**PROJECT SETUP**

יש ליצור סביבה וירטואלית הסבר

<https://www.udemy.com/course/mastering-advanced-mlops-on-gcp-cicd-kubernetes-kubeflow/learn/lecture/48792775#questions>

יש ליצור סביבה , סרטון:

<https://www.udemy.com/course/mastering-advanced-mlops-on-gcp-cicd-kubernetes-kubeflow/learn/lecture/48793423#questions>

**AIRFLOW**

**עדכונים בקובץ yaml**

**להתקנות נוספות:**

    \_PIP\_ADDITIONAL\_REQUIREMENTS: ${\_PIP\_ADDITIONAL\_REQUIREMENTS:-sqlalchemy apache-airflow-providers-postgres apache-airflow-providers-google} # MODIFIED: Added sqlalchemy here

To install the GCSToLocalFilesystemOperator in Apache Airflow, you need to install the **Google Cloud (apache-airflow-providers-google) provider package**. This operator, like GCSListObjectsOperator, is part of Airflow's provider ecosystem, which offers integrations with various external services, including Google Cloud Storage (GCS).

You can achieve this by adding apache-airflow-providers-google to the \_PIP\_ADDITIONAL\_REQUIREMENTS environment variable in your docker-compose.yaml file.

*Here's how to modify your docker-compose.yaml:*

1. ***Locate the x-airflow-common section*** *in your docker-compose.yaml file.*
2. ***Find the environment block****, specifically the \_PIP\_ADDITIONAL\_REQUIREMENTS line.*
3. ***Add apache-airflow-providers-google*** *to the existing list of requirements, ensuring they are separated by commas.*

*If your line currently looks like this:*

*YAML*

*\_PIP\_ADDITIONAL\_REQUIREMENTS: ${\_PIP\_ADDITIONAL\_REQUIREMENTS:-sqlalchemy,apache-airflow-providers-postgres}*

*Change it to:*

*YAML*

*\_PIP\_ADDITIONAL\_REQUIREMENTS: ${\_PIP\_ADDITIONAL\_REQUIREMENTS:-sqlalchemy,apache-airflow-providers-postgres,apache-airflow-providers-google}*

*This ensures that when your Airflow containers start, they will install sqlalchemy, apache-airflow-providers-postgres, and the new apache-airflow-providers-google package.*

1. ***Save your docker-compose.yaml file.***
2. ***Recreate and Restart Airflow Services:*** *To ensure the new Python packages are installed inside your Docker containers, you must recreate the services. Execute the following command:*

*Bash*

*docker-compose up -d --force-recreate*

*This command will stop and remove the existing containers and then create new ones, installing the specified additional requirements during their startup.*

*Once the containers have been recreated and started, the apache-airflow-providers-google package will be available. You can then correctly import and use GCSToLocalFilesystemOperator in your DAGs:*

*Python*

*from airflow.providers.google.cloud.transfers.gcs\_to\_local import GCSToLocalFilesystemOperator*

***Important Notes:***

* ***Google Cloud Connection:*** *To use this operator, you will also need to configure an Airflow connection to Google Cloud. This typically involves setting up a service account key or other authentication methods within the Airflow UI or via environment variables.*
* ***Permissions:*** *Ensure that the Google Cloud service account or credentials used by Airflow have the necessary permissions to read from your GCS buckets and write to the local filesystem within the Airflow worker/task execution environment.*

**For using DBVEAR with postgres**

services:

  postgres:

    image: postgres:13

    environment:

      POSTGRES\_USER: airflow

      POSTGRES\_PASSWORD: airflow

      POSTGRES\_DB: airflow

    ports:

**- "5432:5432" # <--- ADD THIS LINE 5.06.25**

    volumes:

