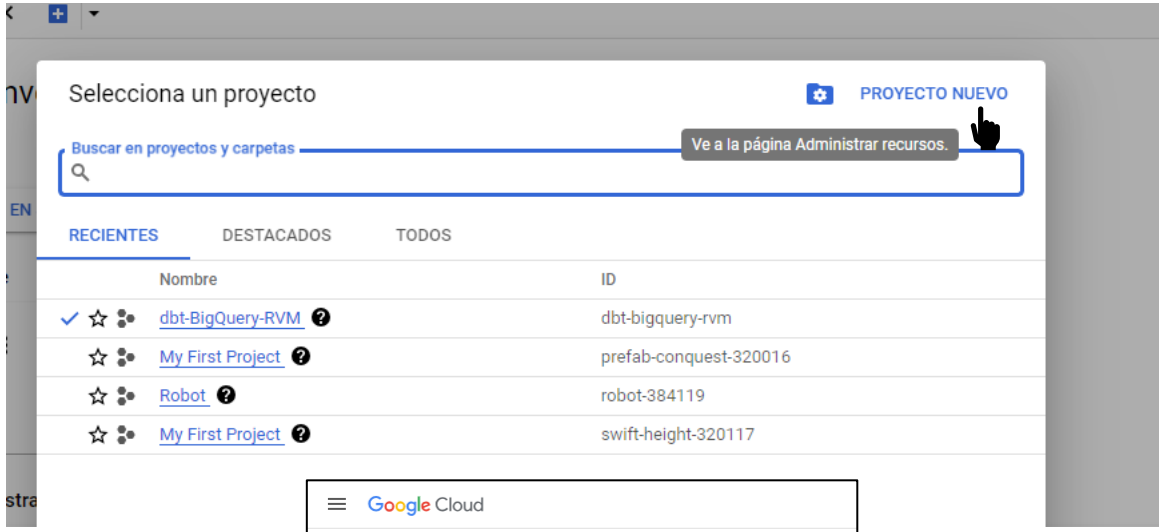
- Go to GCP-BigQuery : https://cloud.google.com/bigquery?hl=en
- Sign in and **Try BigQuery free**

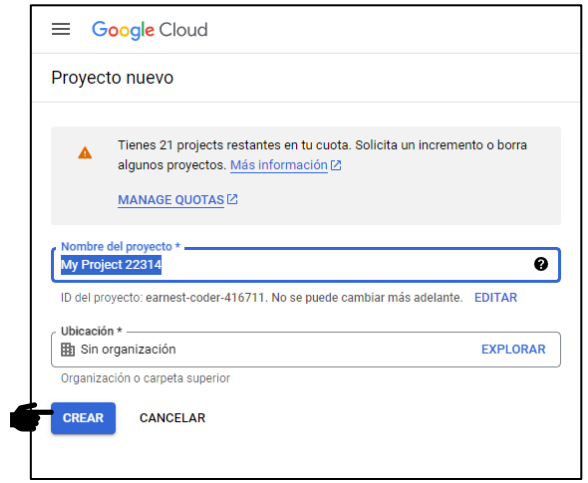


7. SETUP BIGQUERY-GCP

- 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

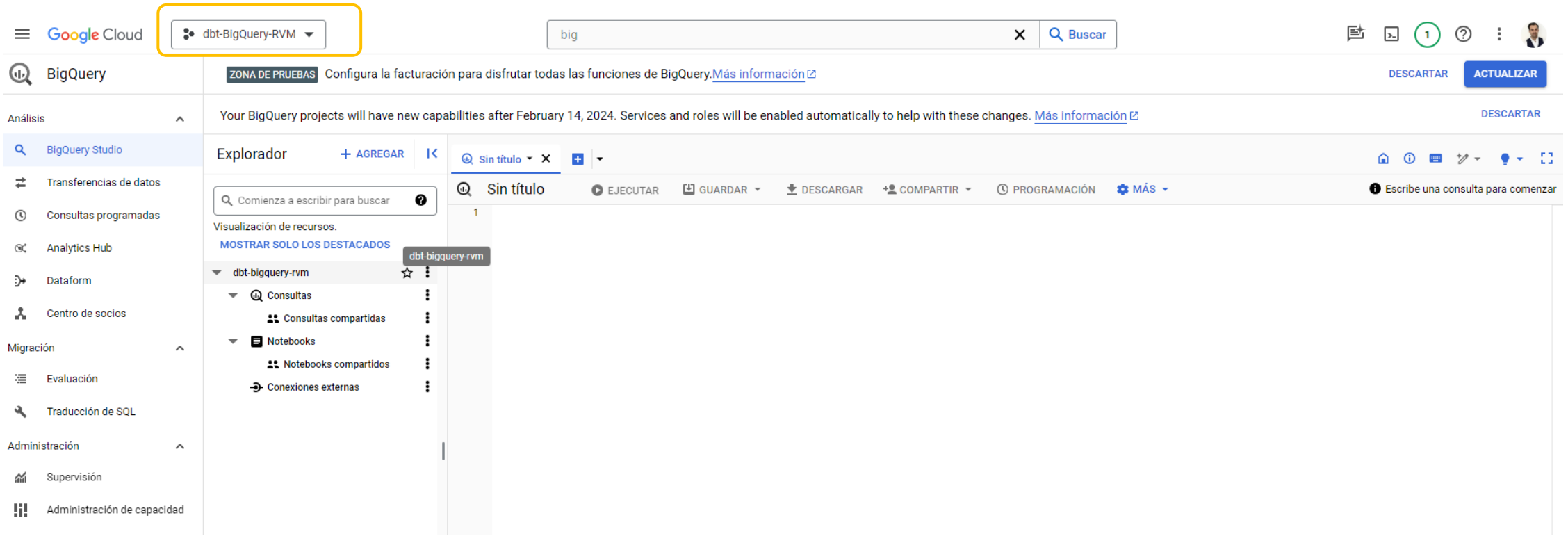






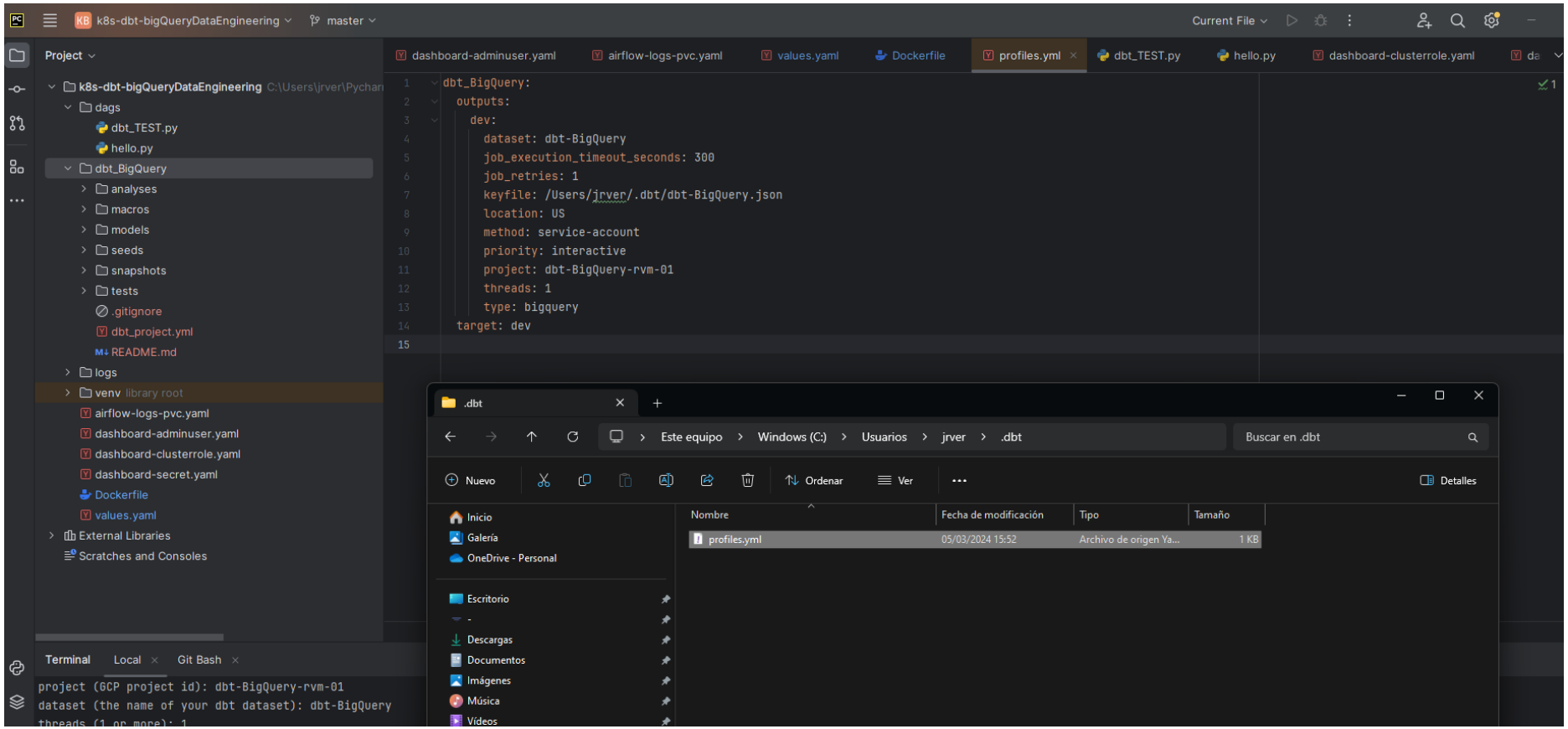
7. SETUP BIGQUERY-GCP

- 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.



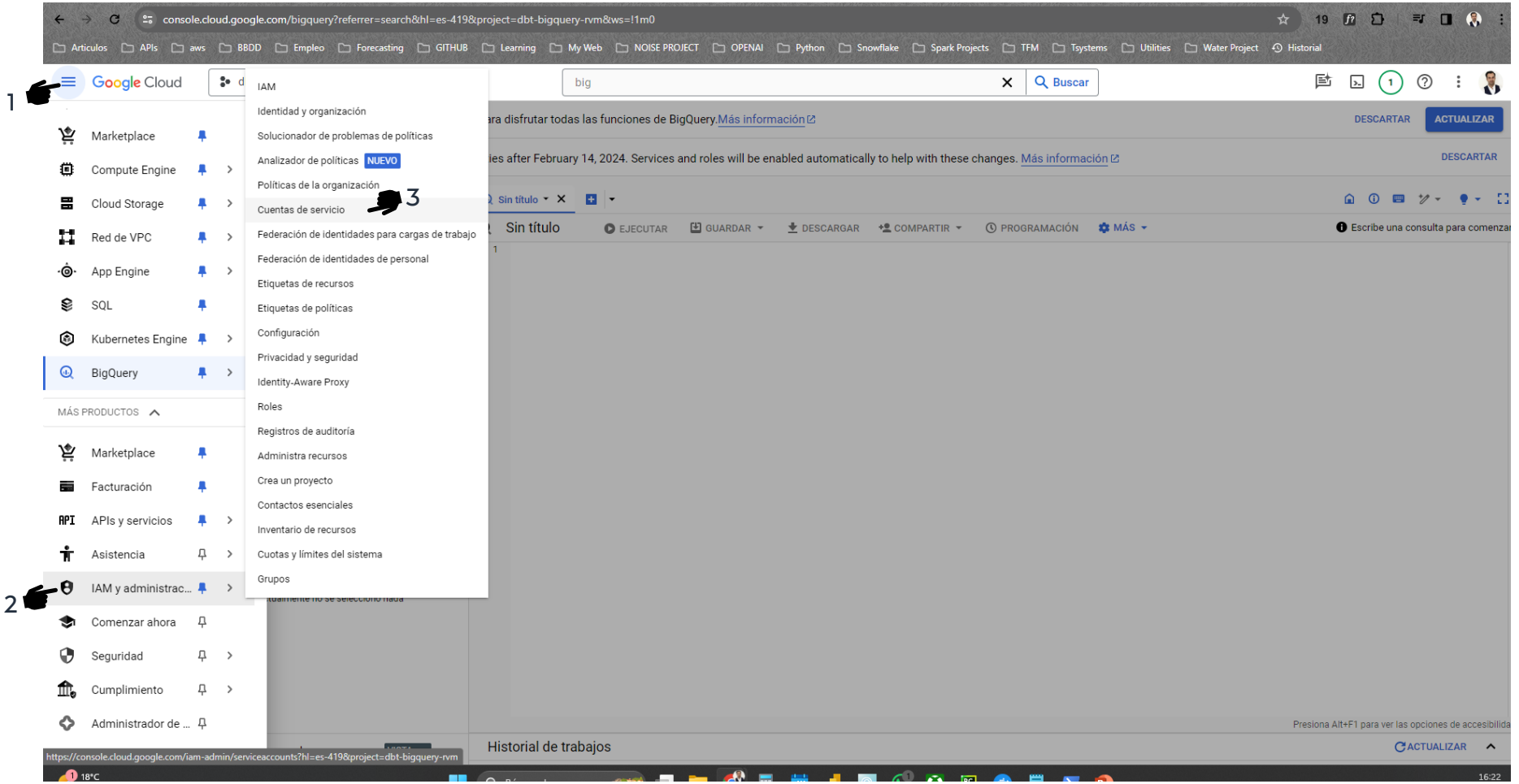
7. SETUP BIGQUERY-GCP

- 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.



