Coursework 2: Setting Up a Simple ML Pipeline

Big Data Engineering

(Report)

Stefan Velev, 0MI3400521

Introduction

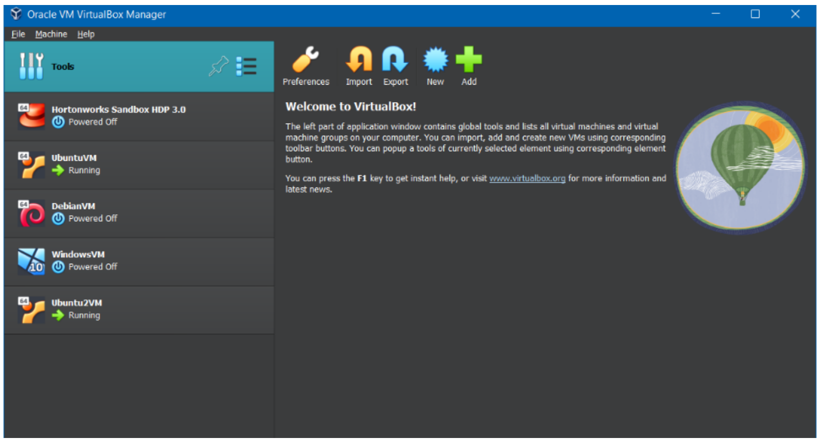
This coursework assignment covers the process of setting up a simple Machine Learning (ML) pipeline – from ingesting raw data to presenting the analysed data in a Google Drive spreadsheet. This entails setting up virtual machines, installing Docker environment, spinning up containers, performing data ingestion, and utilizing cloud services for data storage and presentation.

Step 1: Setting up 2 Ubuntu Virtual Machines

I have set up two virtual machines in Ubuntu using Oracle Virtual Box 7.0.14. I have downloaded the images from www.osboxes.org/ubuntu. The roles of the VMs are as follows:

VM1: *Ubuntu 24.10 Oracular Oriole* – run the entire pipeline infrastructure (Docker Compose with Airflow, MLflow, MongoDB, PostgreSQL)

VM2: *Ubuntu 23.10 Mantic Minotaur* – store the CSV file (employers.csv) and carry out the transfer to VM1 with the help of Samba (SMB protocol)

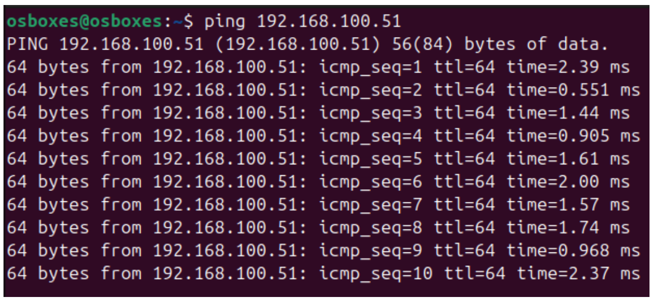


Welcome to VirtualBox!

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.

We have to ensure that both virtual machines can communicate with each other. For that purpose, they have to be in one network. This can be configured from the settings of each virtual machine: **Settings → Network → Attached to Bridged Adapter**. For my case, the relevant IP addresses of the VMs are: 192.168.100.48/24 and 192.168.100.51/24. We have to check that we can ping successfully from one to the other.

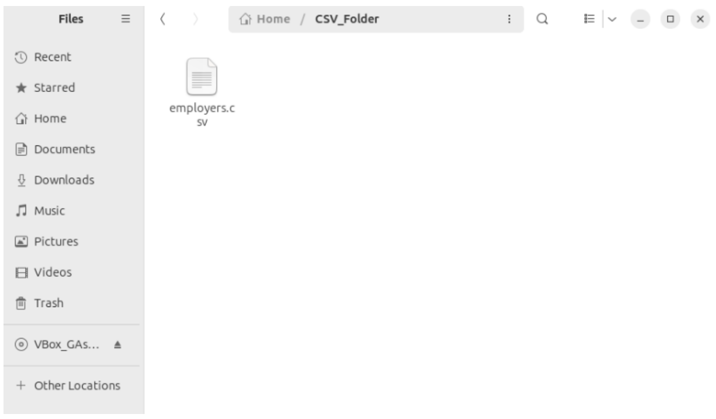


```
osboxes@osboxes:~$ ping 192.168.100.51
PING 192.168.100.51 (192.168.100.51) 56(84) bytes of data.
64 bytes from 192.168.100.51: icmp_seq=1 ttl=64 time=2.39 ms
64 bytes from 192.168.100.51: icmp_seq=2 ttl=64 time=0.551 ms
64 bytes from 192.168.100.51: icmp_seq=3 ttl=64 time=1.44 ms
64 bytes from 192.168.100.51: icmp_seq=4 ttl=64 time=0.905 ms
64 bytes from 192.168.100.51: icmp_seq=5 ttl=64 time=1.61 ms
64 bytes from 192.168.100.51: icmp_seq=6 ttl=64 time=2.00 ms
64 bytes from 192.168.100.51: icmp_seq=7 ttl=64 time=1.57 ms
64 bytes from 192.168.100.51: icmp_seq=8 ttl=64 time=1.74 ms
64 bytes from 192.168.100.51: icmp_seq=9 ttl=64 time=0.968 ms
64 bytes from 192.168.100.51: icmp_seq=10 ttl=64 time=2.37 ms
```