      - postgres-db-volume:/var/lib/postgresql/data

    healthcheck:

      test: ["CMD", "pg\_isready", "-U", "airflow"]

      interval: 10s

      retries: 5

      start\_period: 5s

    restart: always

*It sounds like you've successfully created a PostgreSQL database within a Docker container using Airflow, but you're encountering connection issues from DBeaver on your host machine. This is a common situation when working with Docker, as containers have their own isolated network. The "Connection refused" error means DBeaver can't reach the PostgreSQL server inside the container. Here's a breakdown of how to troubleshoot and connect:*

***Understanding Docker Networking***

*By default, Docker containers are isolated from the host machine's network. When you run a container, Docker sets up a virtual network. To access services running inside a container (like your PostgreSQL database), you need to expose ports and potentially use Docker networking features.*

***Troubleshooting Steps***

1. ***Check Port Mapping (Crucial):***
   * *When you run your PostgreSQL container (likely as part of your Airflow setup), you should have mapped a port on your host machine to port 5432 (the default PostgreSQL port) inside the container. This is typically done with the -p flag in the docker run command or in your docker-compose.yml file.*
   * ***Example:*** *-p 5432:5432 maps host port 5432 to container port 5432. You might have used a different host port (e.g., -p 6543:5432).*
   * ***How to check:***
     + *If you used docker run: Inspect the command you used to start the container.*
     + *If you used docker-compose: Examine your docker-compose.yml file. Look for the ports section in the PostgreSQL service definition.*
     + *You can also use docker ps to list running containers and see their port mappings. The output will show something like 0.0.0.0:5432->5432/tcp.*
2. ***DBeaver Connection Settings (Most Likely Fix):***
   * *In DBeaver, the* ***Host*** *should usually be localhost or 127.0.0.1. This tells DBeaver to connect to a service on your own machine.*
   * *The* ***Port*** *in DBeaver* ***must match the host port*** *you mapped in the Docker command or docker-compose.yml. If you used -p 6543:5432, the DBeaver port should be 6543, not 5432.*
   * *The default PostgreSQL port inside the container (5432) is irrelevant to DBeaver; it only cares about the port exposed on the host.*
3. ***Docker Network (Less Common for localhost):***
   * *If you're using a more complex Docker setup with multiple containers that need to communicate, you might have created a Docker network. In most cases, for connecting from the host machine to a container, this isn't necessary. However, if you are using a Docker network, you might need to use the container name or a service name as the hostname in DBeaver, instead of localhost. This is less likely if you're connecting from the host.*
4. ***Firewall (Unlikely for localhost):***
   * *It's unlikely that your host firewall is blocking connections to localhost on a mapped port, but it's worth a quick check if the above steps don't work. Temporarily disable your firewall to see if it makes a difference (and then re-enable it!).*

***Example Scenario***

*Let's say your docker-compose.yml has this:*

*YAML*

*services:*

*postgres:*

*image: postgres:latest*

*ports:*

*- "6543:5432" # Host port 6543 mapped to container port 5432*

*environment:*

*POSTGRES\_USER: myuser*

*POSTGRES\_PASSWORD: mypassword*

*POSTGRES\_DB: mydb*

*In this case, your DBeaver connection settings should be:*

* ***Host:*** *localhost or 127.0.0.1*
* ***Port:*** *6543 (the host port)*
* ***Username:*** *myuser (from the docker-compose.yml)*
* ***Password:*** *mypassword (from the docker-compose.yml)*
* ***Database:*** *mydb (from the docker-compose.yml)*

***Troubleshooting Steps Summary***

1. ***Verify your Docker port mapping.*** *This is the most critical step.*
2. ***Ensure DBeaver uses the correct host port.***
3. *Double-check your PostgreSQL username, password, and database name in DBeaver, matching the values defined in your Docker setup.*

*By carefully checking your Docker port mappings and DBeaver connection settings, you should be able to resolve the "Connection refused" error.*