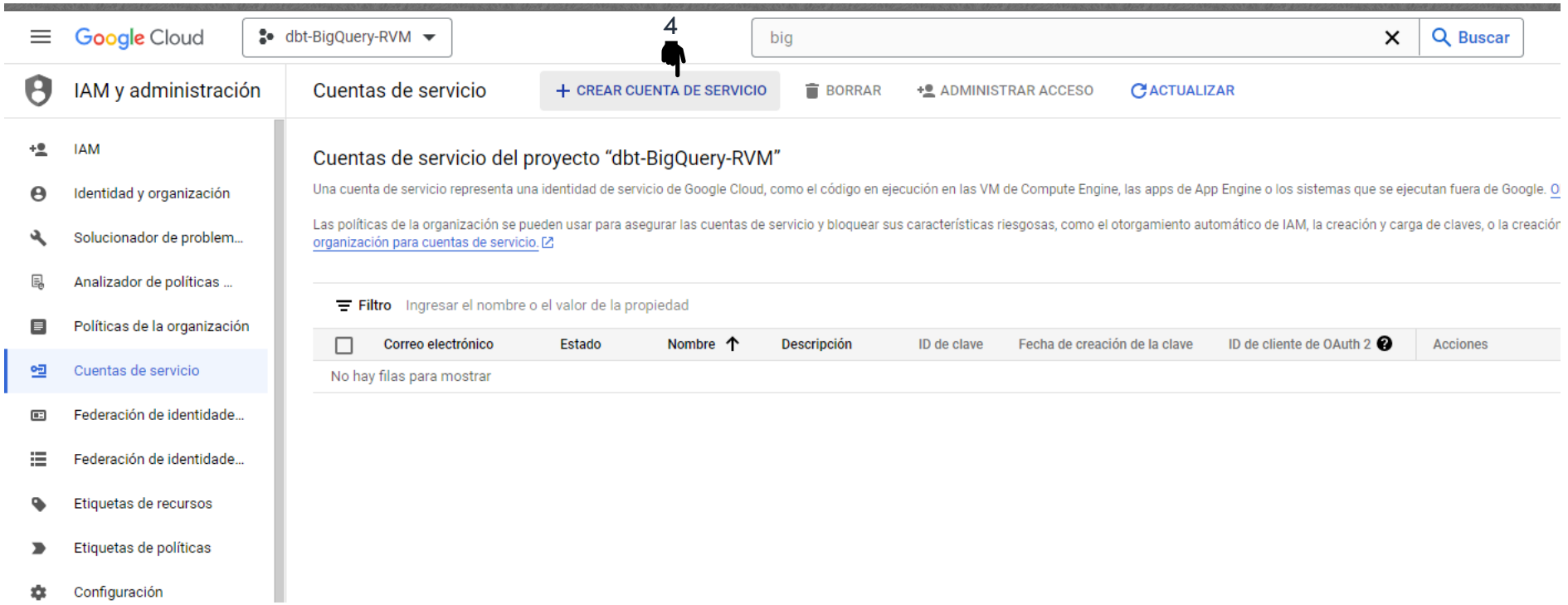
7. SETUP BIGQUERY-GCP

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



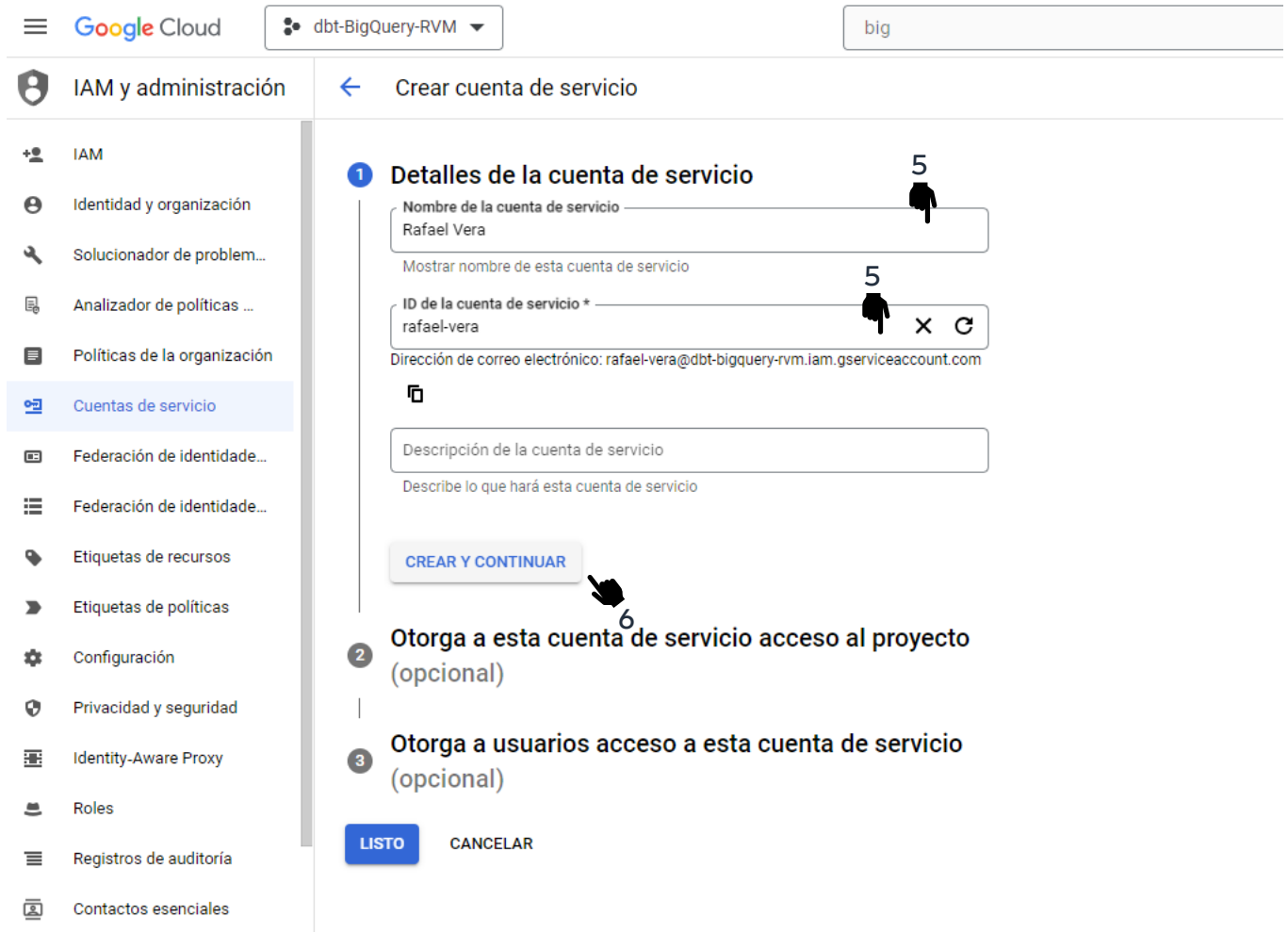
7. SETUP BIGQUERY-GCP

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



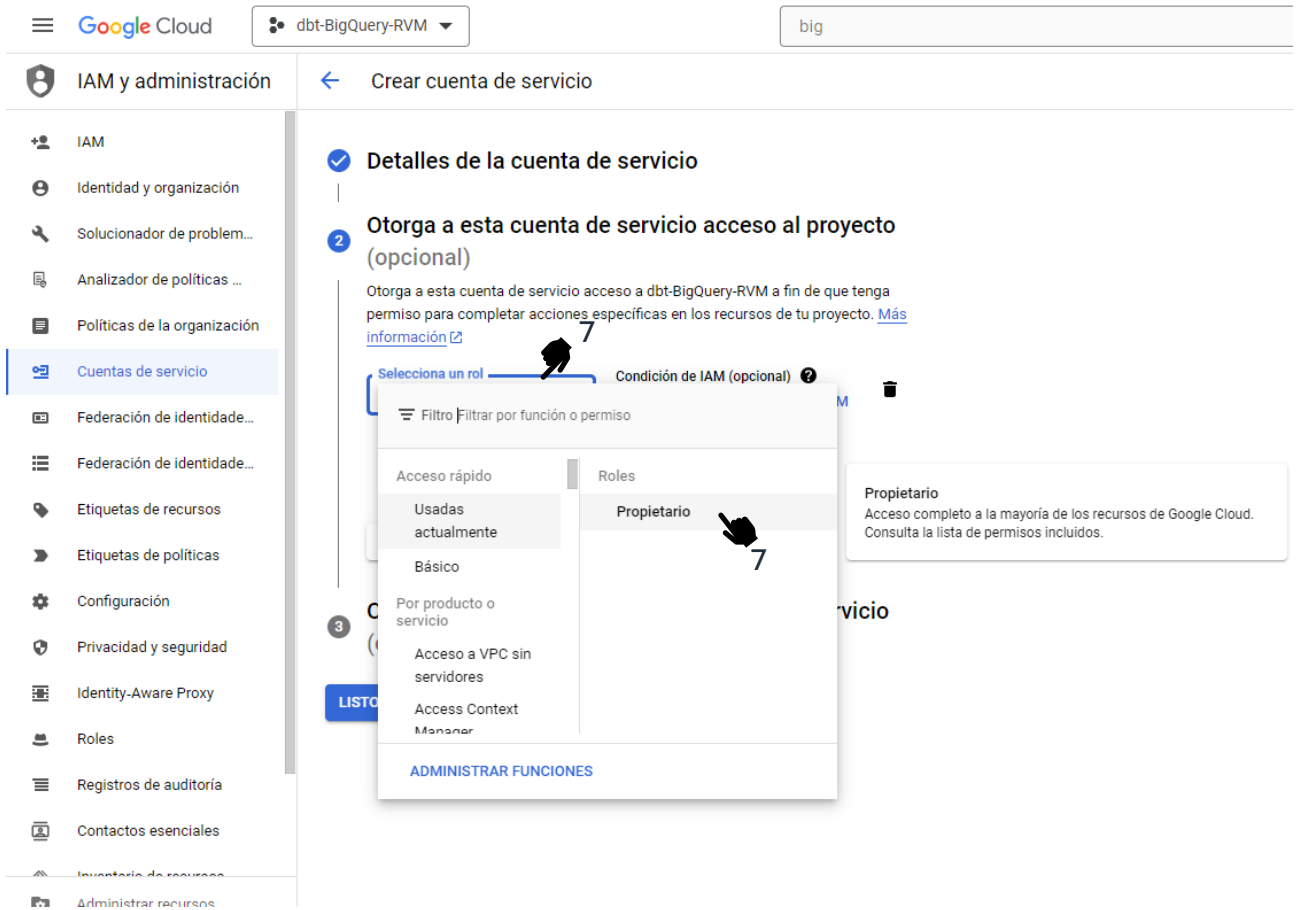
7. SETUP BIGQUERY-GCP

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



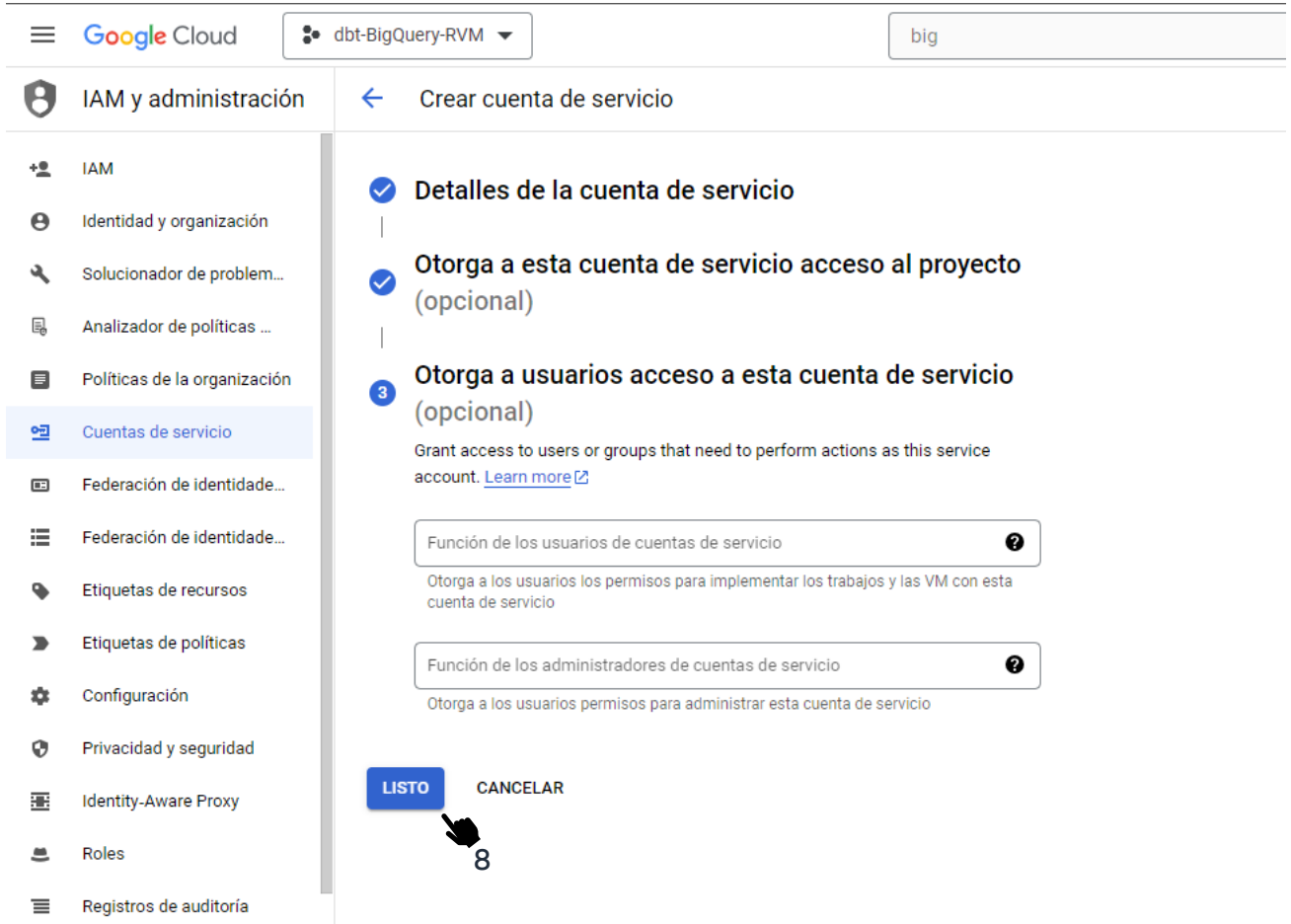
7. SETUP BIGQUERY-GCP

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



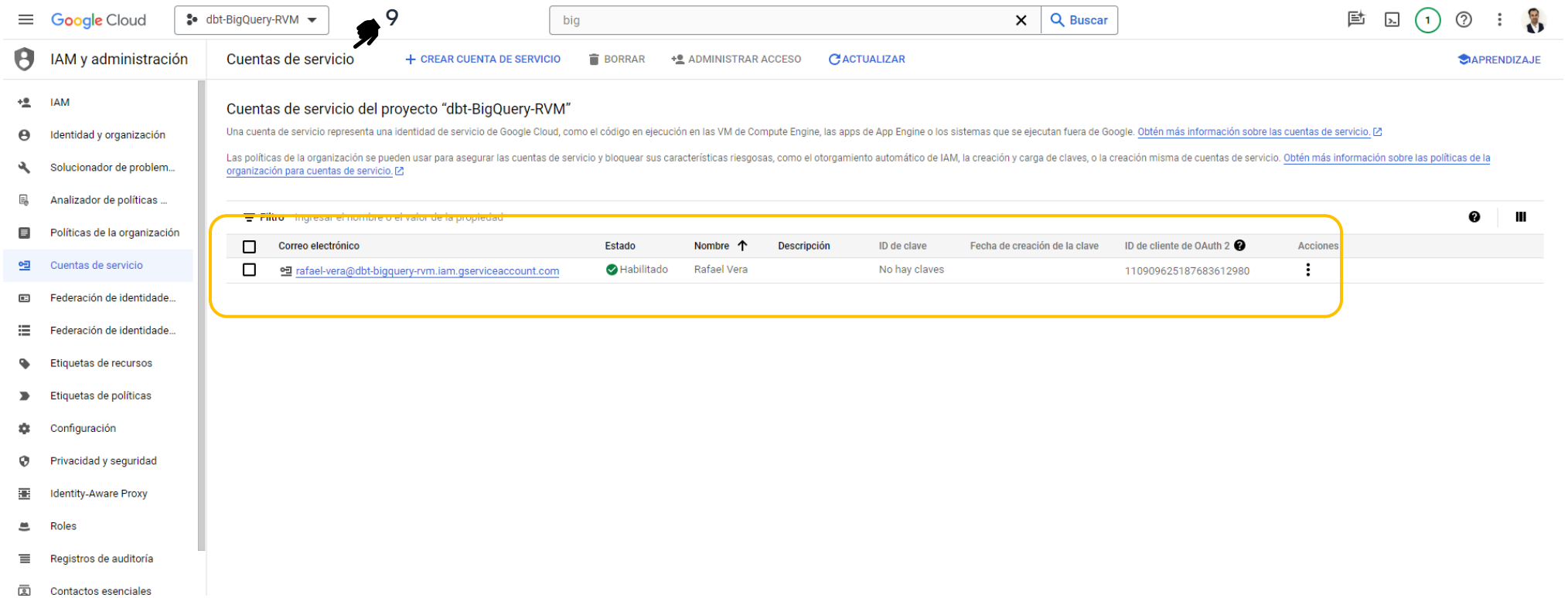
7. SETUP BIGQUERY-GCP

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



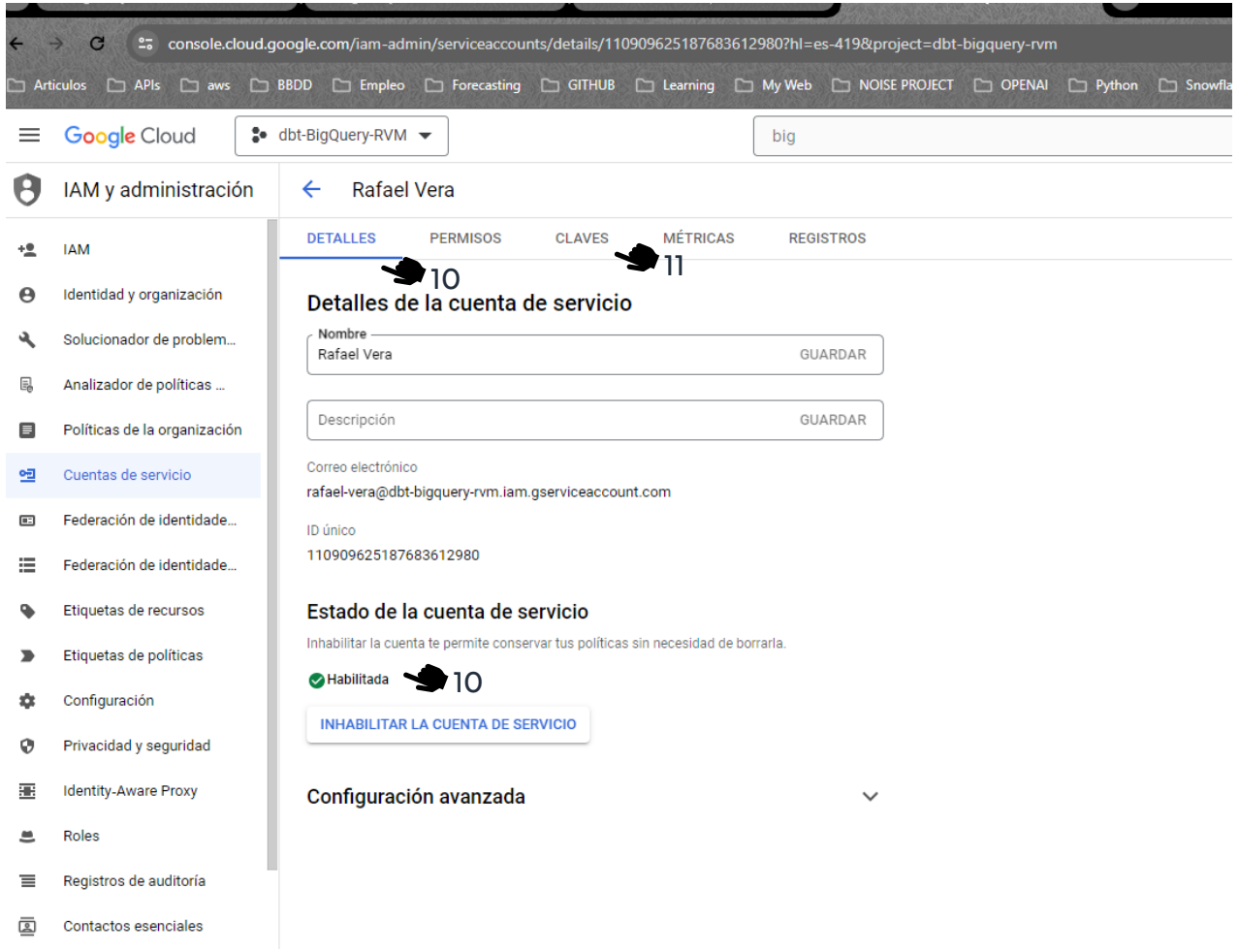
7. SETUP BIGQUERY-GCP

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



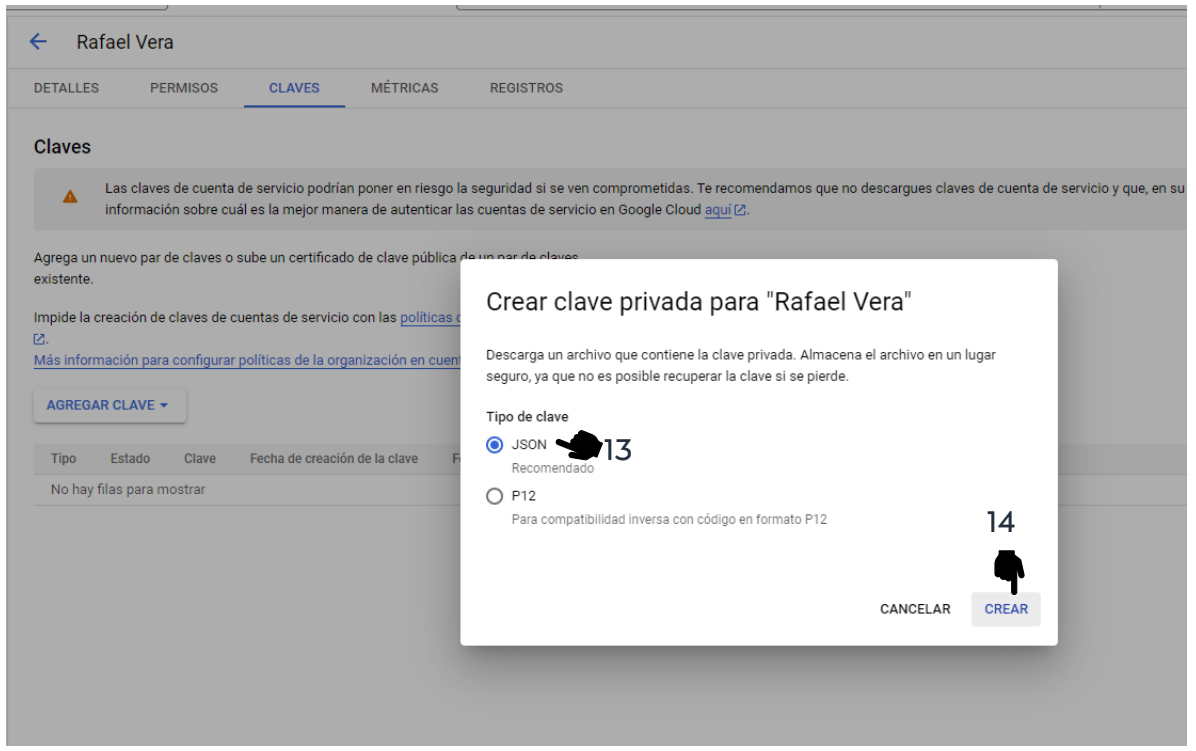
7. SETUP BIGQUERY-GCP

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



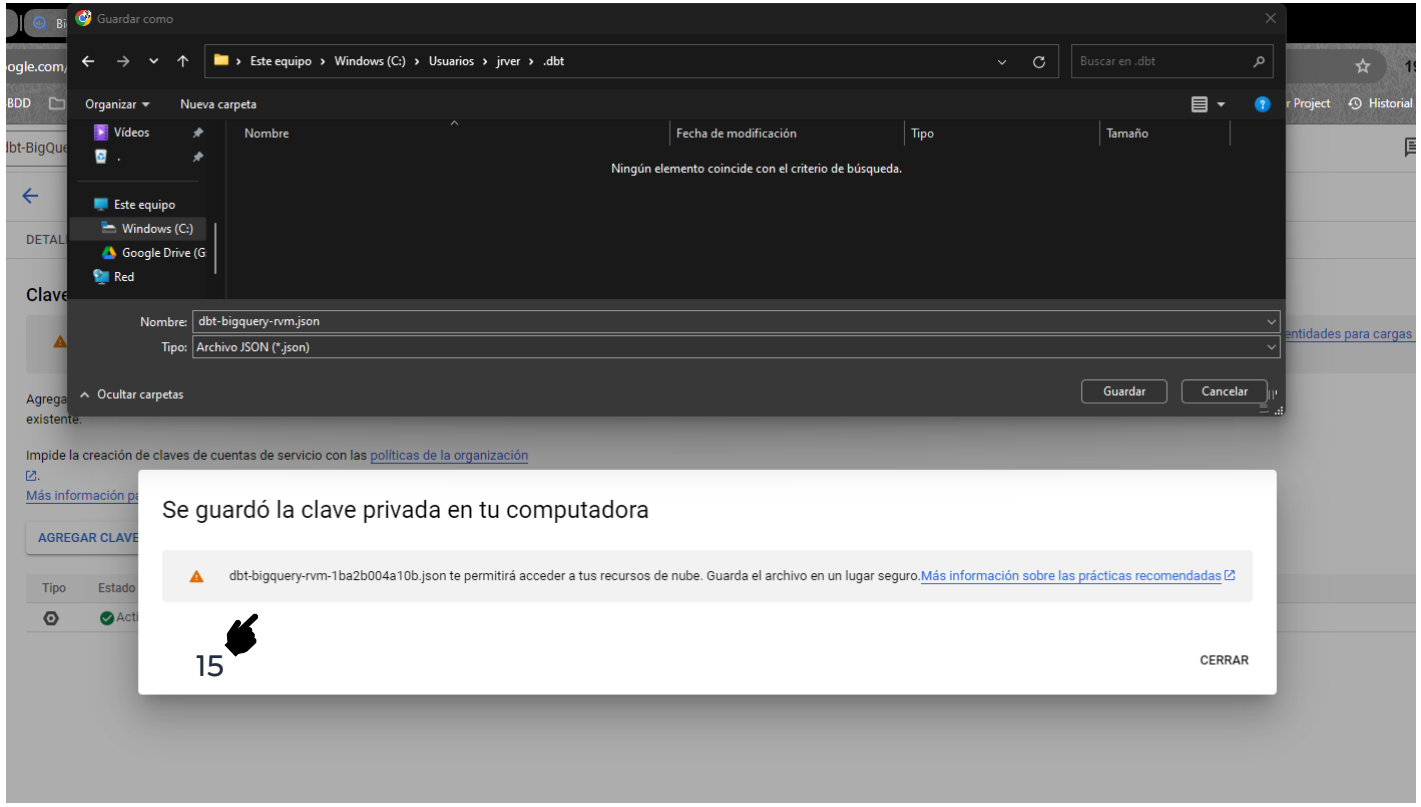