Step 2: Create CSV file and Configure VM2 for Communication with VM1

My first step is to create the CSV file that I will transfer with the help of the pipeline that will be constructed. I have chosen to use the structure of the sample CSV dataset presented in the coursework instructions. The file is called **employers.csv** and starts as follows:

```
id,name,salary,department
1,john,2000,sales
2,andrew,5000,finance
3,mark,8000,hr
4,rey,5000,marketing
5,tan,9000,it
…
```



The next step is related to the transfer of the file between the two virtual machines. I have decided to use **Samba** for that communication. In this way, I will be able to read the CSV file from VM1 as if it were a local file. I have done the installation and configuration with the following commands:

```
sudo apt install samba
mkdir CSV_Folder
cd ./CSV_Folder
touch employers.csv
```

For the configuration I have added the block below in the Samba config file **/etc/samba/smb.conf**:

```
[csvshare]
   path = /home/osboxes/CSV_Folder
   browseable = yes
   read only = no
   guest ok = yes
   force user = osboxes
```

Then I restarted the Samba service and gave proper permissions so that everyone can read:

```
sudo systemctl restart smbd
chmod 777 ~/CSV_Folder
```

On VM1 I installed **CIFS utilities** with the following command:

```
sudo apt install cifs-utils
```

The final step was to mount the shared folder in VM1. In order to connect to VM2's Samba share, I auto-mounted it on boot for persistent share by editing **/etc/fstab** and adding the following configuration at the end:

```
//192.168.100.51/csvshare /mnt/vm2share cifs guest 0 0
```

Now on every start of the machines, I have the file in **/mnt/vm2share** and every time I modify it on VM2, it gets updated on VM1 immediately. This is important for the next steps because we want automatic update in the pipeline whenever new data is ingested.

Step 3: Install Docker & Docker Compose on VM1

VM1 hosts my Docker environment, so I had to install **Docker Engine** and **Docker Compose**. For that purpose, I have followed the official documentation of **Docker**:

https://docs.docker.com/engine/install/ubuntu/ (for Docker Engine)
https://docs.docker.com/compose/install/linux/ (for Docker Compose)

Step 4: Develop a Custom Docker Compose File to Spin Up Containers

On VM1 I had to coin **docker-compose.yml** file that starts up **Apache Airflow** for workflow orchestration, runs up **MLflow** for experiment tracking, spins up **MongoDB** for data persistence, and ensures all containers are networked properly. For that purpose, I have relied on the provided **Docker** files in the Moodle course for **Airflow** and **MLflow** and complemented them with configuration for **MongoDB**.