**DOCKER airflow**

Open docker in desktop

CD to airflow folder

Where yaml file

Run

**DOCKER COMMANDS**

**DELETE: docker compose down -v --remove-orphans**

*Here are some basic and commonly used Docker commands:*

***1. Managing Images:***

* ***docker pull <image\_name>:<tag>****: Downloads an image from a registry (like Docker Hub) to your local machine.* 
  + *Example: docker pull ubuntu:latest*
  + *Example: docker pull apache/airflow:3.0.0*
* ***docker images****: Lists all images downloaded on your local machine.*
* ***docker rmi <image\_id\_or\_name>****: Removes one or more images.* 
  + *Example: docker rmi ubuntu:latest*
* ***docker build -t <image\_name>:<tag> .****: Builds a Docker image from a Dockerfile in the current directory. The . specifies the build context.*

***2. Managing Containers:***

* ***docker run <image\_name>:<tag>****: Creates and starts a new container from an image.* 
  + *Example: docker run -d -p 80:80 nginx (runs Nginx in detached mode, mapping host port 80 to container port 80)*
  + *--name <container\_name>: Assigns a custom name to the container.*
  + *-d (detached mode): Runs the container in the background.*
  + *-p <host\_port>:<container\_port>: Publishes (maps) a container's port to a host port.*
  + *-v <host\_path>:<container\_path>: Mounts a host path into the container.*
  + *-e <ENV\_VAR>=<value>: Sets environment variables inside the container.*
* ***docker ps****: Lists all running containers.*
* ***docker ps -a****: Lists all containers, including stopped ones.*
* ***docker stop <container\_id\_or\_name>****: Stops a running container gracefully.*
* ***docker start <container\_id\_or\_name>****: Starts a stopped container.*
* ***docker restart <container\_id\_or\_name>****: Restarts a running or stopped container.*
* ***docker rm <container\_id\_or\_name>****: Removes one or more stopped containers.*
* ***docker logs <container\_id\_or\_name>****: Fetches the logs of a container.* 
  + *-f (follow): Continuously streams new logs.*
* ***docker exec -it <container\_id\_or\_name> <command>****: Executes a command inside a running container.* 
  + *-i (interactive): Keeps STDIN open even if not attached.*
  + *-t (tty): Allocates a pseudo-TTY.*
  + *Example: docker exec -it my\_container /bin/bash (opens a bash shell inside the container)*

***3. Docker Compose (for multi-container applications):***

* ***docker-compose up****: Builds, creates, starts, and attaches to containers defined in a docker-compose.yaml file.* 
  + *-d: Runs containers in detached mode (background).*
  + *--build: Rebuilds images before starting containers.*
  + *--force-recreate: Recreates containers even if their configuration hasn't changed.*
  + *--no-deps: Starts a service without starting its dependencies.*
* ***docker-compose down****: Stops and removes containers, networks, and volumes created by docker-compose up.* 
  + *-v (or --volumes): Removes anonymous volumes associated with services.*
  + *--remove-orphans: Removes services that are not defined in the Compose file but were started by Compose.*
* ***docker-compose restart <service\_name>****: Restarts specific services defined in your docker-compose.yaml.*
* ***docker-compose logs <service\_name>****: Displays log output from services.*

