Name: Mrunali Katta

ID: 017516785

**Homework 9**

Github link:

1. Modify your docker-compose.yaml file to include sentence-transformers and pinecone packages (1pt)
   * Restart your docker containers ("docker compose down" and "docker compose up")
   * docker-compose.yaml should be a part of your github submission

* Modified the existing **docker-compose.yaml** file to include sentence-transformers and pinecone packages as with following line.

\_PIP\_ADDITIONAL\_REQUIREMENTS: ${\_PIP\_ADDITIONAL\_REQUIREMENTS:- yfinance apache-airflow-providers-snowflake snowflake-connector-python sentence-transformers pinecone}

"Note: pinecone was used instead of pinecone-client in \_PIP\_ADDITIONAL\_REQUIREMENTS because it successfully resolved and installed the required Pinecone SDK without errors in our setup."

Attaching the full yaml file below line.

**docker-compose.yaml**

# Licensed to the Apache Software Foundation (ASF) under one

# or more contributor license agreements.  See the NOTICE file

# distributed with this work for additional information

# regarding copyright ownership.  The ASF licenses this file

# to you under the Apache License, Version 2.0 (the

# "License"); you may not use this file except in compliance

# with the License.  You may obtain a copy of the License at

#

#   http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing,

# software distributed under the License is distributed on an

# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY

# KIND, either express or implied.  See the License for the

# specific language governing permissions and limitations

# under the License.

#

# Basic Airflow cluster configuration for CeleryExecutor with Redis and PostgreSQL.

#

# WARNING: This configuration is for local development. Do not use it in a production deployment.

#

# This configuration supports basic configuration using environment variables or an .env file

# The following variables are supported:

#

# AIRFLOW\_IMAGE\_NAME           - Docker image name used to run Airflow.

#                                Default: apache/airflow:2.9.2

# AIRFLOW\_UID                  - User ID in Airflow containers

#                                Default: 50000

# AIRFLOW\_PROJ\_DIR             - Base path to which all the files will be volumed.

#                                Default: .

# Those configurations are useful mostly in case of standalone testing/running Airflow in test/try-out mode

#

# \_AIRFLOW\_WWW\_USER\_USERNAME   - Username for the administrator account (if requested).

#                                Default: airflow

# \_AIRFLOW\_WWW\_USER\_PASSWORD   - Password for the administrator account (if requested).

#                                Default: airflow

# \_PIP\_ADDITIONAL\_REQUIREMENTS - Additional PIP requirements to add when starting all containers.

#                                Use this option ONLY for quick checks. Installing requirements at container

#                                startup is done EVERY TIME the service is started.

#                                A better way is to build a custom image or extend the official image

#                                as described in https://airflow.apache.org/docs/docker-stack/build.html.

#                                Default: ''

# Feel free to modify this file to suit your needs.

---

x-airflow-common:

  &airflow-common

  # In order to add custom dependencies or upgrade provider packages you can use your extended image.

  # Comment the image line, place your Dockerfile in the directory where you placed the docker-compose.yaml

  # and uncomment the "build" line below, Then run `docker-compose build` to build the images.

  image: ${AIRFLOW\_IMAGE\_NAME:-apache/airflow:2.10.1}

  # build: .

  environment:

    &airflow-common-env

    AIRFLOW\_\_CORE\_\_EXECUTOR: LocalExecutor

    AIRFLOW\_\_DATABASE\_\_SQL\_ALCHEMY\_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow

    AIRFLOW\_\_CORE\_\_FERNET\_KEY: ''

    AIRFLOW\_\_CORE\_\_DAGS\_ARE\_PAUSED\_AT\_CREATION: 'true'

    AIRFLOW\_\_CORE\_\_LOAD\_EXAMPLES: 'false'

    AIRFLOW\_\_API\_\_AUTH\_BACKENDS: 'airflow.api.auth.backend.basic\_auth,airflow.api.auth.backend.session'

    # yamllint disable rule:line-length

    # Use simple http server on scheduler for health checks

    # See https://airflow.apache.org/docs/apache-airflow/stable/administration-and-deployment/logging-monitoring/check-health.html#scheduler-health-check-server

    # yamllint enable rule:line-length

    AIRFLOW\_\_SCHEDULER\_\_ENABLE\_HEALTH\_CHECK: 'true'