The final version of the **docker-compose.yml** file is:

```
version: '3.8'

services:
  postgres:
    image: postgres:13
    container_name: postgres
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U airflow"]
      interval: 5s
      retries: 5
    networks:
      - mlnet

  airflow-webserver:
    build: .
    image: custom-airflow:latest
    container_name: airflow-webserver
    environment:
      - AIRFLOW__CORE__EXECUTOR=LocalExecutor
      - AIRFLOW__CORE__FERNET_KEY=rqBLDe7ftXkYvPLKjwdkCR42Up-LVteBKtO_5DQfUww=
      - AIRFLOW__DATABASE__SQL_ALCHEMY_CONN=postgresql+psycopg2://airflow:airflow@postgres:5432/airflow
      - AIRFLOW__CORE__DAGS_FOLDER=/opt/airflow/dags
      - _AIRFLOW_WWW_USER_USERNAME=admin
      - _AIRFLOW_WWW_USER_PASSWORD=admin
    depends_on:
      postgres:
        condition: service_healthy
    volumes:
      - ./airflow:/opt/airflow
      - ./dags:/opt/airflow/dags
      - /mnt/vm2share:/opt/airflow/data
      - ./secrets:/opt/airflow/secrets
    ports:
      - "8080:8080"
    command: bash -c "airflow db init && airflow users create \
                      --username $${_AIRFLOW_WWW_USER_USERNAME} \
                      --password $${_AIRFLOW_WWW_USER_PASSWORD} \
                      --firstname Admin \
                      --lastname User \
                      --role Admin \
                      --email admin@example.com && \
                      airflow webserver"
    networks:
      - mlnet

  airflow-scheduler:
    build: .
    image: custom-airflow:latest
    container_name: airflow-scheduler
    depends_on:
      postgres:
        condition: service_healthy
    environment:
      - AIRFLOW__CORE__EXECUTOR=LocalExecutor
      - AIRFLOW__CORE__FERNET_KEY=rqBLDe7ftXkYvPLKjwdkCR42Up-LVteBKtO_5DQfUww=
      - AIRFLOW__DATABASE__SQL_ALCHEMY_CONN=postgresql+psycopg2://airflow:airflow@postgres:5432/airflow
    volumes:
      - ./airflow:/opt/airflow
      - ./dags:/opt/airflow/dags
      - /mnt/vm2share:/opt/airflow/data
      - ./secrets:/opt/airflow/secrets
    command: airflow scheduler
    networks:
      - mlnet

  mlflow:
    image: ghcr.io/mlflow/mlflow:v2.4.1
    container_name: mlflow
    ports:
      - "5000:5000"
    command: mlflow server --backend-store-uri sqlite:///mlflow.db --default-artifact-root
/mlflow/artifacts --host 0.0.0.0
    volumes:
      - ./mlruns:/mlflow/artifacts
    networks:
      - mlnet

  mongo:
    image: mongo:4.4
    container_name: mongodb
    restart: always
    ports:
      - "27017:27017"
    volumes:
      - mongodata:/data/db
    networks:
      - mlnet

volumes:
  postgres_data:
  mongodata:

networks:
  mlnet:
```

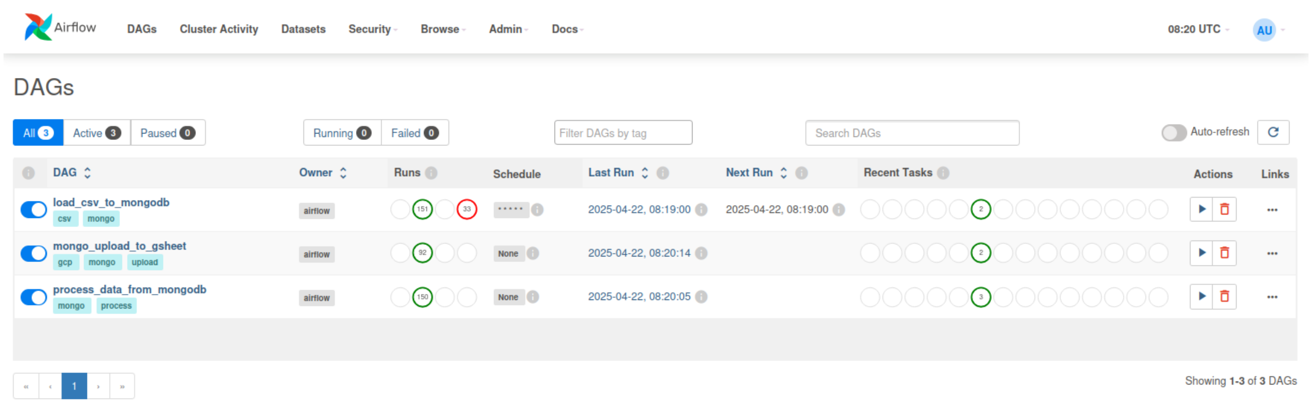
The preliminary requirements are to have directories named **airflow**, **dags**, **secrets**, and **mlruns**. Moreover, we have to ensure that we have **/mnt/vm2share** with the CSV file from the previous steps. In order to have proper communication, we have to define shared Docker network that will be the same for all the containers.

For the installation of the relevant packages and libraries we need **Dockerfile** which in my solution is with the following content:

```
FROM apache/airflow:2.9.1

USER airflow

RUN pip install --no-cache-dir pymongo apache-airflow-providers-mongo gspread google-auth mlflow
```

When we want to start the **Docker** environment we have to use the following commands:

```
sudo docker-compose build
sudo docker-compose -f ./docker-compose.yml up
```



DAGs

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
load_csv_to_mongodb csv mongo	airflow	151 33	* * * * *	2025-04-22, 08:19:00	2025-04-22, 08:19:00	2	▶ 🗑	···
mongo_upload_to_gsheet gcp mongo upload	airflow	92	None	2025-04-22, 08:20:14		2	▶ 🗑	···
process_data_from_mongodb mongo process	airflow	150	None	2025-04-22, 08:20:05		3	▶ 🗑	···

« ‹ 1 › »