***4. Cleaning Up (Useful for development):***

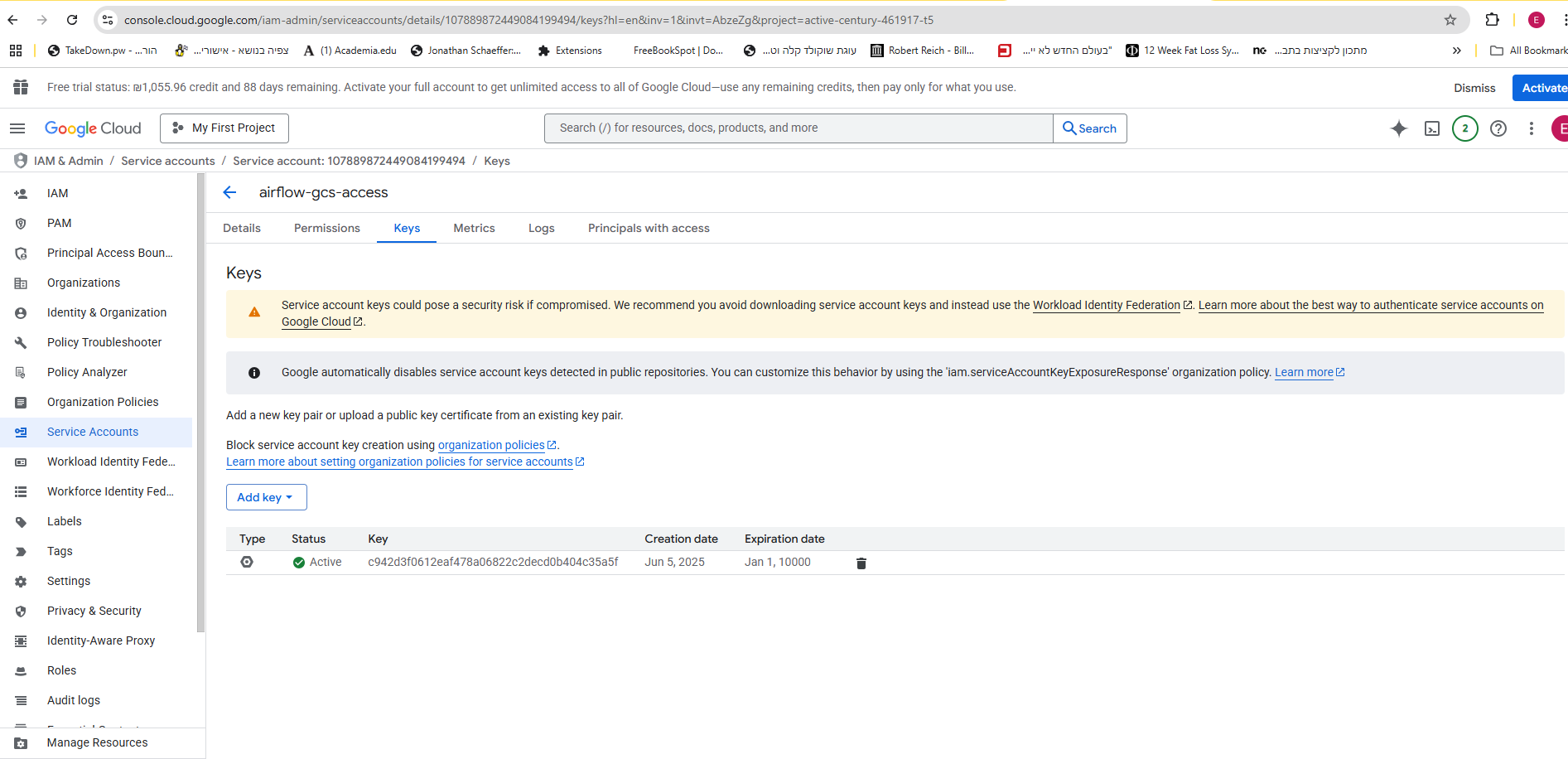
* ***docker system prune****: Removes all stopped containers, all dangling images, all dangling networks, and build cache.* 
  + *-a (or --all): Removes all unused images (not just dangling ones) and all unused containers.*
  + *--volumes: Also removes all unused volumes. Use with caution as this deletes data!*

**FILES IN GOOGLE CLOUD**

**https://console.cloud.google.com/welcome/new?\_gl=1\*8ibuqh\*\_up\*MQ..&gclid=Cj0KCQjwgIXCBhDBARIsAELC9ZjWzPxNbgxiics3bL1ez0fSruMaBc7q6l-CmgWL3Cot77atcGyfXGkaAmLaEALw\_wcB&gclsrc=aw.ds&hl=en&inv=1&invt=AbzeKw&project=active-century-461917-t5**