7. SETUP BIGQUERY-GCP

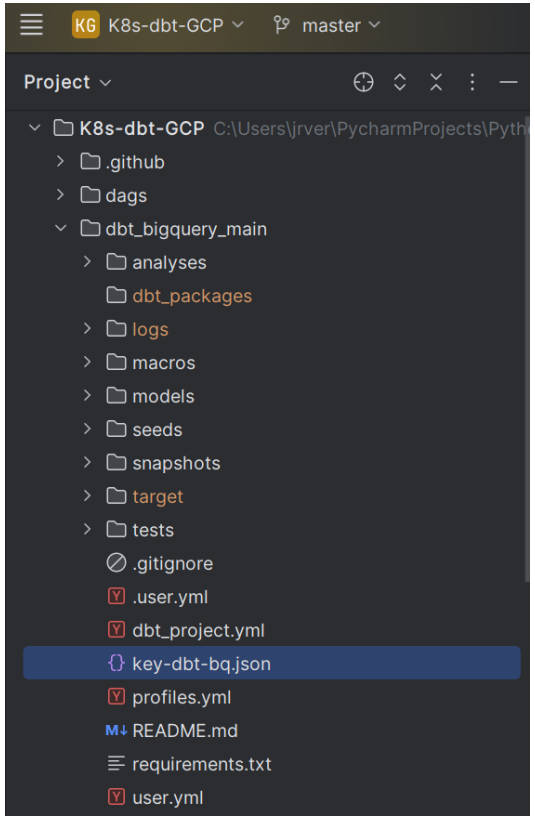
- **Getting keyfile**
- 12. Create a new Key -> pop up shows up
- 13. Add new key -> Type JSON
- 14. Create key -> pops up a window to select the downloading path into your computer



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 enviroment together with profiles.yml

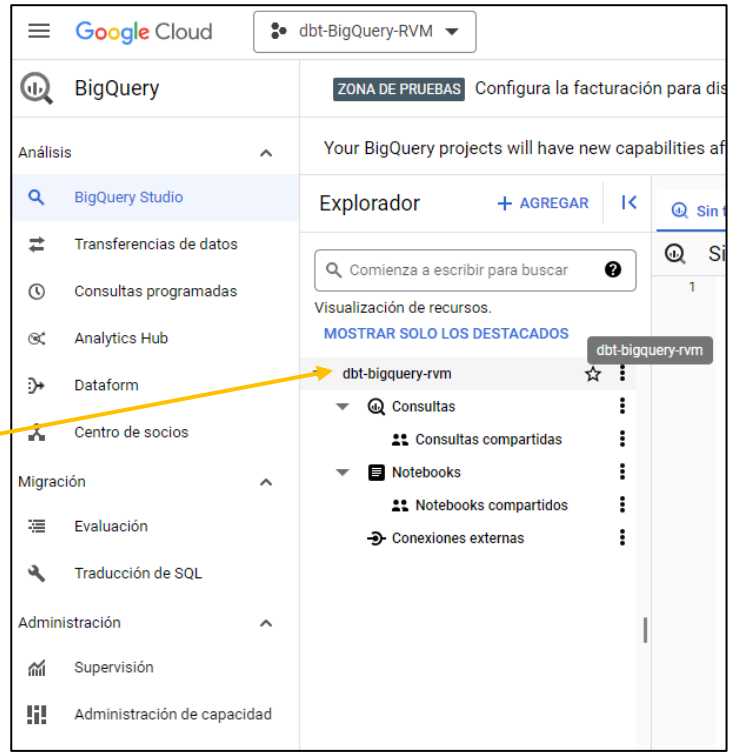


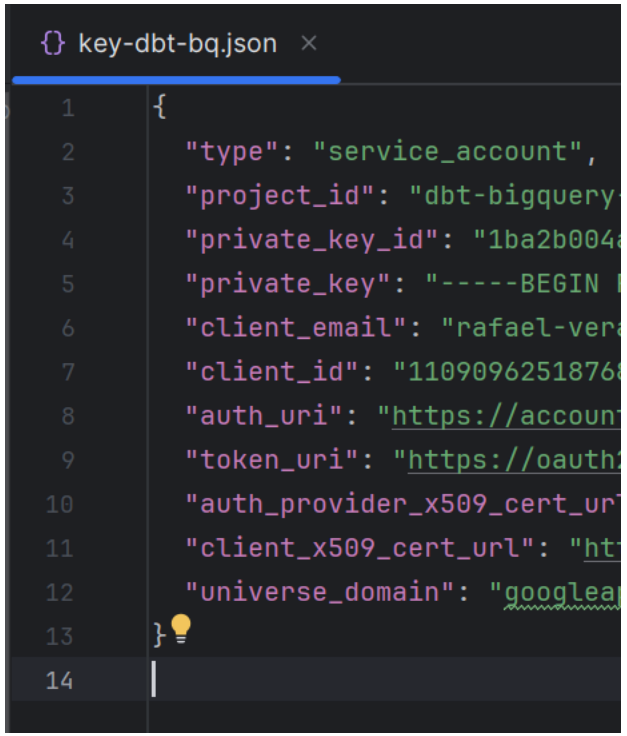


7. SETUP BIGQUERY-GCP

- **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.

```
dbt_BigQuery:
  outputs:
    dev:
      dataset: dbt-BigQuery
      job_execution_timeout_seconds: 300
      job_retries: 1
      keyfile: "/Users/jrver/.dbt/dbt-BigQuery.json"
      location: US
      method: service-account
      priority: interactive
      project: dbt-BigQuery-rvm-01
      threads: 1
      type: bigquery
  target: dev
```