Showing **1-3** of **3** DAGs

Step 5: Create DAG: mongo_ingest_dag.py

In the created directory **dags** we have to put the **Python** code of the DAGs that we want to load in **Airflow**. The first step is the creation of the ingest DAG – for extracting the CSV data and loading it in **MongoDB**. With the help of **Pandas**, we read it and convert it to dictionary which is the appropriate data structure for the **MongoDB** database. The connection to **MongoDB** is performed with **MongoClient** pointing to the relevant URL with the correct port that is defined from us in the docker-compose file (27017). The database that we will use is named: **"company"** and the name of the collection is **"employers"**. With the help of **insert_many** we store all the data. The DAG is scheduled to run every minute with the help of the **CRON** syntax: **"* * * * *"**. The connection to the next DAG is done with **TriggerDagRunOperator** which ensures continuation immediately after the **load_task** finishes successfully.

```python
import pandas as pd
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.trigger_dagrun import TriggerDagRunOperator
from datetime import datetime
from pymongo import MongoClient

def load_csv_to_mongo():
    df = pd.read_csv("/opt/airflow/data/employers.csv")

    df['salary'] = df['salary'].astype(int)

    records = df.to_dict(orient='records')

    # Connect to MongoDB
    client = MongoClient("mongodb://mongo:27017/")
    db = client["company"]
    collection = db["employers"]

    collection.delete_many({})

    collection.insert_many(records)
    print("Data is inserted into MongoDB")

with DAG(
    dag_id="load_csv_to_mongodb",
    start_date=datetime(2025, 4, 1),
    schedule_interval="* * * * *",
    catchup=False,
    tags=["mongo", "csv"],
) as dag:
    load_task = PythonOperator(
        task_id="load_csv",
        python_callable=load_csv_to_mongo
    )

    trigger_process_dag = TriggerDagRunOperator(
        task_id="trigger_process_dag",
        trigger_dag_id="process_data_from_mongodb"
    )

    load_task >> trigger_process_dag
```

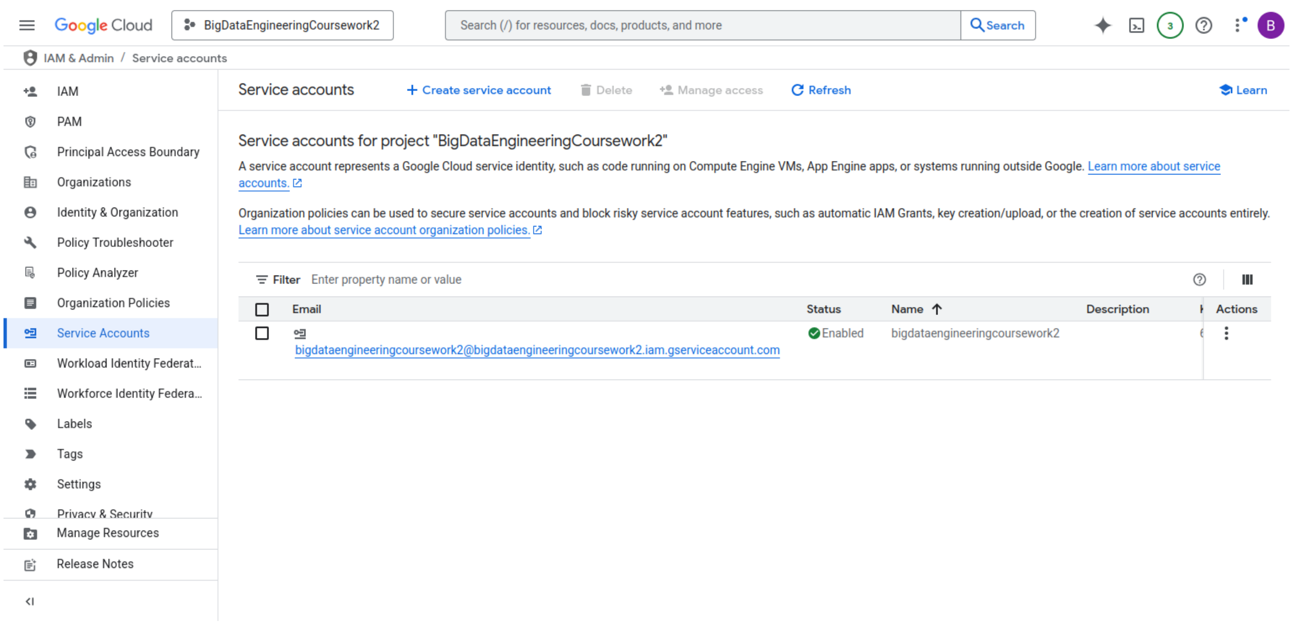
Step 6: Create DAG: mongo_process_dag.py