<https://console.cloud.google.com/storage/browser?inv=1&invt=AbzfCw&project=active-century-461917-t5&prefix=&forceOnBucketsSortingFiltering=true&bucketType=live>

Create KEY



Need to set up permissions

Go to the 3 lines

תמונה שמכילה גופן, טקסט, לוגו, לבן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

Go to

תמונה שמכילה טקסט, צילום מסך, מספר, תוכנה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

*Setting permissions in Google Cloud Platform (GCP) is primarily managed through* ***Identity and Access Management (IAM)****. IAM allows you to define precisely who has what access to which resources within your GCP project.*

*Here are the core concepts of GCP IAM:*

* ***Members (Who):*** *These are the identities that can perform actions. They can include:* 
  + *Google Accounts (individual users)*
  + *Service Accounts (applications or virtual machines that need to authenticate to GCP services without user involvement)*
  + *Google Groups*
  + *Google Workspace (formerly G Suite) domains*
* ***Roles (What):*** *Roles are collections of permissions. Instead of granting individual permissions, you assign roles to members. Roles can be:* 
  + ***Primitive Roles:*** *Such as Owner, Editor, or Viewer, which are broad and generally not recommended for fine-grained control.*
  + ***Predefined Roles:*** *These are granular roles designed for specific GCP services (e.g., "Storage Object Viewer," "BigQuery Data Editor"). These are highly recommended for more specific access.*
  + ***Custom Roles:*** *If predefined roles do not meet your exact requirements, you can create custom roles by combining specific permissions.*
* ***Resources (Where/What):*** *These are the GCP entities that members can access. Permissions can be granted at various levels of the resource hierarchy:* 
  + ***Organization Level:*** *Grants access to all projects within an organization.*
  + ***Folder Level:*** *Grants access to all projects within a folder.*
  + ***Project Level:*** *Grants access to all resources within a specific project.*
  + ***Resource Level:*** *Grants access to a specific resource, such as a GCS bucket, BigQuery dataset, or Cloud SQL instance.*

*Here's how to set permissions step-by-step using the Google Cloud Console, using the example of granting a Service Account (like airflow-gcs-access@...iam.gserviceaccount.com) permissions to a Google Cloud Storage (GCS) bucket (eb\_mlops\_bucket):*

1. ***Access the Google Cloud Console:*** *Open your web browser and go to https://console.cloud.google.com/.*
2. ***Select Your Project:*** *Use the top-left dropdown to select the correct Google Cloud Project where your resources are located.*
3. ***Navigate to the Resource or IAM Page:***
   * *For project-level permissions, go to IAM & Admin -> IAM.*
   * *For specific resource permissions (e.g., a GCS bucket), navigate to the specific service (e.g., Cloud Storage -> Buckets), then locate and click on the specific bucket (eb\_mlops\_bucket in your case), and finally click on the* ***"Permissions" tab****.*
4. ***Add or Edit a Principal:***
   * *To add new permissions, click the* ***"+ GRANT ACCESS"*** *button.*
   * *To edit existing permissions, find the member and click the "Edit principal" icon next to their entry.*
5. ***Specify the Member (Who):*** *In the "New principals" field, enter the email address of the* ***Service Account*** *(e.g., airflow-gcs-access@active-century-461917-t5.iam.gserviceaccount.com), Google Account, or Group.*
6. ***Select the Role(s) (What):*** *In the "Select a role" dropdown, choose the appropriate role. For GCS object access, common roles include:* 
   * ***Storage Object Viewer****: Grants storage.objects.list (list files) and storage.objects.get (read file content) permissions, suitable for GCSListObjectsOperator.*
   * ***Storage Object Creator****: Grants permission to upload new objects.*
   * ***Storage Object Admin****: Grants full control over objects (list, create, delete, read). Always choose the* ***least privileged role*** *necessary for the service account to perform its tasks.*
7. ***Add Another Role (Optional):*** *If the member requires multiple roles, click "ADD ANOTHER ROLE" and select additional roles.*
8. ***Save Changes:*** *Click* ***"Save"*** *or* ***"Done"*** *to apply the new permissions.*