    # WARNING: Use \_PIP\_ADDITIONAL\_REQUIREMENTS option ONLY for a quick checks

    # for other purpose (development, test and especially production usage) build/extend Airflow image.

    \_PIP\_ADDITIONAL\_REQUIREMENTS: ${\_PIP\_ADDITIONAL\_REQUIREMENTS:- yfinance apache-airflow-providers-snowflake snowflake-connector-python sentence-transformers pinecone}

    # The following line can be used to set a custom config file, stored in the local config folder

    # If you want to use it, outcomment it and replace airflow.cfg with the name of your config file

    # AIRFLOW\_CONFIG: '/opt/airflow/config/airflow.cfg'

  volumes:

    - ${AIRFLOW\_PROJ\_DIR:-.}/dags:/opt/airflow/dags

    - ${AIRFLOW\_PROJ\_DIR:-.}/logs:/opt/airflow/logs

    - ${AIRFLOW\_PROJ\_DIR:-.}/config:/opt/airflow/config

    - ${AIRFLOW\_PROJ\_DIR:-.}/plugins:/opt/airflow/plugins

  user: "${AIRFLOW\_UID:-50000}:0"

  depends\_on:

    &airflow-common-depends-on

    postgres:

      condition: service\_healthy

services:

  postgres:

    image: postgres:13

    environment:

      POSTGRES\_USER: airflow

      POSTGRES\_PASSWORD: airflow

      POSTGRES\_DB: airflow

    volumes:

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

    healthcheck:

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

      interval: 10s

      retries: 5

      start\_period: 5s

    restart: always

  airflow:

    <<: \*airflow-common

    command: >

      bash -c "airflow scheduler & airflow webserver"

    ports:

      - "8081:8080"

    healthcheck:

      test: ["CMD", "curl", "--fail", "http://localhost:8080/health"]

      interval: 30s

      timeout: 10s

      retries: 5

      start\_period: 30s

    restart: always

    depends\_on:

      <<: \*airflow-common-depends-on

      airflow-init:

        condition: service\_completed\_successfully

  airflow-init:

    <<: \*airflow-common

    entrypoint: /bin/bash

    # yamllint disable rule:line-length

    command:

      - -c

      - |

        if [[ -z "${AIRFLOW\_UID}" ]]; then

          echo

          echo -e "\033[1;33mWARNING!!!: AIRFLOW\_UID not set!\e[0m"

          echo "If you are on Linux, you SHOULD follow the instructions below to set "

          echo "AIRFLOW\_UID environment variable, otherwise files will be owned by root."

          echo "For other operating systems you can get rid of the warning with manually created .env file:"

          echo "    See: https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html#setting-the-right-airflow-user"

          echo

        fi

        one\_meg=1048576

        mem\_available=$$(($$(getconf \_PHYS\_PAGES) \* $$(getconf PAGE\_SIZE) / one\_meg))

        cpus\_available=$$(grep -cE 'cpu[0-9]+' /proc/stat)

        disk\_available=$$(df / | tail -1 | awk '{print $$4}')

        warning\_resources="false"

        if (( mem\_available < 4000 )) ; then

          echo

          echo -e "\033[1;33mWARNING!!!: Not enough memory available for Docker.\e[0m"

          echo "At least 4GB of memory required. You have $$(numfmt --to iec $$((mem\_available \* one\_meg)))"

          echo

          warning\_resources="true"

        fi

        if (( cpus\_available < 2 )); then

          echo

          echo -e "\033[1;33mWARNING!!!: Not enough CPUS available for Docker.\e[0m"

          echo "At least 2 CPUs recommended. You have $${cpus\_available}"

          echo

          warning\_resources="true"

        fi

        if (( disk\_available < one\_meg \* 10 )); then

          echo

          echo -e "\033[1;33mWARNING!!!: Not enough Disk space available for Docker.\e[0m"

          echo "At least 10 GBs recommended. You have $$(numfmt --to iec $$((disk\_available \* 1024 )))"

          echo

          warning\_resources="true"

        fi

        if [[ $${warning\_resources} == "true" ]]; then

          echo

          echo -e "\033[1;33mWARNING!!!: You have not enough resources to run Airflow (see above)!\e[0m"

          echo "Please follow the instructions to increase amount of resources available:"

          echo "   https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html#before-you-begin"

          echo

        fi

        mkdir -p /sources/logs /sources/dags /sources/plugins

        chown -R "${AIRFLOW\_UID}:0" /sources/{logs,dags,plugins}

        exec /entrypoint airflow version

    # yamllint enable rule:line-length

    environment:

      <<: \*airflow-common-env

      \_AIRFLOW\_DB\_MIGRATE: 'true'

      \_AIRFLOW\_WWW\_USER\_CREATE: 'true'

      \_AIRFLOW\_WWW\_USER\_USERNAME: ${\_AIRFLOW\_WWW\_USER\_USERNAME:-airflow}

      \_AIRFLOW\_WWW\_USER\_PASSWORD: ${\_AIRFLOW\_WWW\_USER\_PASSWORD:-airflow}

      \_PIP\_ADDITIONAL\_REQUIREMENTS: ''

    user: "0:0"

    volumes:

      - ${AIRFLOW\_PROJ\_DIR:-.}:/sources

  airflow-cli:

    <<: \*airflow-common

    profiles:

      - debug

    environment:

      <<: \*airflow-common-env

      CONNECTION\_CHECK\_MAX\_COUNT: "0"

    # Workaround for entrypoint issue. See: https://github.com/apache/airflow/issues/16252

    command:

      - bash

      - -c

      - airflow

volumes:

  postgres-db-volume:

**assignment8.py**

from airflow import DAG

from datetime import datetime, timedelta

from airflow.decorators import task

from airflow.models import Variable

import pandas as pd

import time

import requests

import os

from sentence\_transformers import SentenceTransformer

from pinecone import Pinecone, ServerlessSpec

# Default arguments for DAG

default\_args = {

    'owner': 'airflow',

    'depends\_on\_past': False,

    'email\_on\_failure': False,

    'email\_on\_retry': False,

    'retries': 1,

    'retry\_delay': timedelta(minutes=5),

}

with DAG(

    dag\_id='Medium\_to\_pinecone\_pipeline',

    default\_args=default\_args,

    description='Build a Medium Posting Search Engine using Pinecone',

    schedule\_interval=timedelta(days=7),

    start\_date=datetime(2025, 4, 1),

    catchup=False,

    tags=['medium', 'pinecone', 'search-engine'],

) as dag:

    """

    DAG to build a Medium article search engine using Pinecone vector database

    """

    @task

    def download\_data():

        """Download Medium dataset using requests"""

        # Create data directory if it doesn't exist

        data\_dir = '/tmp/medium\_data'

        os.makedirs(data\_dir, exist\_ok=True)

        # File path to save data

        file\_path = f"{data\_dir}/medium\_data.csv"

        # Download the data using requests

        url = 'https://s3-geospatial.s3.us-west-2.amazonaws.com/medium\_data.csv'

        response = requests.get(url)

        # Check if the request was successful

        if response.status\_code == 200:

            with open(file\_path, 'wb') as f:

                for chunk in response.iter\_content(chunk\_size=8192):

                    f.write(chunk)

            # Count lines to verify

            with open(file\_path, 'r') as f:

                line\_count = sum(1 for \_ in f)

            print(f"Downloaded file has {line\_count} lines")

        else:

            raise Exception(f"Failed to download data: HTTP Status {response.status\_code}")

        return file\_path

    @task

    def preprocess\_data(data\_path):

        """Clean and prepare data for embedding"""

        # Load data

        df = pd.read\_csv(data\_path)

        # Clean up

        df['title'] = df['title'].astype(str).fillna('')

        df['subtitle'] = df['subtitle'].astype(str).fillna('')

        # Create metadata

        df['metadata'] = df.apply(lambda row: {'title': row['title'] + " " + row['subtitle']}, axis=1)

        # Add ID

        df['id'] = df.reset\_index(drop='index').index.astype(str)

        # Save preprocessed data

        preprocessed\_path = '/tmp/medium\_data/medium\_preprocessed.csv'

        df.to\_csv(preprocessed\_path, index=False)

        print(f"Preprocessed data saved to {preprocessed\_path}")

        return preprocessed\_path

    @task

    def create\_pinecone\_index():

        """Create or reset Pinecone index"""

        # Get Pinecone API key from Airflow Variables

        api\_key = Variable.get("pinecone\_api\_key")

        # Initialize Pinecone

        pc = Pinecone(api\_key=api\_key)