The next DAG that is automatically triggered after the first DAG is the one related to processing of data. Since our dataset is about workers in a company with details for their salaries and the department they belong to, I have decided to make statistics about the number of people in a department, the total salary and the average salary in the department. I store this analysed data in separate collection called **"department_stats"** in **MongoDB**. In fact, this processed data will be what I store in **Google Sheets** file in the next DAG. I trigger the next DAG again with **TriggerDagRunOperator**. We can notice that in the current DAG I have dummy task (**start_task**) that I use for proper arrangement of the tasks and easier modification if necessary, later.

```
import gspread
from google.oauth2.service_account import Credentials
from pymongo import MongoClient
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.trigger_dagrun import TriggerDagRunOperator
from airflow.operators.dummy_operator import DummyOperator
from airflow.providers.mongo.hooks.mongo import MongoHook
from datetime import datetime

def process_data_from_mongo():
    hook = MongoHook(conn_id="mongo_conn")
    client = hook.get_conn()
    db = client["company"]
    collection = db["employers"]

    data = list(collection.find())

    department_stats = {}
    for record in data:
        dept = record['department']
        salary = int(record['salary'])

        if dept not in department_stats:
            department_stats[dept] = {"total_salary": 0, "staff_count": 0}

        department_stats[dept]["total_salary"] += salary
        department_stats[dept]["staff_count"] += 1

    for dept in department_stats:
        total = department_stats[dept]["total_salary"]
        count = department_stats[dept]["staff_count"]
        avg = total / count
        department_stats[dept]["average_salary"] = avg

    print("Department-wise statistics:", department_stats)

    db["department_stats"].delete_many({})
    db["department_stats"].insert_many([
        {"department": k, **v} for k, v in department_stats.items()
    ])

with DAG(
    dag_id="process_data_from_mongodb",
    start_date=datetime(2025, 4, 1),
    schedule_interval=None,
    catchup=False,
    tags=["mongo", "process"]
) as dag:

    start_task = DummyOperator(task_id="start_task")

    process_task = PythonOperator(
        task_id="process_data",
        python_callable=process_data_from_mongo
    )

    trigger_upload_dag = TriggerDagRunOperator(
        task_id="trigger_upload_dag",
        trigger_dag_id="mongo_upload_to_gsheet"
    )


    start_task >> process_task >> trigger_upload_dag
```

Step 7: Create Google Cloud Project for data storage and presentation

For the creation of the **Google Cloud Project (GCP)** I have made a new registration in **Google Cloud** with email: bigdataengineering610@gmail.com. Then we have to go to **Google Cloud Console**. We have to create new project with the name in my case: **BigDataEngineeringCoursework2**. The next step is to enable **Google Drive API** and **Google Sheets API**. We can do that from **APIs & Services → Library**. After that we have to create a **Service Account** from **IAM & Admin → Service Accounts → Create Service Account**. In my case the name of the service account is: **bigdataengineeringcoursework2**. For roles we can select **Editor** for full access. Then in the list of service accounts we have to find our new account and click **Manage keys**. Under **Keys** we choose **Add Key → Create new key → JSON → Create**. That will download a **JSON** file that we have to save and put in the **secrets** directory that we have created in step 4.



Service accounts for project "BigDataEngineeringCoursework2"

A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or systems running outside Google. Learn more about service accounts.

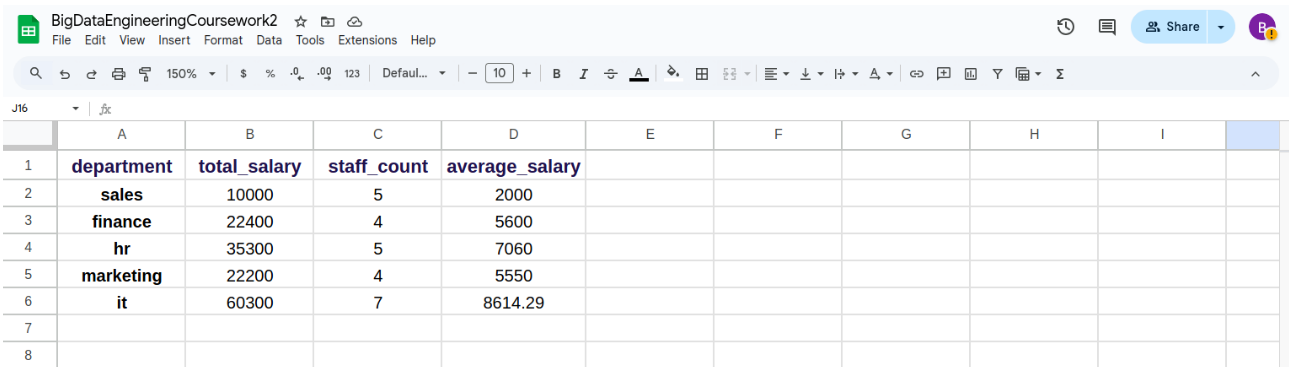
Organization policies can be used to secure service accounts and block risky service account features, such as automatic IAM Grants, key creation/upload, or the creation of service accounts entirely. Learn more about service account organization policies.