*After these steps, the service account will have the specified access to the resource, allowing your Airflow DAGs to perform GCS operations without "403 Forbidden" errors.*

*Add connections:*

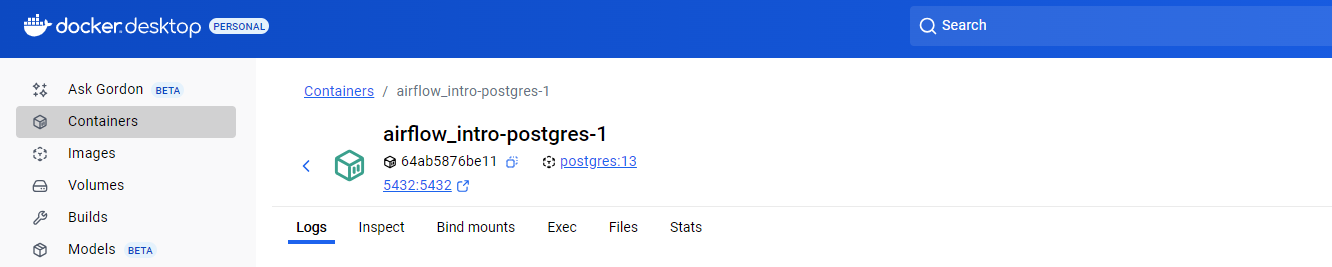
*Airflow*

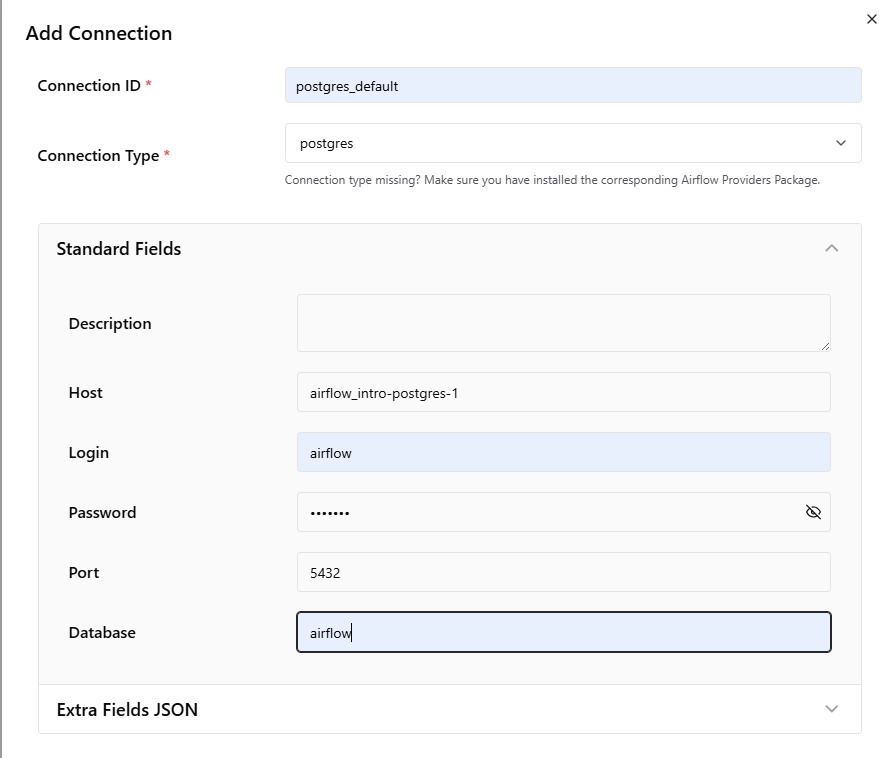
*Admin--🡪connections*