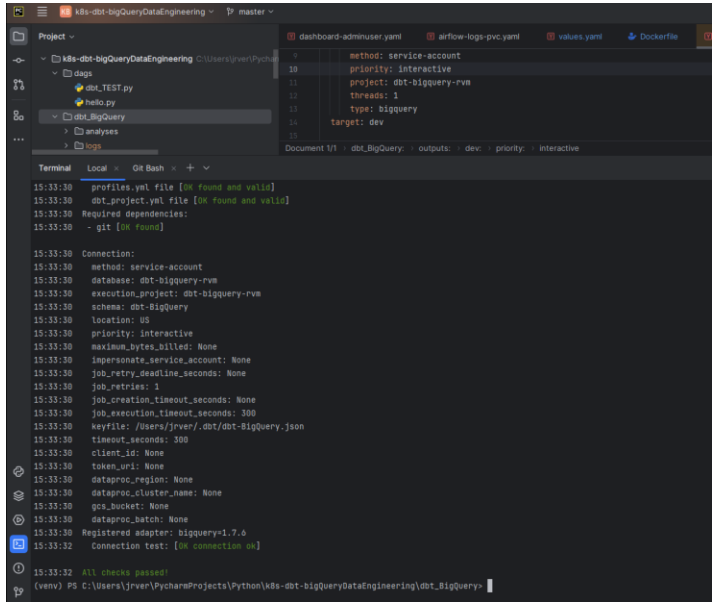


7. SETUP BIGQUERY-GCP

- **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

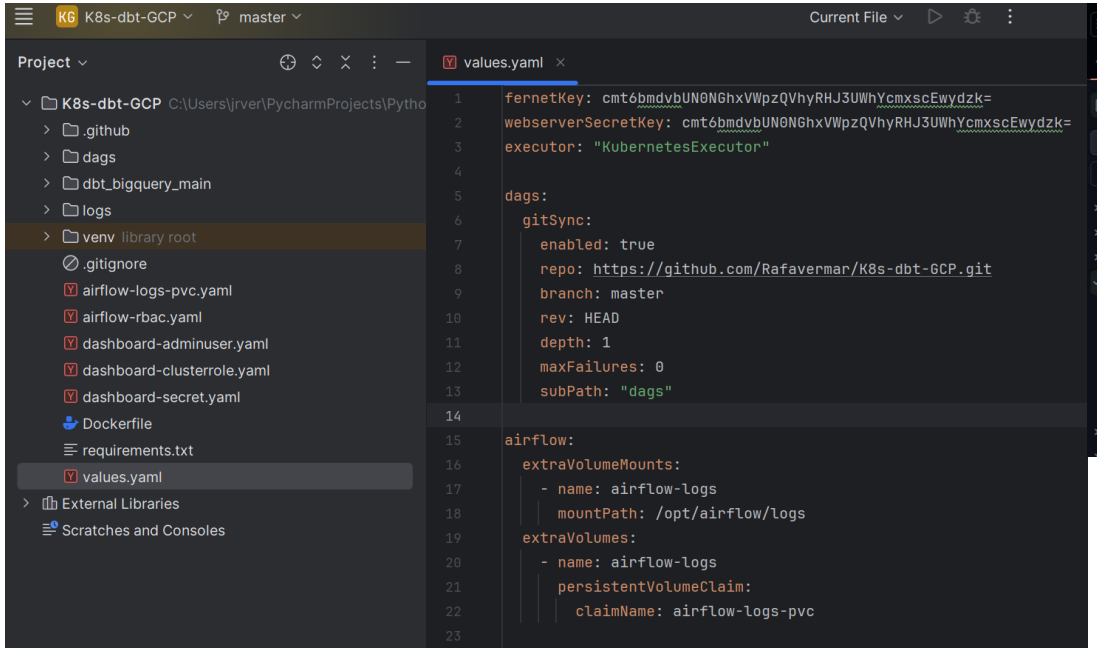
  - We have not yet created any dbt model, there are only the bydefault examples into models folder
  - After debuging you can see all connections succeded
    - That means that the configuration is correctly setup

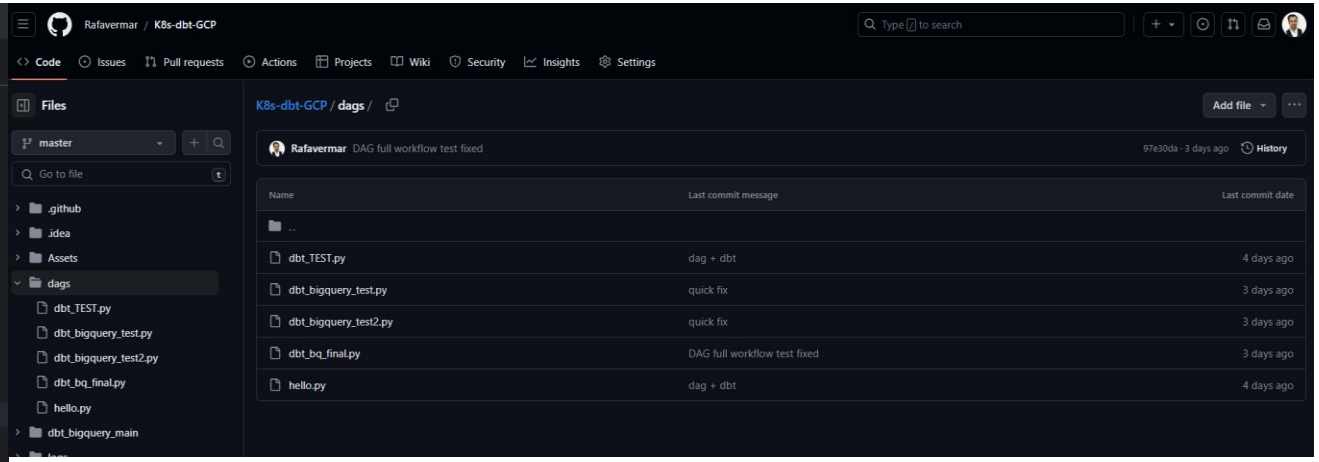


8. SYNCHRONIZE DAGs WITH GITHUB

- 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**



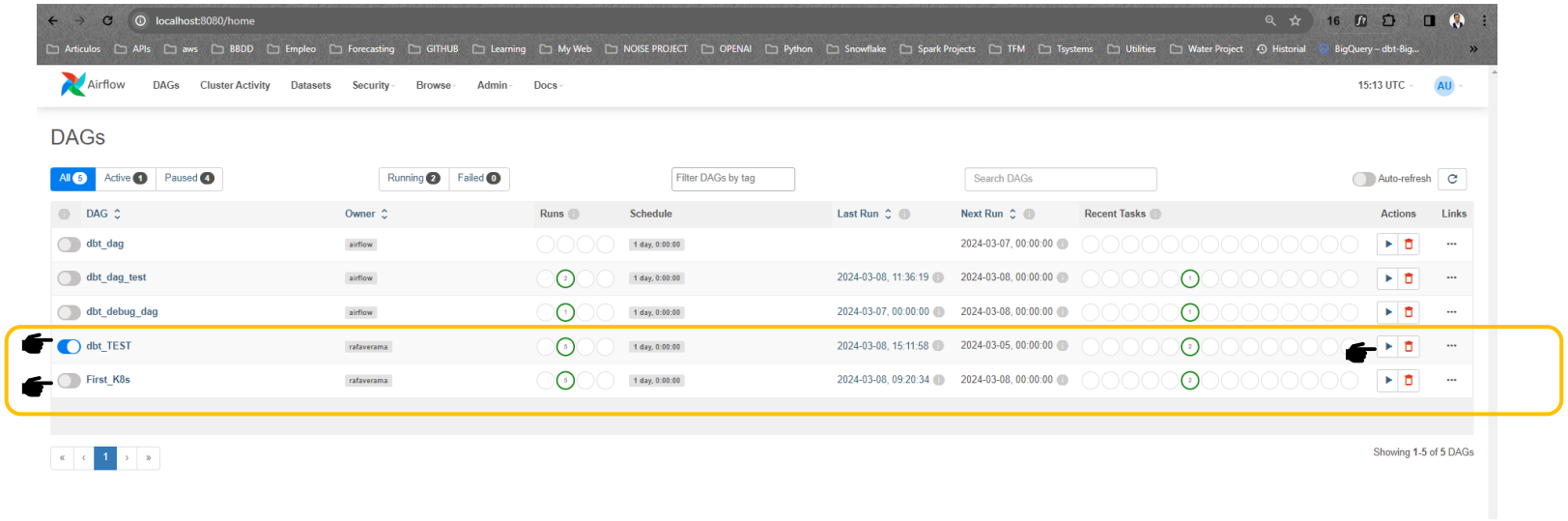


8. SYNCHRONIZE DAGS WITH GITHUB

- **3. Launch DAG : hello.py and dbt_Test**
  - Let's see both DAGs in Airflow UI (K8s pod on localhost and still fordwarding ports)
  - Note: the image below shows all the successfully run DAG. You will see only DAG names and no runs

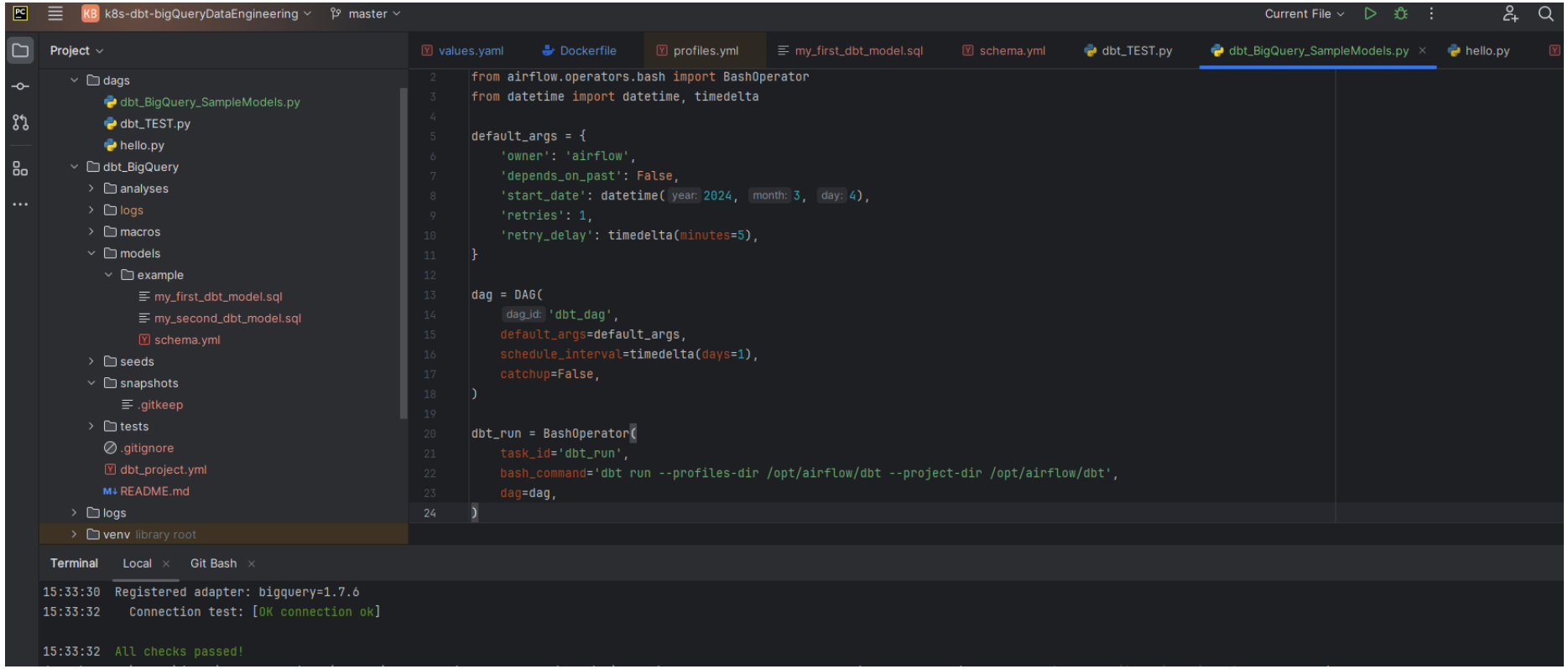
- hello.py and dbt_Test are DAGs which don't involve dbt into the orchestrated process (just for try airflow)
  - **hello.py** -> performs a simple BashOperator tasks to print something in bash
  - **dbt_Test** -> performs two simple tasks: get a .csv from my public repo, create a DF selecting few columns and preview this data

- **To do this, in Airflow UI: Activate the DAG (change to blue color) and click run on the right**



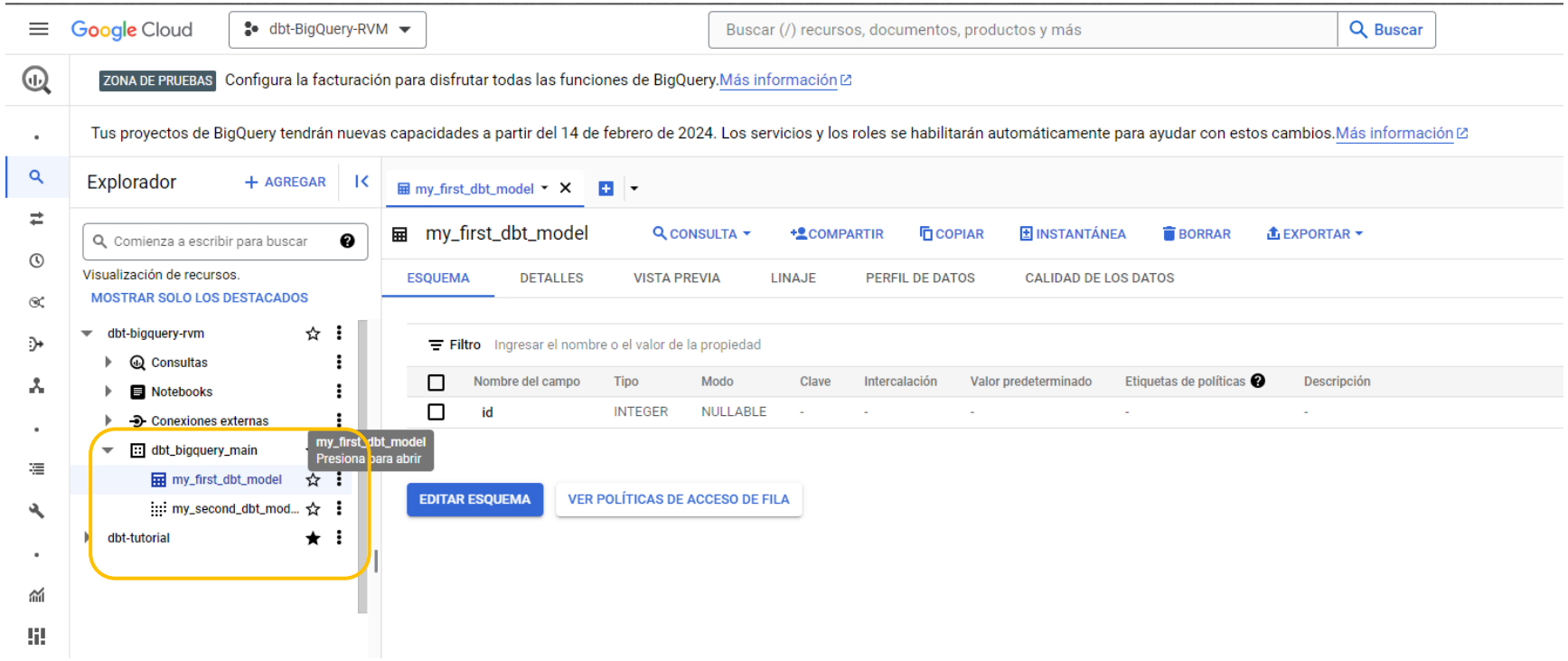
8. SYNCHRONIZE DAGS WITH GITHUB

- **4. Create a DAG as the image, commit and push it to Github**
  - Note: Dbt_BigQuery_SAmpleModels.py DOESNOT EXIST in the repo anymore.
    - It was an intermediate DAG for testing purpouses
  - 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.