Email	Status	Name ↑	Description	Actions
bigdataengineeringcoursework2@bigdataengineeringcoursework2.iam.gserviceaccount.com	Enabled	bigdataengineeringcoursework2		⋮

As a next step we can go to **Google Sheets** and create a sheet called: **BigDataEngineeringCoursework2**. We have to copy the email address of the service account that we can find in the previously downloaded **JSON** key, and share it with that email. We do not have to forget to give editor access to the sheet. After that, our service account can write to the **Google Sheet** using the **GCP APIs**.

Step 8: Create DAG: mongo_upload_dag.py

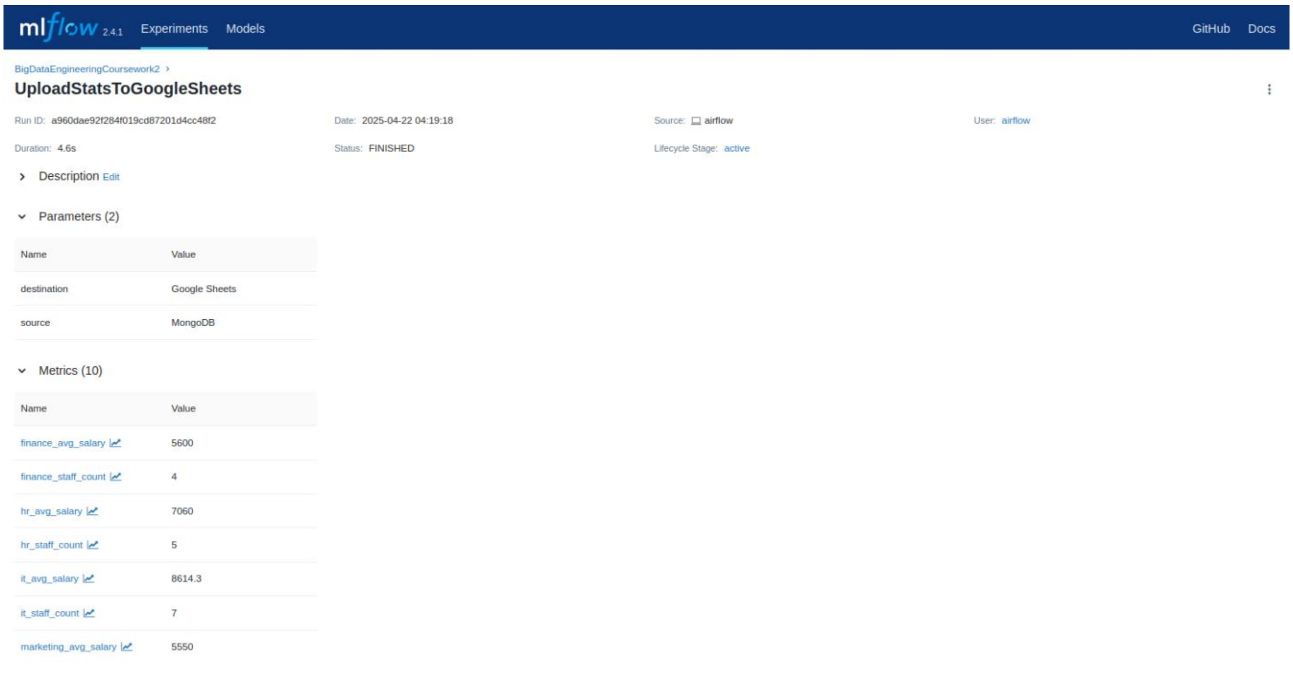
In that step we have to upload our analysed data to **Google Sheets** with the help of the configured **GCP APIs** in the previous step. We achieve that with the help of libraries like **gspread** and **google-auth** that can be installed from the command **pip install** in the **Dockerfile**. The connection in the DAG is standard and specific for the API so we will not go into details about it. We will mention that it clears existing sheet content and uploads headers and data. If we change the original CSV dataset in VM2, we can notice that the analysed data gets changed automatically after a minute.