*Add connection*

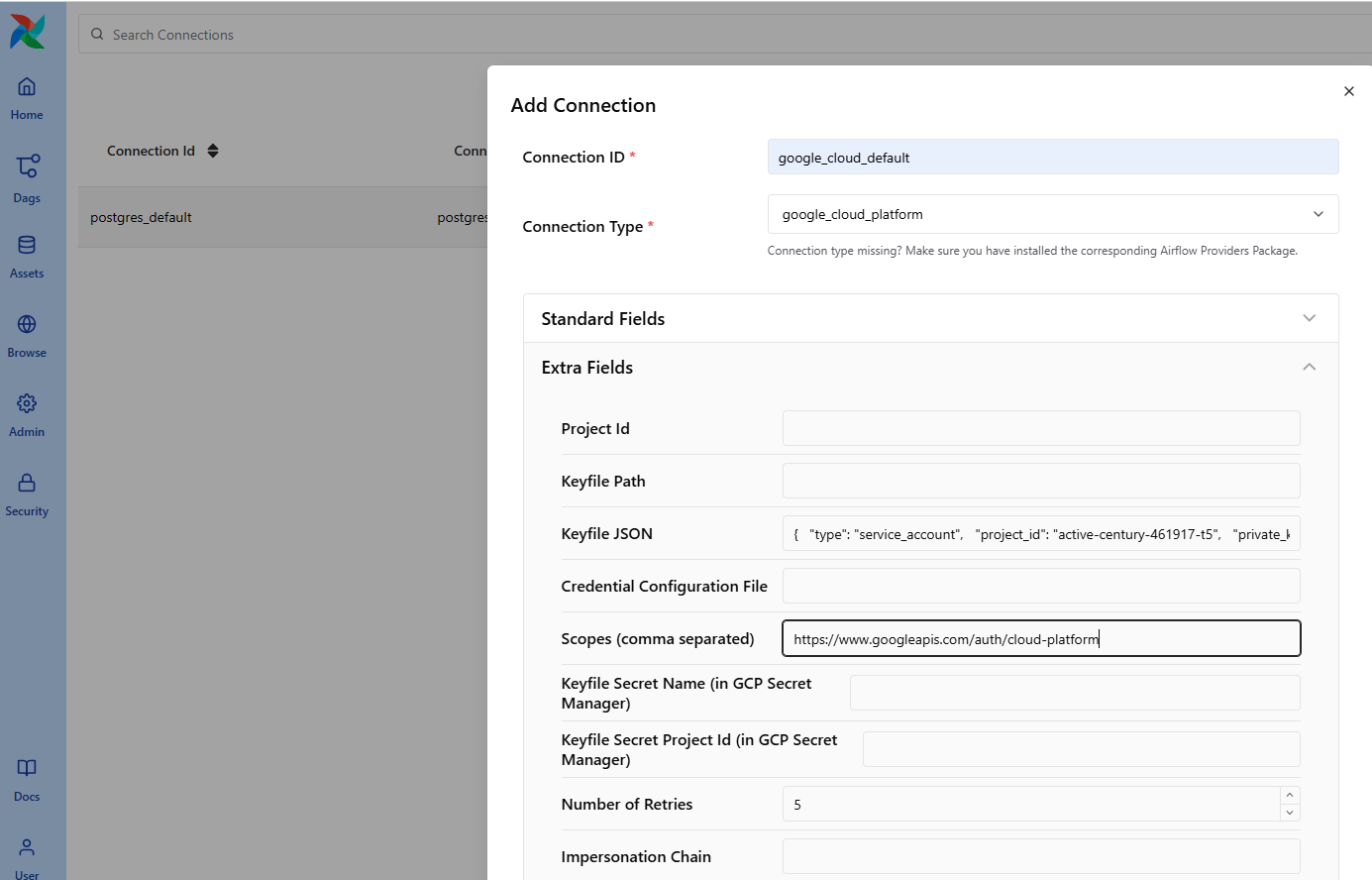
* + 1. *Postgres*

[*https://www.udemy.com/course/mastering-advanced-mlops-on-gcp-cicd-kubernetes-kubeflow/learn/lecture/48793611#questions*](https://www.udemy.com/course/mastering-advanced-mlops-on-gcp-cicd-kubernetes-kubeflow/learn/lecture/48793611#questions)