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



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/requires.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:

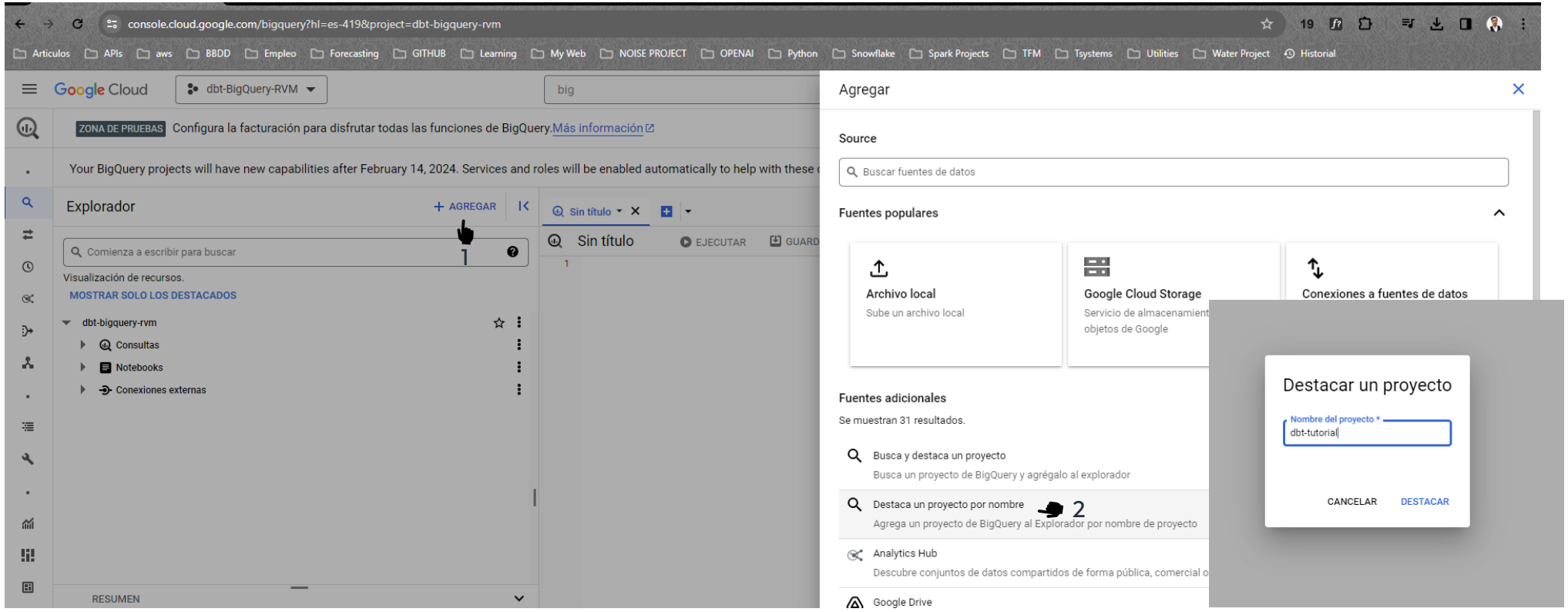
  > dbt clean
  > dbt deps

- 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
```

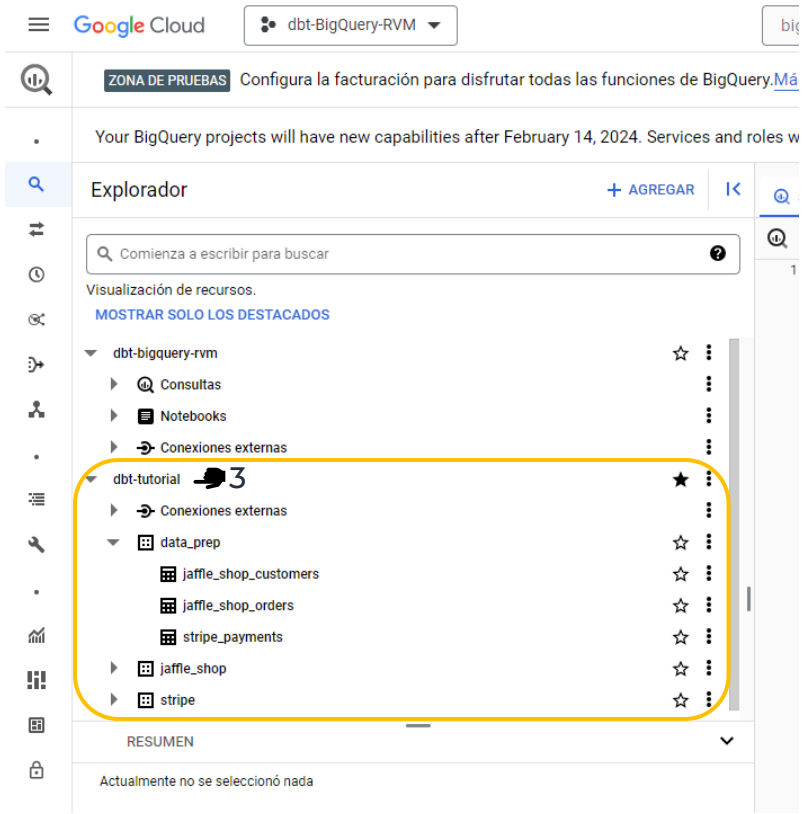
9. CREATE DBT MODEL AND ITS DAG

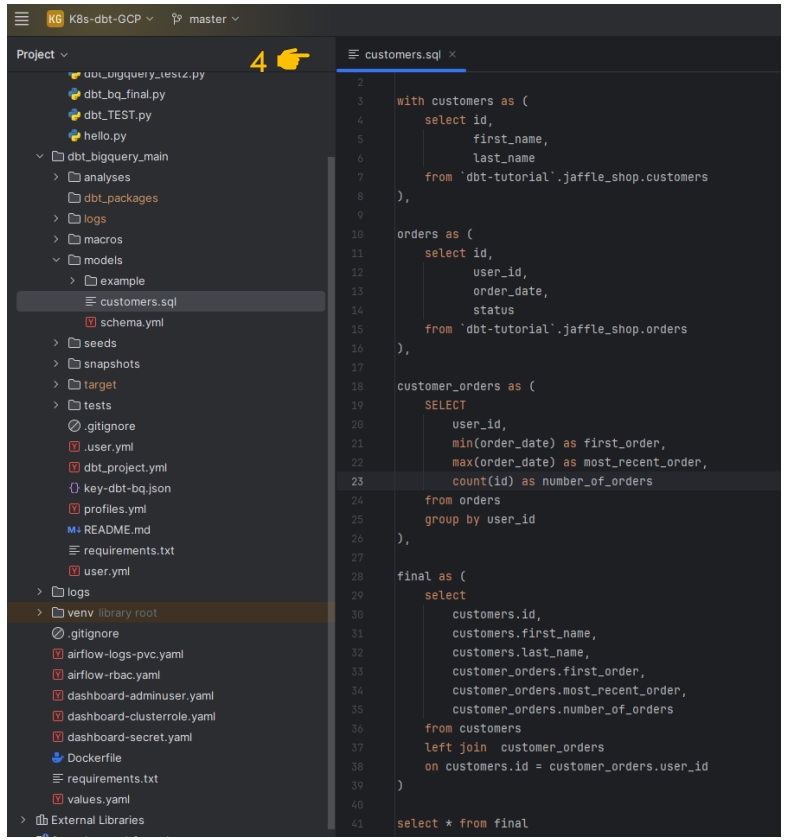
- **1**. ADD new Project by name inside our existing BigQuery Project
- **2.** Search by name: **dbt-tutorial**



9. CREATE DBT MODEL AND ITS DAG

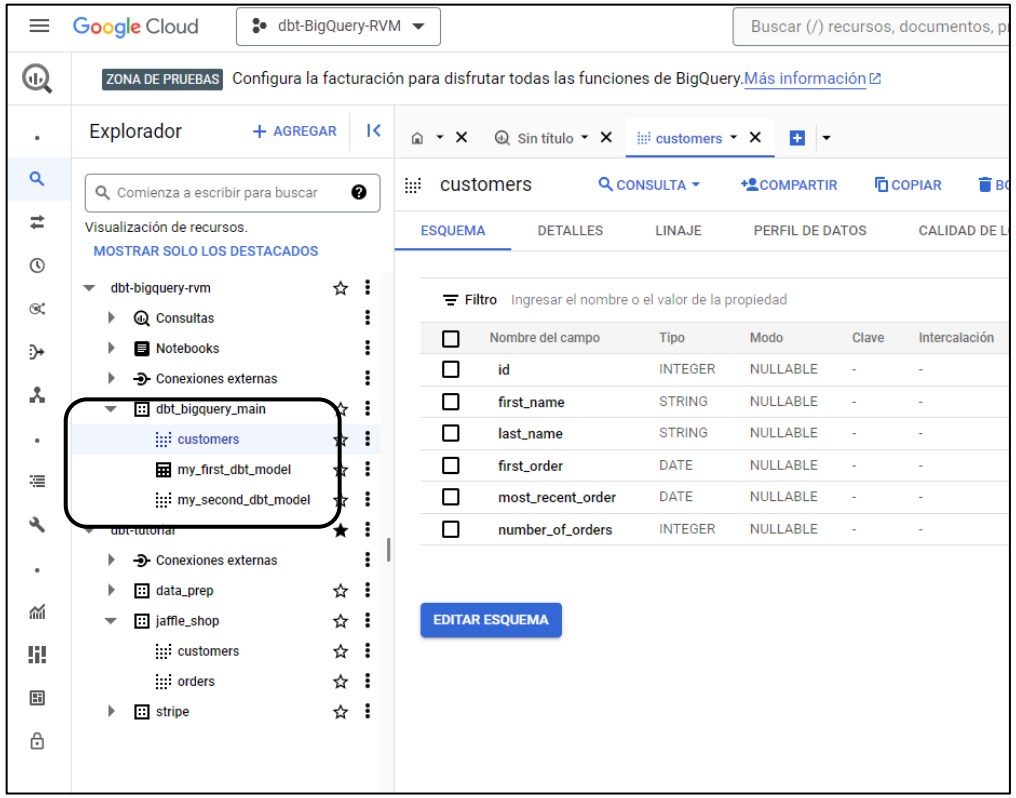
- **3.** See this Project added and marked as favourite
- This data are offered from Google for free to practice with dbt-BigQuery
- We are extracting this data from **jaffle_shop_customers** and **jaffle_shop_orders** tables to use them in our dbt model and creating a new table named: customers
- **4.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