BigDataEngineeringCoursework2
File Edit View Insert Format Data Tools Extensions Help
Share
150%
J16

department	total_salary	staff_count	average_salary
sales	10000	5	2000
finance	22400	4	5600
hr	35300	5	7060
marketing	22200	4	5550
it	60300	7	8614.29

```
import gspread
import mlflow
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.dummy import DummyOperator
from datetime import datetime
from google.oauth2.service_account import Credentials
from pymongo import MongoClient

def upload_stats_to_gsheet():
    mlflow.set_tracking_uri("http://mlflow:5000")
    mlflow.set_experiment("BigDataEngineeringCoursework2")

    with mlflow.start_run(run_name="UploadStatsToGoogleSheets"):
        mlflow.log_param("source", "MongoDB")
        mlflow.log_param("destination", "Google Sheets")

        mongo_client = MongoClient("mongodb://mongo:27017/")
        db = mongo_client["company"]
        stats = list(db["department_stats"].find({}, {"_id": 0}))

        if not stats:
            print("No statistics can be found")
            return

        for stat in stats:
            department = stat["department"]
            avg_salary = stat.get("average_salary")
            count = stat.get("staff_count")

            if avg_salary is not None:
                mlflow.log_metric(f"{department}_avg_salary", avg_salary)
            if count is not None:
                mlflow.log_metric(f"{department}_staff_count", count)

        SERVICE_ACCOUNT_FILE = '/opt/airflow/secrets/bigdataengineeringcoursework2-6782d28a1616.json'
        SCOPES = ['https://www.googleapis.com/auth/spreadsheets',
'https://www.googleapis.com/auth/drive']

        credentials = Credentials.from_service_account_file(SERVICE_ACCOUNT_FILE, scopes=SCOPES)
        client = gspread.authorize(credentials)
        sheet = client.open("BigDataEngineeringCoursework2").sheet1

        sheet.clear()
        headers = list(stats[0].keys())
        data = [headers] + [[row.get(h, "") for h in headers] for row in stats]
        sheet.update('A1', data)

        print("Data uploaded to Google Sheets and logged to MLflow.")


with DAG(
    dag_id="mongo_upload_to_gsheet",
    start_date=datetime(2025, 4, 1),
    schedule_interval=None,
    catchup=False,
    tags=["mongo", "upload", "gcp"]
) as dag:

    start = DummyOperator(task_id="start")

    upload = PythonOperator(
        task_id="upload_to_gsheet",
        python_callable=upload_stats_to_gsheet
    )

    start >> upload
```

MLflow is another tool that we can use to log statistics to when uploading to **Google Sheets**. In that way we can get visibility into department-wise statistics like average salary and number of workers for each DAG run that are suitable for traceability and reporting. In the DAG we have to set the **MLflow tracking URI** and the **MLflow experiment**. Then we log parameters in which we define the source and the destination and metrics related to our domain. We can check the metrics by opening http://localhost:5000 in the browser and exploring the experiment named **BigDataEngineeringCoursework2**.



BigDataEngineeringCoursework2 ›

UploadStatsToGoogleSheets

Run ID: a960dae92f284f019cd87201d4cc48f2
Date: 2025-04-22 04:19:18
Source: airflow
User: airflow
Duration: 4.6s
Status: FINISHED
Lifecycle Stage: active

› Description Edit

Parameters (2)

Name	Value
destination	Google Sheets
source	MongoDB

Metrics (10)

Name	Value
finance_avg_salary	5600
finance_staff_count	4
hr_avg_salary	7060
hr_staff_count	5
it_avg_salary	8614.3
it_staff_count	7
marketing_avg_salary	5550