*GOOGLE CLOUD GCP*





*GIT*

If you have a folder on your local computer and you want to start using Git to manage its version history and potentially collaborate, here's a step-by-step guide:

**1. Install Git**

If you don't already have Git installed, you'll need to do that first.

* **Windows:** Download Git for Windows from <https://git-scm.com/download/win> and follow the installation instructions.
* **macOS:** Git is often pre-installed. You can check by opening Terminal and typing git --version. If not, you can install it via Homebrew (brew install git) or by installing Xcode Command Line Tools (xcode-select --install).
* **Linux:** Use your distribution's package manager (e.g., sudo apt-get install git for Debian/Ubuntu, sudo yum install git for Fedora/RHEL).

**2. Initialize a Git Repository in Your Folder**

Navigate to your existing folder using your command line (Terminal on macOS/Linux, Git Bash or Command Prompt/PowerShell on Windows).

Bash

cd /path/to/your/folder

Once inside your folder, initialize a new Git repository:

Bash

git init

This command creates a hidden .git subdirectory inside your folder. This is where Git stores all the history, commits, and configurations for your project. Your project is now a Git repository.

**3. Configure Git (First-Time Setup)**

If this is your first time using Git on this computer, you should configure your username and email address. This information will be associated with your commits.

Bash

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

(Replace "Your Name" and "your.email@example.com" with your actual name and email).

**4. Check the Status of Your Files**

Now that your folder is a Git repository, you can see the status of your files:

Bash

git status

This will show you which files are untracked (new files that Git isn't managing yet) and which have been modified.

**5. Stage Your Files (Add to Staging Area)**

Before you can save changes (make a "commit"), you need to tell Git which changes you want to include in the next commit. This is called "staging" your files.

To stage all current changes (new files, modified files):

Bash

git add .

If you only want to stage specific files:

Bash

git add file1.txt folder/file2.js

After git add, run git status again. You'll see the staged files listed.

**6. Commit Your Changes (Save Version History)**

A commit is a snapshot of your staged files at a specific point in time. Every commit has a unique ID and a commit message that describes the changes.

Bash

git commit -m "Initial commit"

The -m flag is for the commit message. Make it descriptive so you and others can understand what changes were made in this commit.

**7. Continue Your Work**

As you make more changes to your files:

* Modify files.
* Save them.
* Run git status to see what changed.
* git add . (or specific files) to stage the changes you want to commit.
* git commit -m "Descriptive message about what I changed" to save the new snapshot.

**8. (Optional) Connect to a Remote Repository (GitHub, GitLab, Bitbucket)**

If you want to back up your code online, collaborate with others, or work on multiple computers, you'll connect your local Git repository to a remote one.

* **Create a new repository on your chosen Git hosting service** (e.g., GitHub). Do **not** initialize it with a README or license if your local folder already has content, as it will create merge conflicts.
* Once created, the hosting service will give you commands like these:

Bash

git remote add origin https://github.com/your-username/your-repo-name.git

git branch -M main # (Optional: renames your default branch to 'main' if it's 'master')

git push -u origin main # Pushes your local 'main' branch to the 'origin' remote

Replace the URL with your actual remote repository URL. The -u flag sets the upstream branch, so future git push and git pull commands are simpler.

Now your local folder is a Git repository, its changes are tracked, and you can push them to a remote server for backup and collaboration!