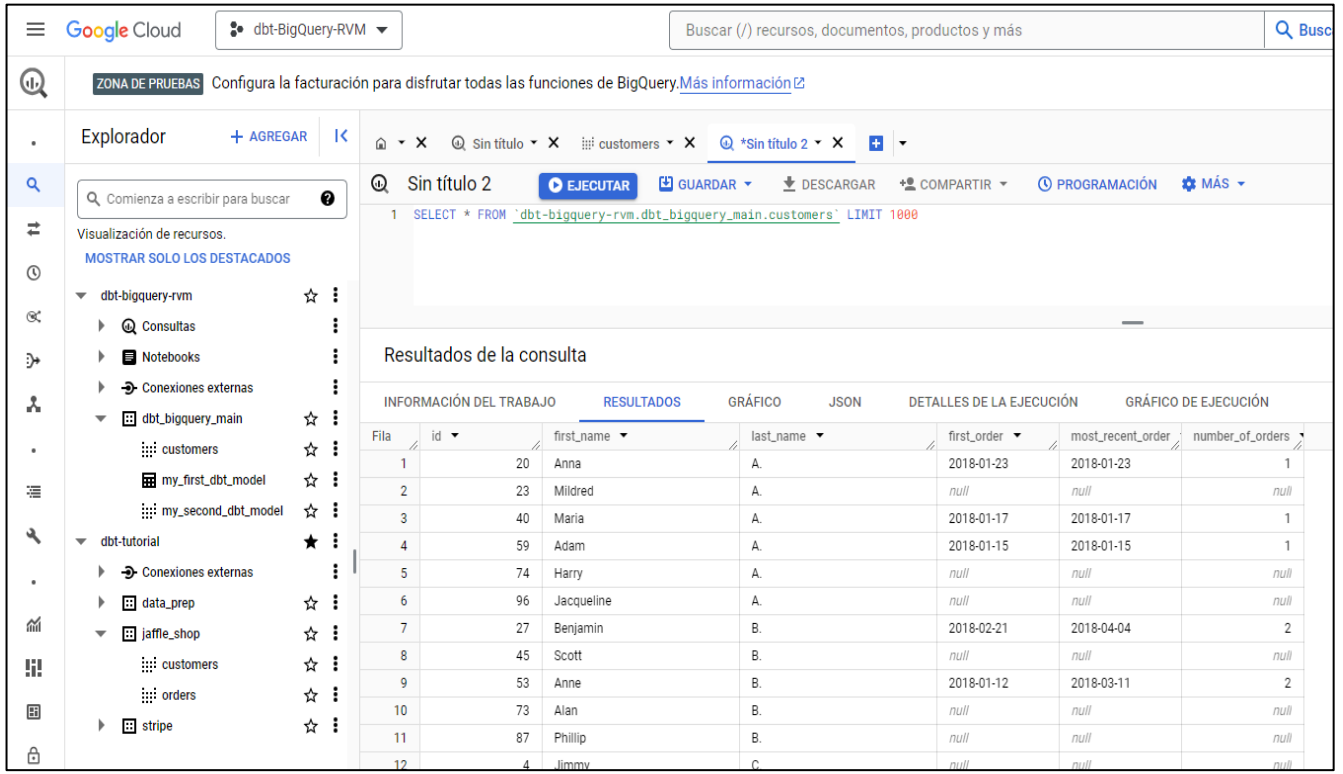




9. CREATE DBT MODEL AND ITS DAG

- **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 wash 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.



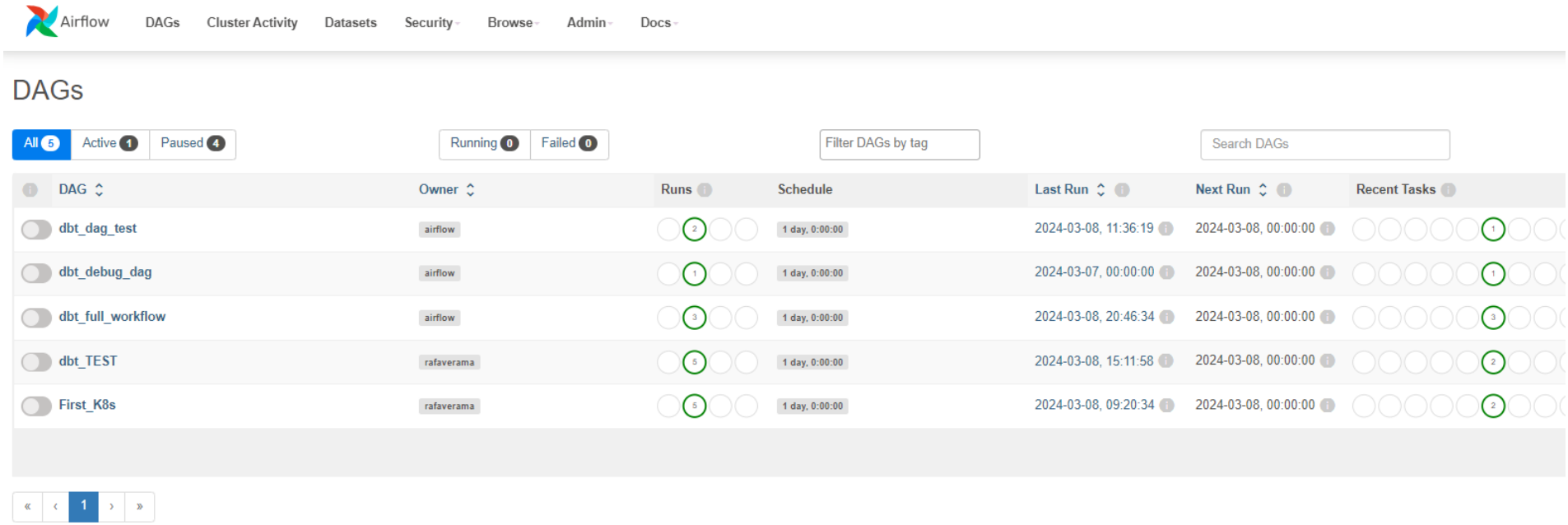


9. CREATE DBT MODEL AND ITS DAG

- 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**) -> excutes 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

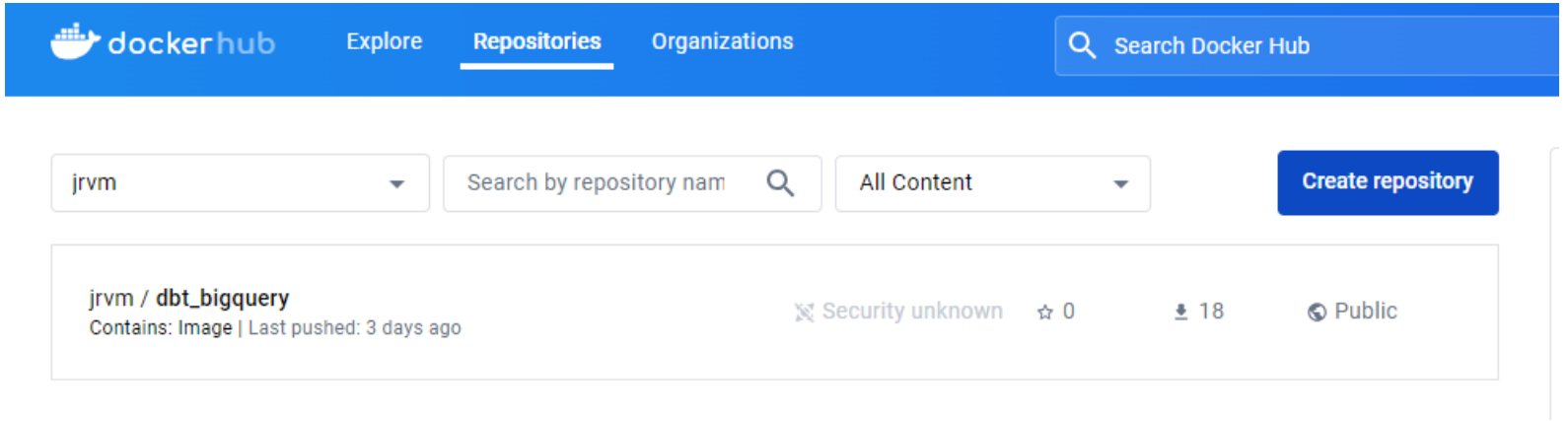
- Before running this DAGS we need to dockerize the dbt Project and set the CI/CD with **Github actions**



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.

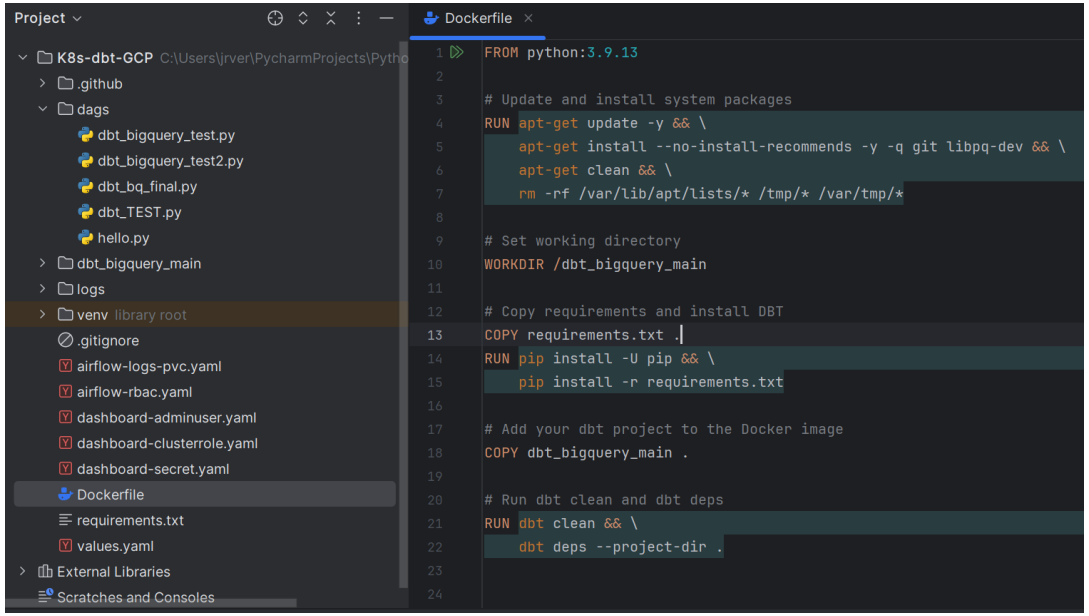


11. DOCKERIZE THE DBT PROJECT

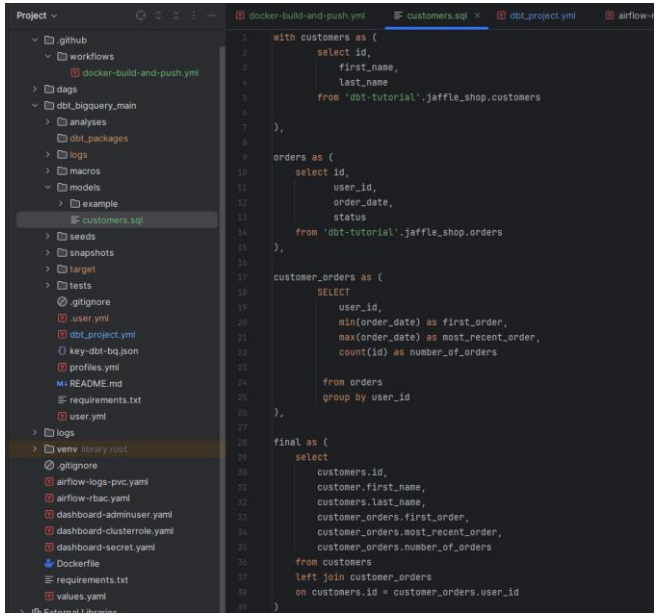
- **1**. Save all the dependencias needed into a requirements.txt
  > Pip freeze -> requirements.txt

- **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**

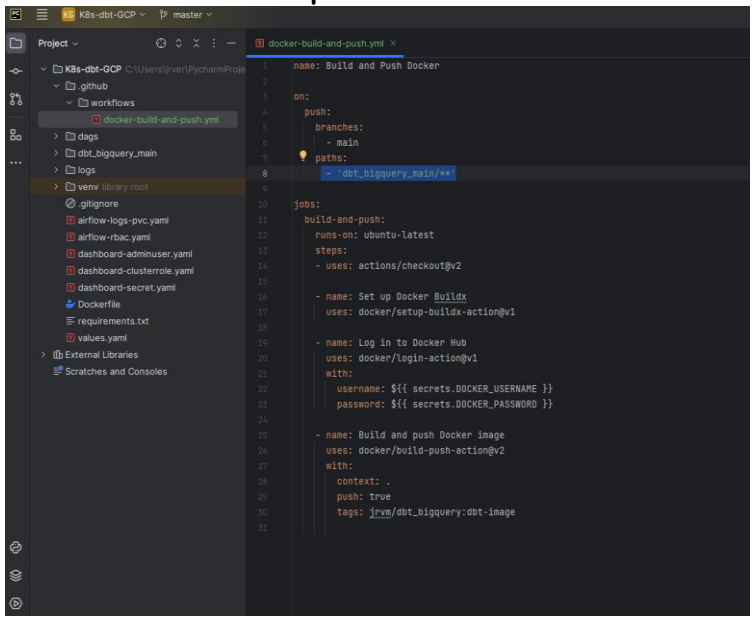
2



3

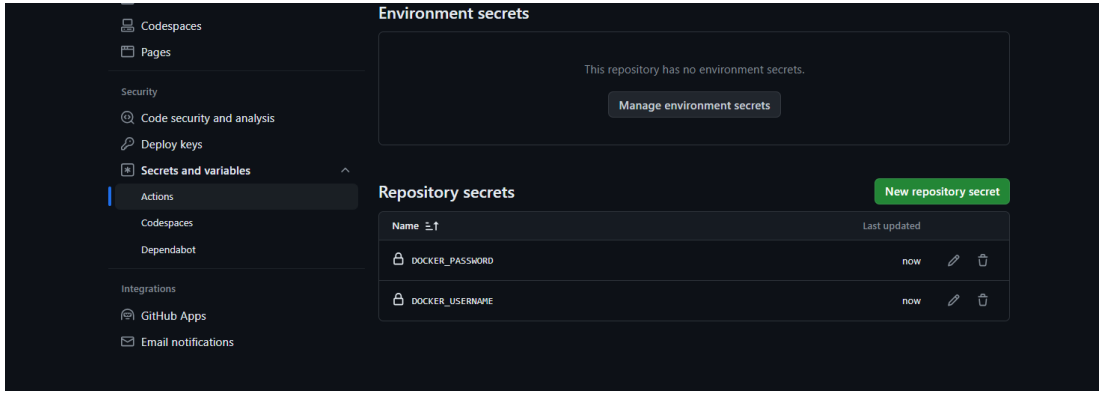