        # Serverless spec for Pinecone

        spec = ServerlessSpec(

            cloud="aws",

            region="us-east-1"

        )

        index\_name = 'semantic-search-fast'

        # Check if index already exists and delete it

        existing\_indexes = [index\_info["name"] for index\_info in pc.list\_indexes()]

        if index\_name in existing\_indexes:

            pc.delete\_index(index\_name)

        # Create new index

        pc.create\_index(

            index\_name,

            dimension=384,  # dimensionality of minilm

            metric='dotproduct',

            spec=spec

        )

        # Wait for index to be initialized

        while not pc.describe\_index(index\_name).status['ready']:

            time.sleep(1)

        print(f"Pinecone index '{index\_name}' created successfully")

        return index\_name

    @task

    def generate\_embeddings\_and\_upsert(data\_path, index\_name):

        """Generate embeddings and upsert to Pinecone"""

        # Get API key

        api\_key = Variable.get("pinecone\_api\_key")

        # Load preprocessed data

        df = pd.read\_csv(data\_path)

        # Initialize embedding model

        model = SentenceTransformer('all-MiniLM-L6-v2', device='cpu')

        # Generate embeddings

        print("Generating embeddings...")

        batch\_size = 100

        total\_batches = (len(df) + batch\_size - 1) // batch\_size

        # Connect to Pinecone

        pc = Pinecone(api\_key=api\_key)

        index = pc.Index(index\_name)

        # Process in batches to avoid memory issues

        for i in range(0, len(df), batch\_size):

            print(f"Processing batch {i//batch\_size + 1}/{total\_batches}")

            batch\_df = df.iloc[i:i+batch\_size].copy()

            # Extract metadata entries

            metadata\_list = batch\_df['metadata'].apply(eval).tolist()

            # Generate embeddings for this batch

            titles = [meta['title'] for meta in metadata\_list]

            embeddings = model.encode(titles)

            # Prepare upsert data

            upsert\_data = []

            for j, (\_, row) in enumerate(batch\_df.iterrows()):

                upsert\_data.append({

                    'id': str(row['id']),

                    'values': embeddings[j].tolist(),

                    'metadata': metadata\_list[j]

                })

            # Upsert to Pinecone

            index.upsert(upsert\_data)

        print(f"Successfully upserted {len(df)} records to Pinecone")

        return index\_name

    @task

    def test\_search\_query(index\_name):

        """Test the search with a sample query"""

        # Get API key

        api\_key = Variable.get("pinecone\_api\_key")

        # Initialize embedding model

        model = SentenceTransformer('all-MiniLM-L6-v2', device='cpu')

        # Connect to Pinecone

        pc = Pinecone(api\_key=api\_key)

        index = pc.Index(index\_name)

        # Sample query

        query = "what is ethics in AI"

        query\_embedding = model.encode(query).tolist()

        # Search

        results = index.query(

            vector=query\_embedding,

            top\_k=5,

            include\_metadata=True

        )

        # Print results

        print(f"Search results for query: '{query}'")

        for result in results['matches']:

            print(f"ID: {result['id']}, Score: {result['score']}, Title: {result['metadata']['title'][:50]}...")

    # Define task dependencies using the TaskFlow API

    data\_path = download\_data()