Conclusion and Challenges Faced

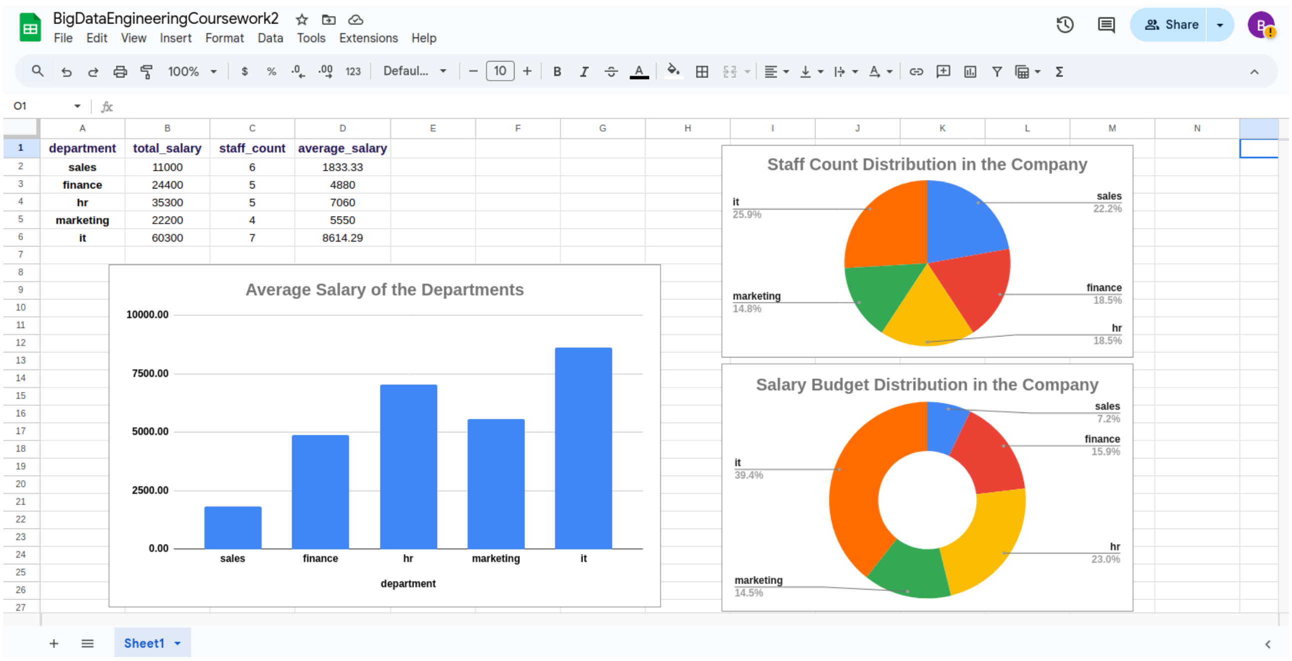
When working with different technologies a lot of technical challenges can be faced. As a whole, working on a virtual machine can be a challenge because you do not have available all the resources of the PC. One of the first difficult moments to fix was assembling the **docker-compose** file. At first, I forgot to create the preliminary directories that are required. Another challenge was the container of **Airflow** on port 8080 because it needs connection to database and the set up is not straightforward. In addition, when I am changing the **docker** files drastically, I had to remove all the containers, images and volumes so that I can ensure that the problem is not hidden in old versions. Another error that I could eventually resolve was the indentation of the **YAML** file.

Another tricky moment was with the storage of the **CSV** dataset. My first strategy was to create symlink in separate directory **data** that points to **/mnt/vm2share/employers.csv**. However, this did not work because even if I add the directory **data** as a volume for Airflow, the path to **/mnt/vm2share** is not attached in the **docker-compose** file. I had to attach it to fix the problem.

For the ingest DAG, I remember that I could not find the logs for the errors that I received. I could see the final result of the DAG as a failed run attempt but I could not find the logs in the UI. I found them as files in the working directory. Another tough moment was with the connection from **Airflow** to **MongoDB**. Fortunately, the default connection that **Airflow** provides was enough for me to connect to the database. The problem here was with the version because at first, I used **MongoDB 5.0+** that requires a **CPU** with **AVX** support. I lacked it so in order to resolve the error, I migrated to **MongoDB 4.4**. When we talk about the database, another challenge was with the verification whether the inserted data is really added. I checked for that using the client in terminal with the command **docker exec -it mongodb mongo**.

One of the final challenges was with the **WiFi** network. I started the coursework assignment with one **IP address**. However, I switched the networks because I moved to different place, and it took me a while to perceive that the new IP address is different. My task was to modify it in the mount file for persistent share.

To conclude, in this project, a complete data pipeline was implemented to automate the processing and visualization of employee data using **Apache Airflow**, **MongoDB**, **Google Sheets**, and **MLflow**. The setup included two virtual machines connected via **Samba** for **CSV** sharing, **Docker** containers for service isolation, and scheduled DAGs for orchestrating tasks from data ingestion to reporting. Despite challenges such as network reconfiguration and permission issues, the system was successfully deployed, offering a reproducible and scalable workflow for data analysis and presentation. The pipeline can be later used with different **CSV** files for different analyses making the solution universal.


BigDataEngineeringCoursework2
File Edit View Insert Format Data Tools Extensions Help
Share
Sheet1

department	total_salary	staff_count	average_salary
sales	11000	6	1833.33
finance	24400	5	4880
hr	35300	5	7060
marketing	22200	4	5550
it	60300	7	8614.29

Average Salary of the Departments
10000.00
7500.00
5000.00
2500.00
0.00
sales
finance
hr
marketing
it
department
Staff Count Distribution in the Company
it 25.9%
sales 22.2%
marketing 14.8%
finance 18.5%
hr 18.5%
Salary Budget Distribution in the Company
sales 7.2%
finance 15.9%
it 39.4%
hr 23.0%
marketing 14.5%