4

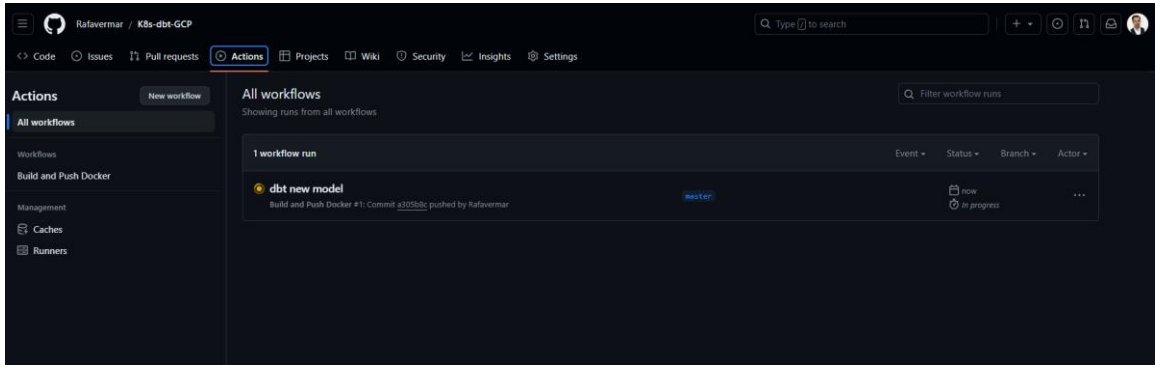


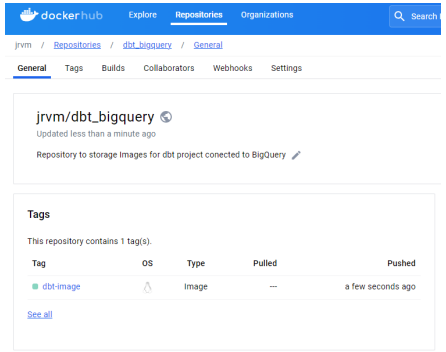
11. DOCKERIZE THE DBT PROJECT

- **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 doesnot 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.



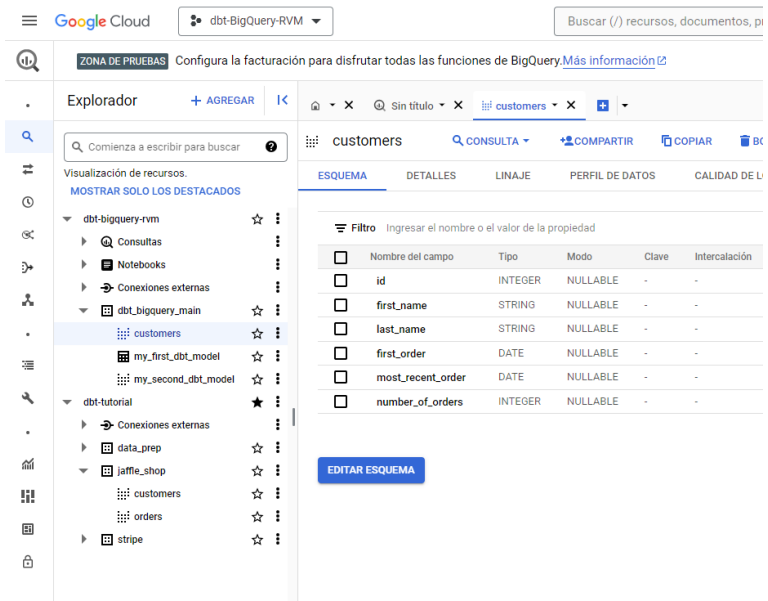


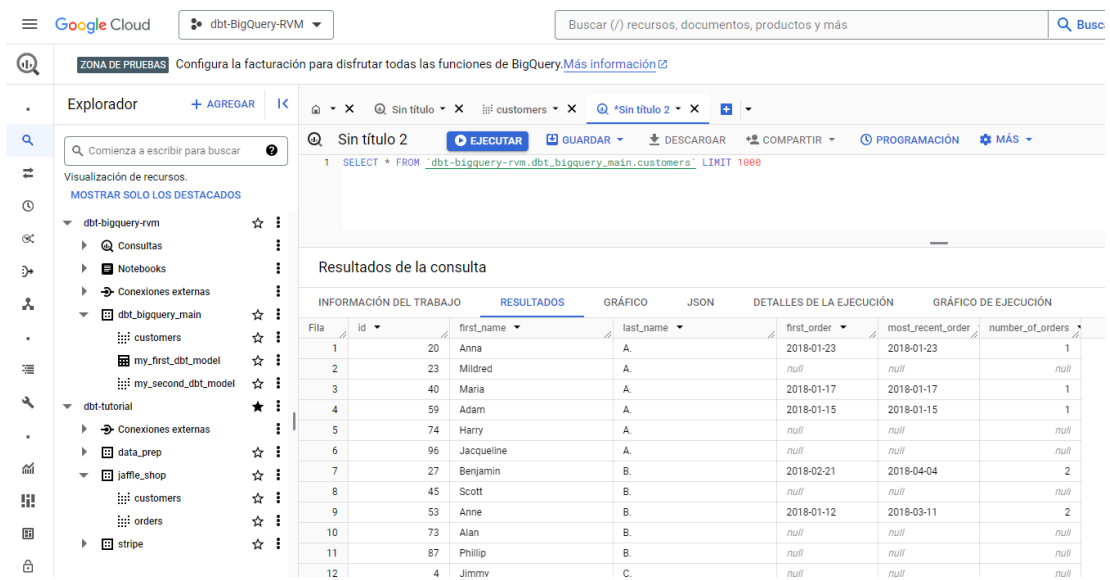
12. ORCHESTRATE CI/CD WITH GITHUB ACTIONS

- 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



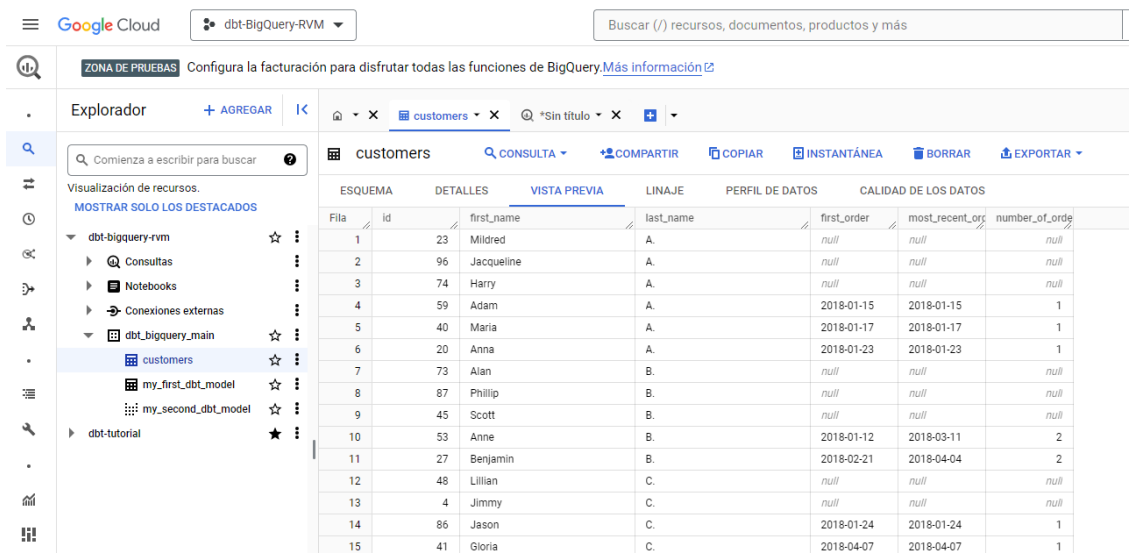


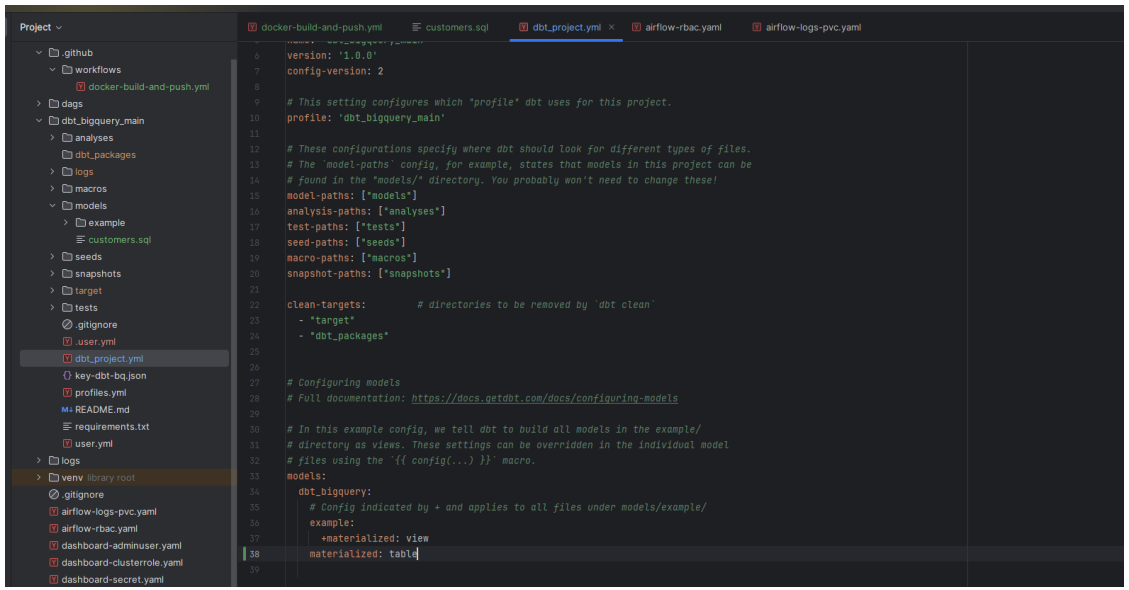
12. ORCHESTRATE CI/CD WITH GITHUB ACTIONS

- 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

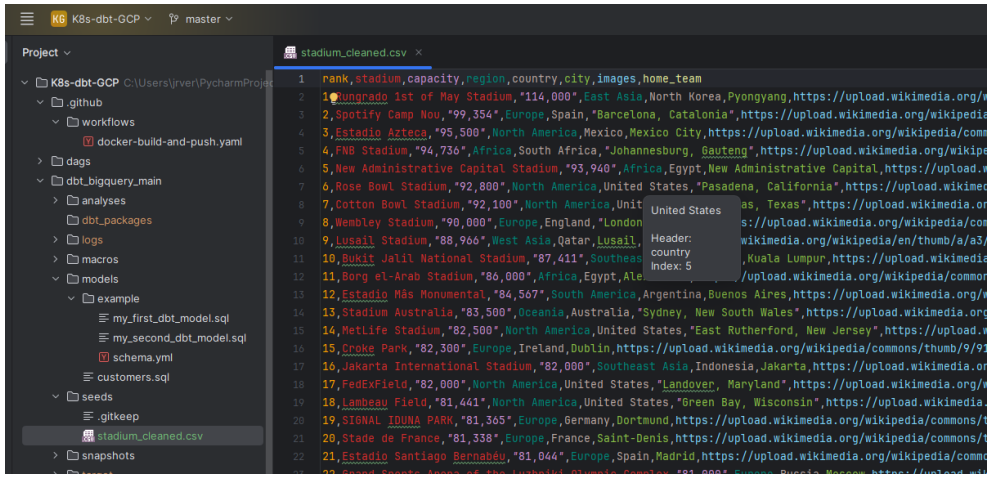
- The customers table is persisted in Bigquery as a table.

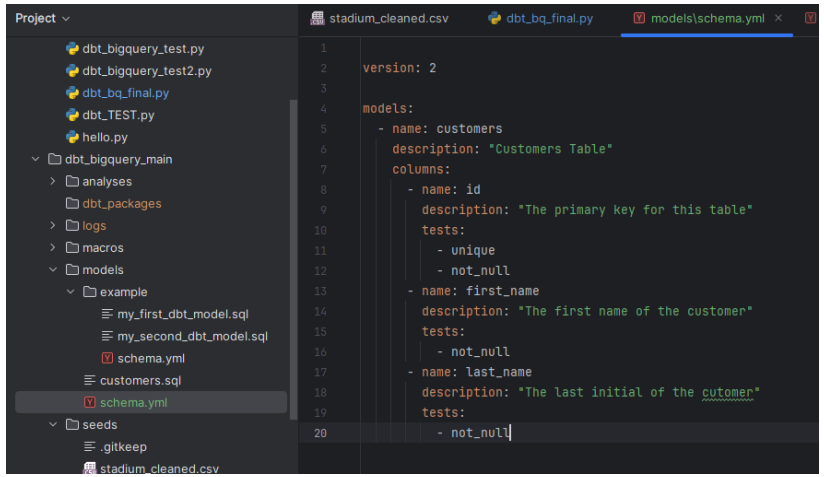


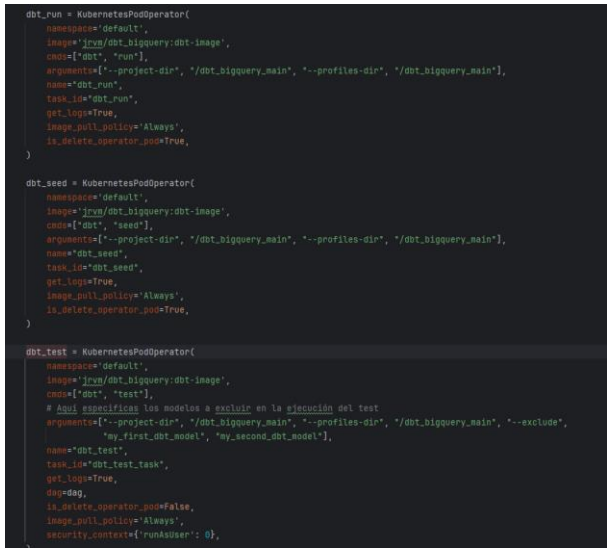


12. ORCHESTRATE CI/CD WITH GITHUB ACTIONS

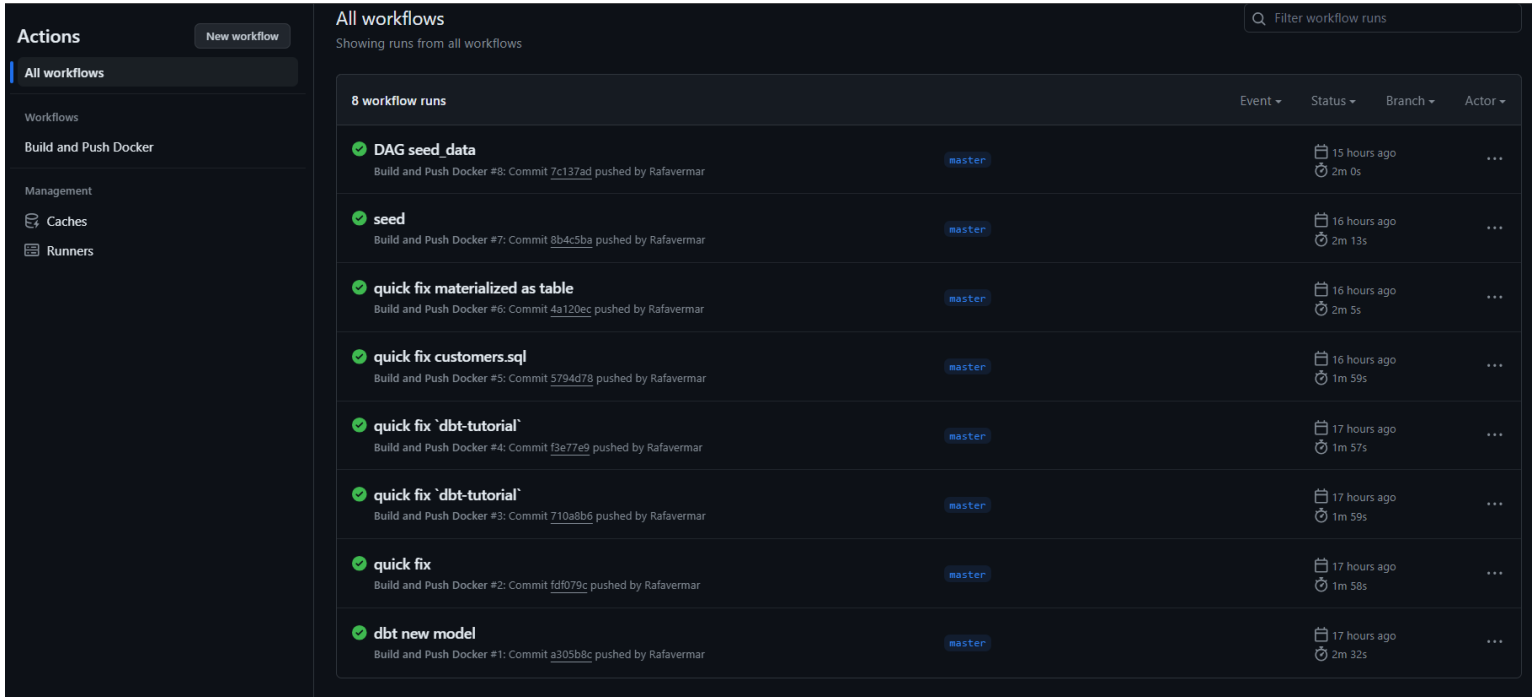
- Now you can use the same DAG before or just jump into the **dag_full_workflow**
- **To make this dbt_bq_final.py** (dag_full_workflow) works:
  - **1. Add the .csv in seed folder** → to make the dbt seed command from the DAG Works
  - **2. Create a new schema.yml** in models folders and write some test → to make dbt test command work
  - **3. Create the dbt_bq_final.py** DAG
  - **4. Commit and push to Github**
  - **5. Observe the progession 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**

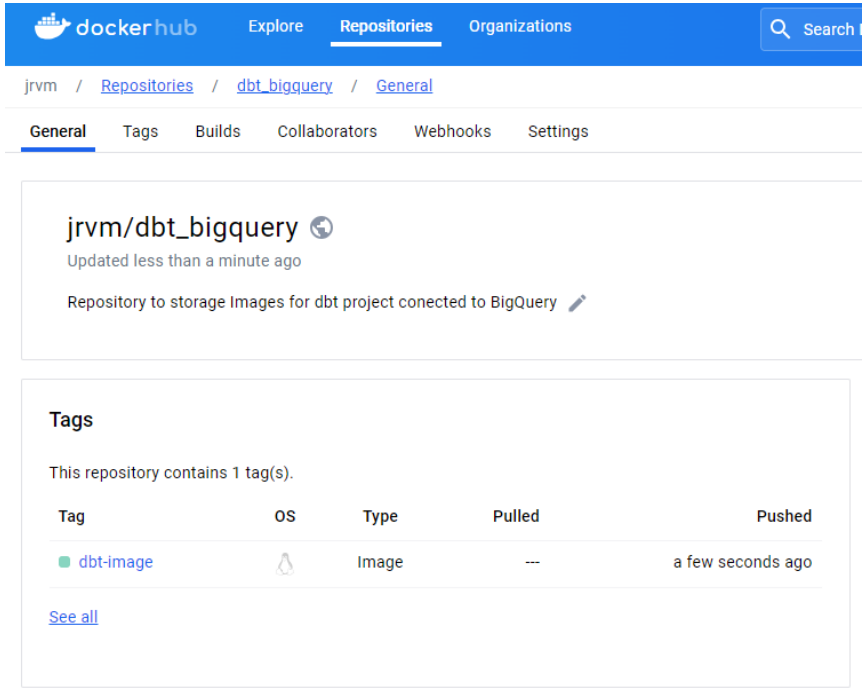




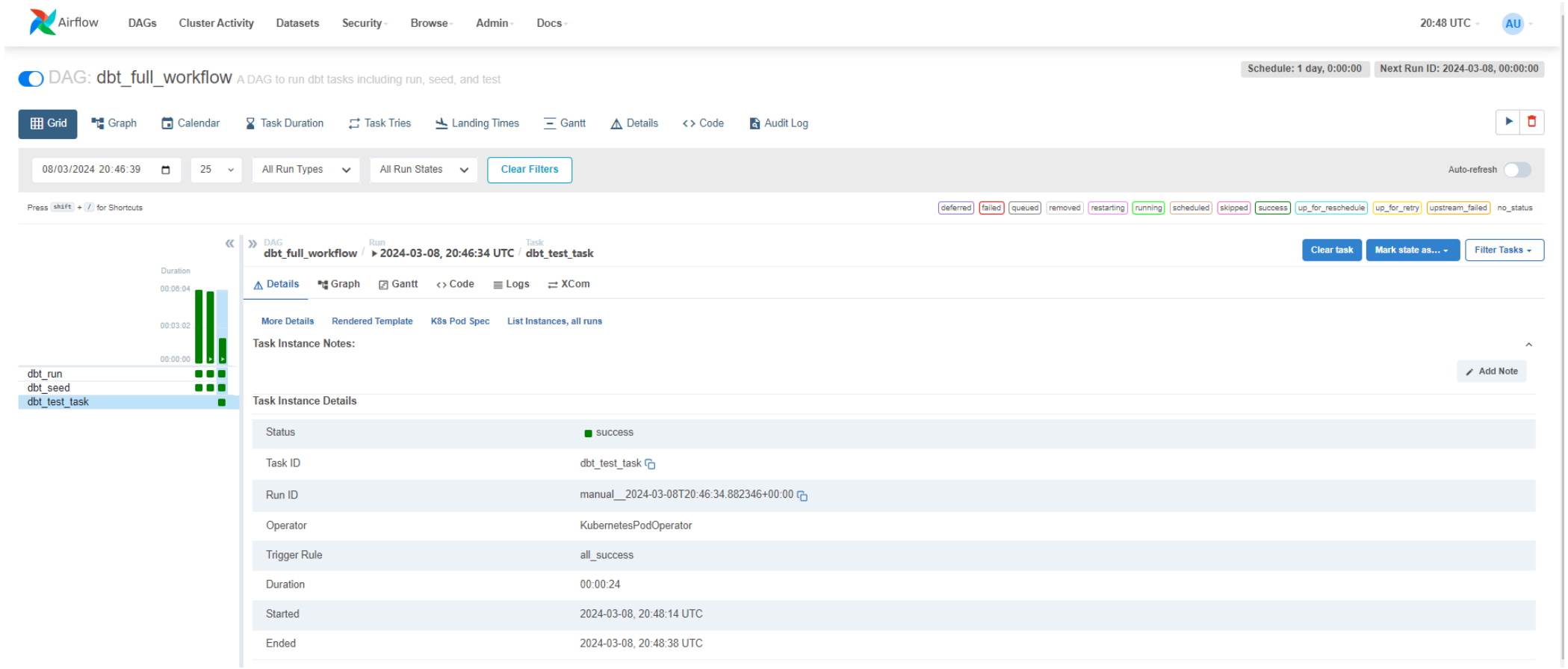


12. ORCHESTRATE CI/CD WITH GITHUB ACTIONS

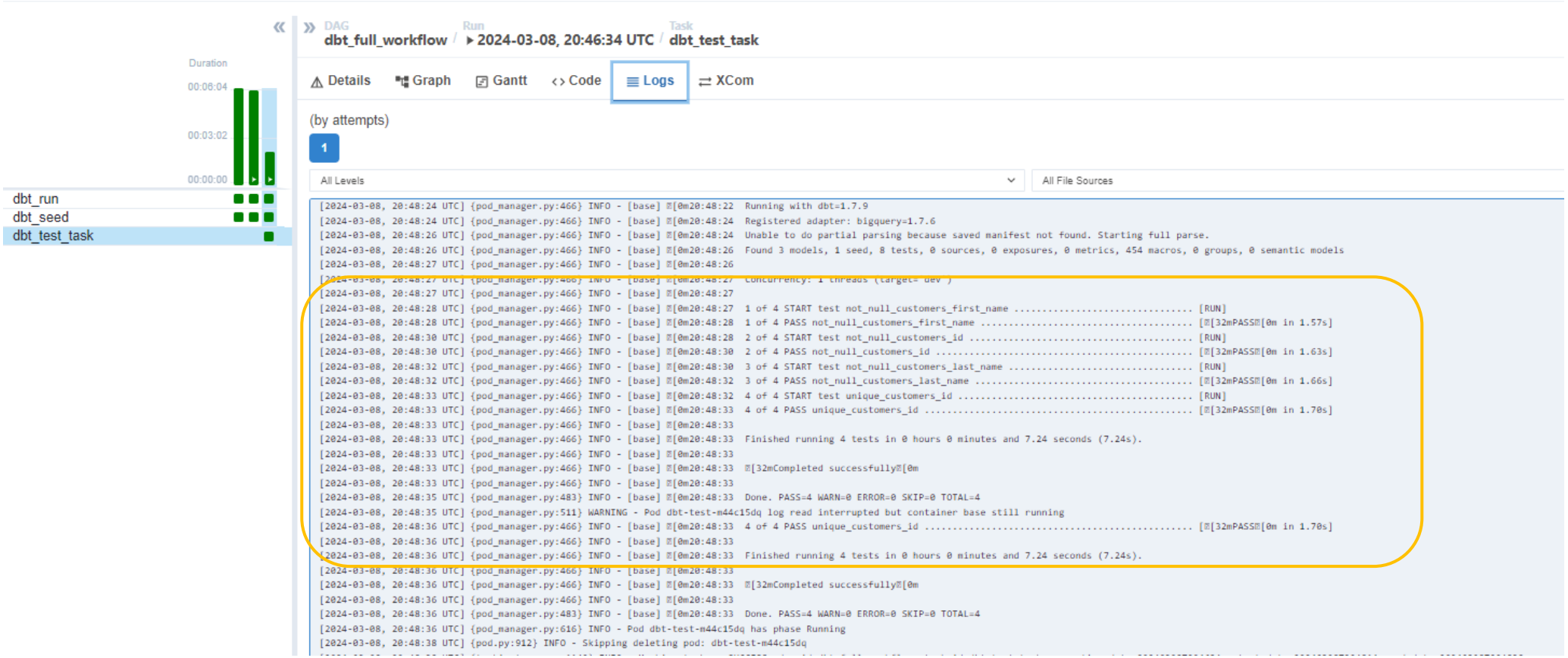




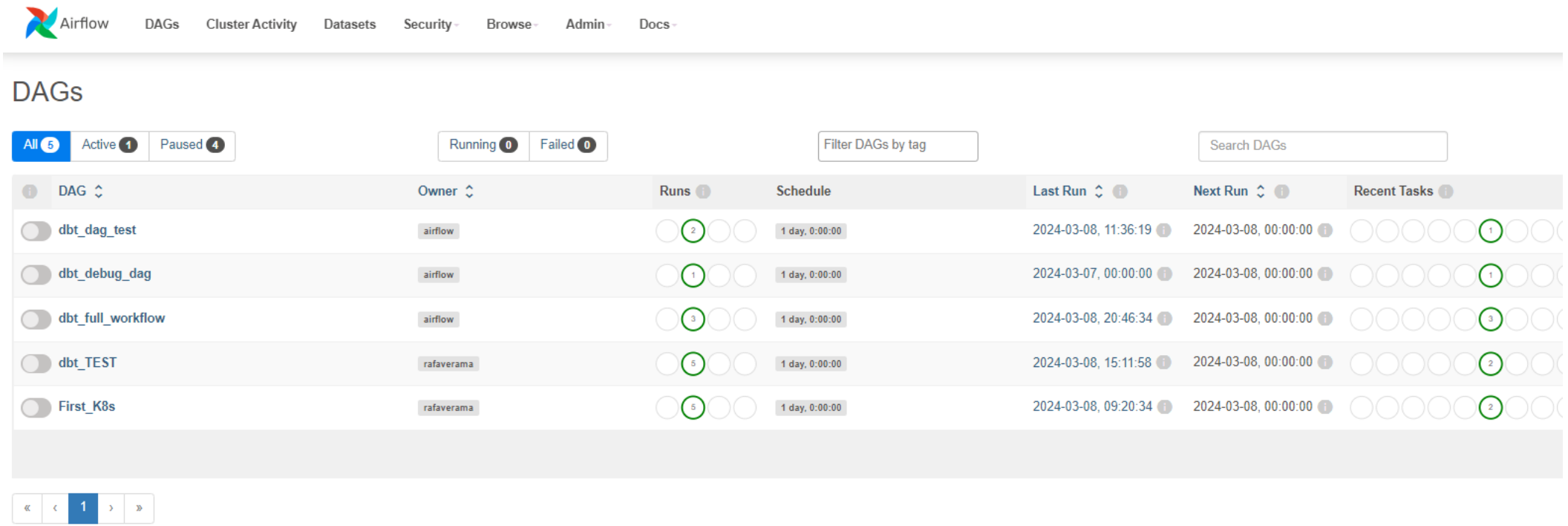
13. RUN DAG FULL WORKFLOW



13. RUN DAG FULL WORKFLOW

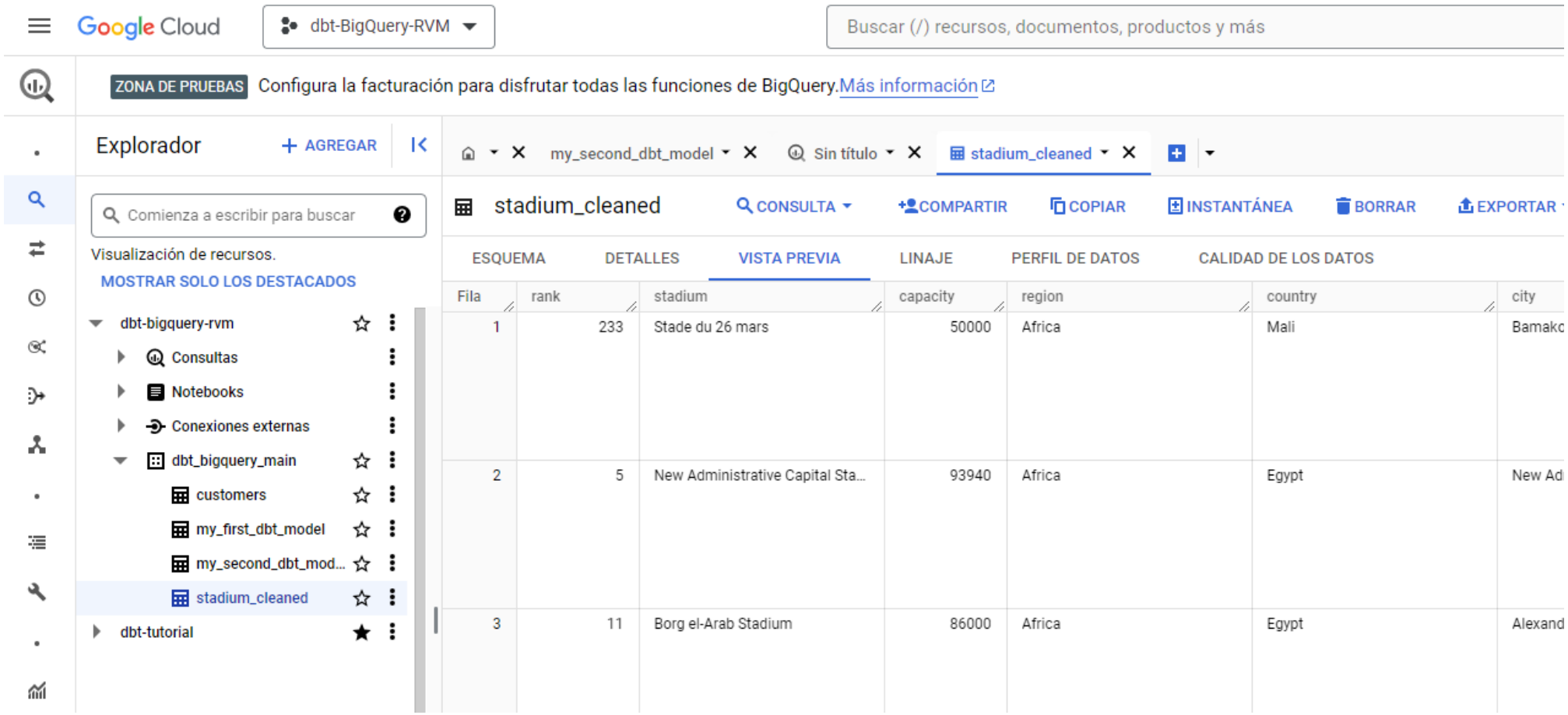


13. RUN DAG FULL WORKFLOW

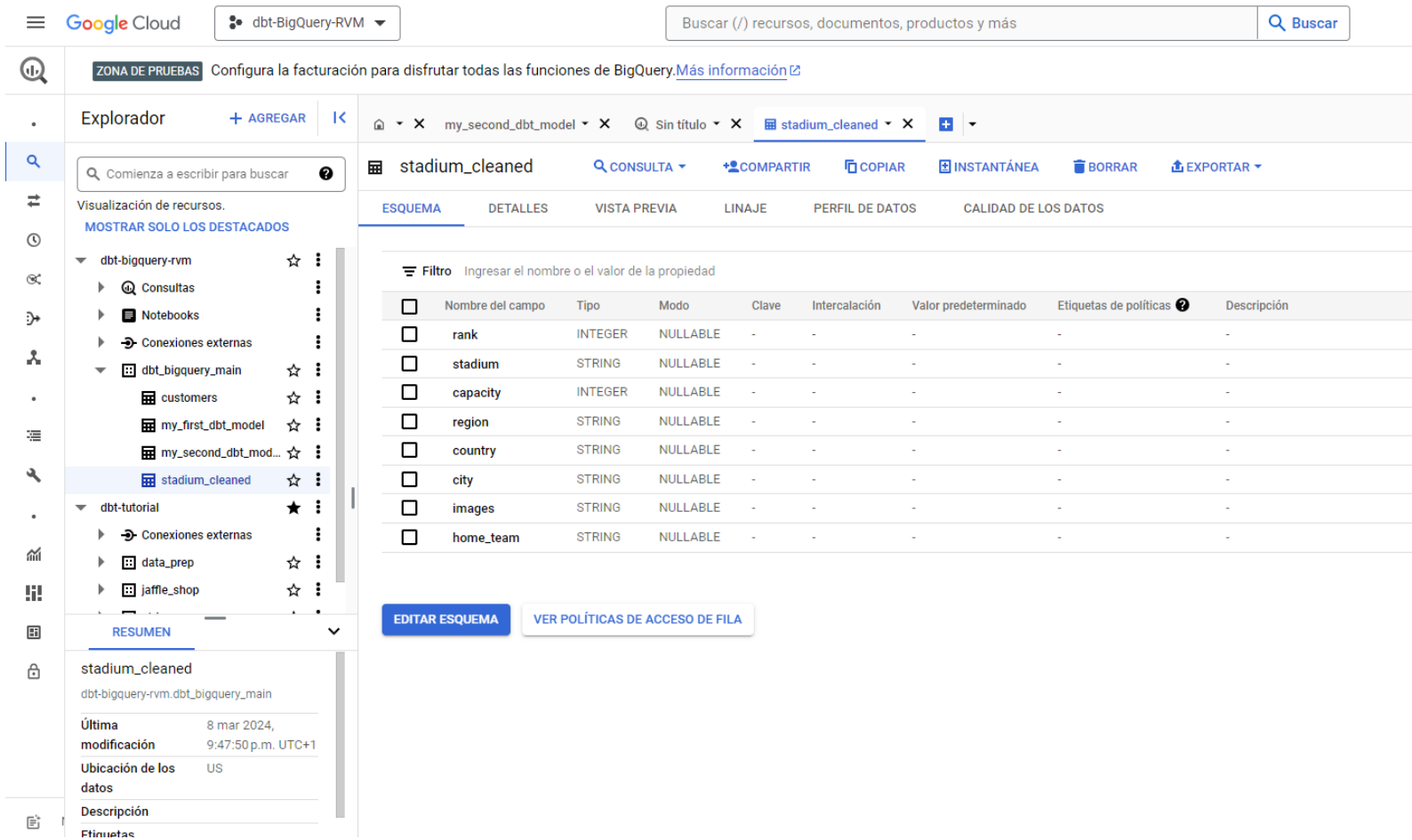


13. RUN DAG FULL WORKFLOW

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



13. RUN DAG FULL WORKFLOW



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