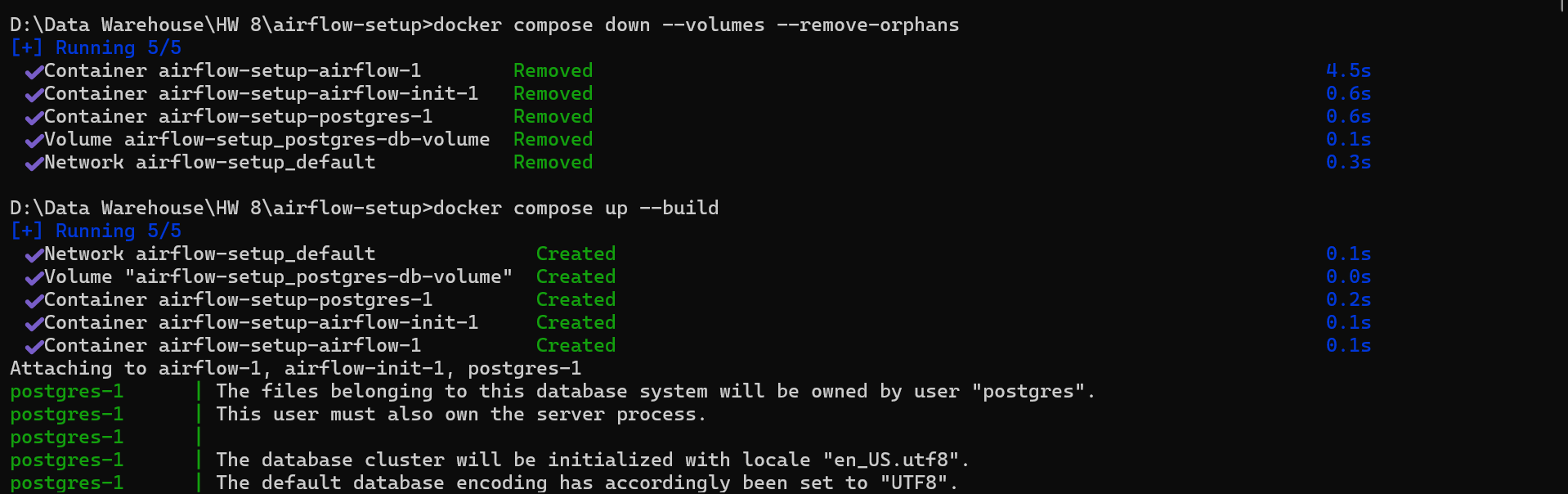
    preprocessed\_path = preprocess\_data(data\_path)

    print(preprocessed\_path)

    index\_name = create\_pinecone\_index()

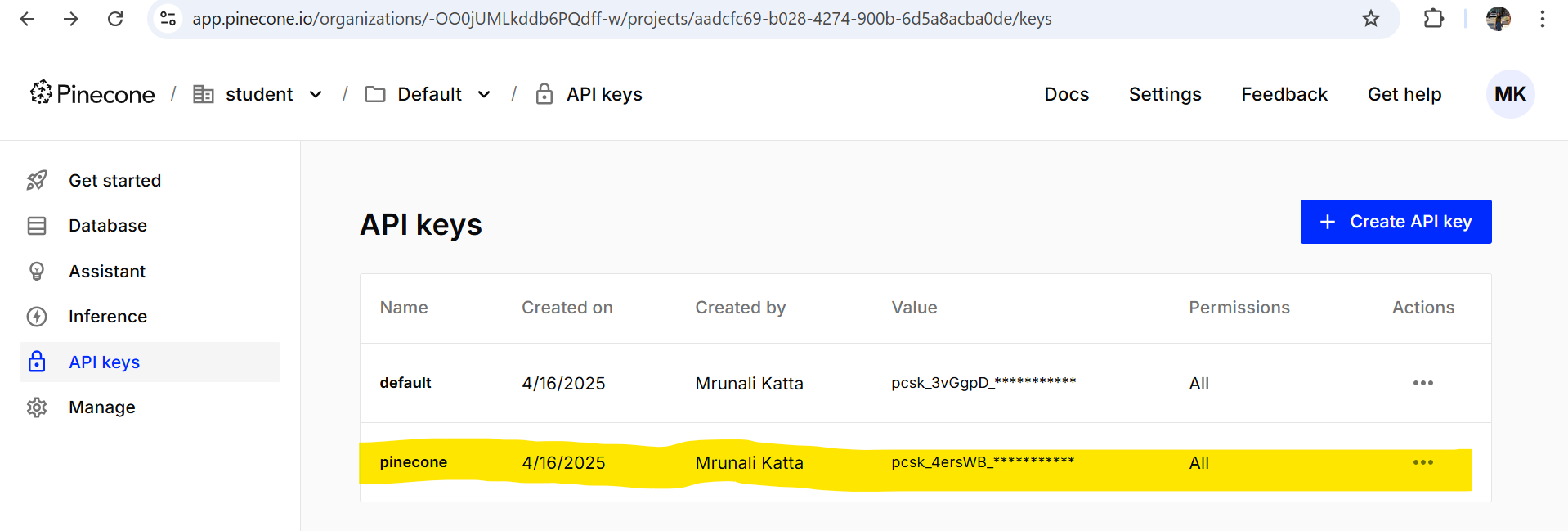
    final\_index\_name = generate\_embeddings\_and\_upsert(preprocessed\_path, index\_name)

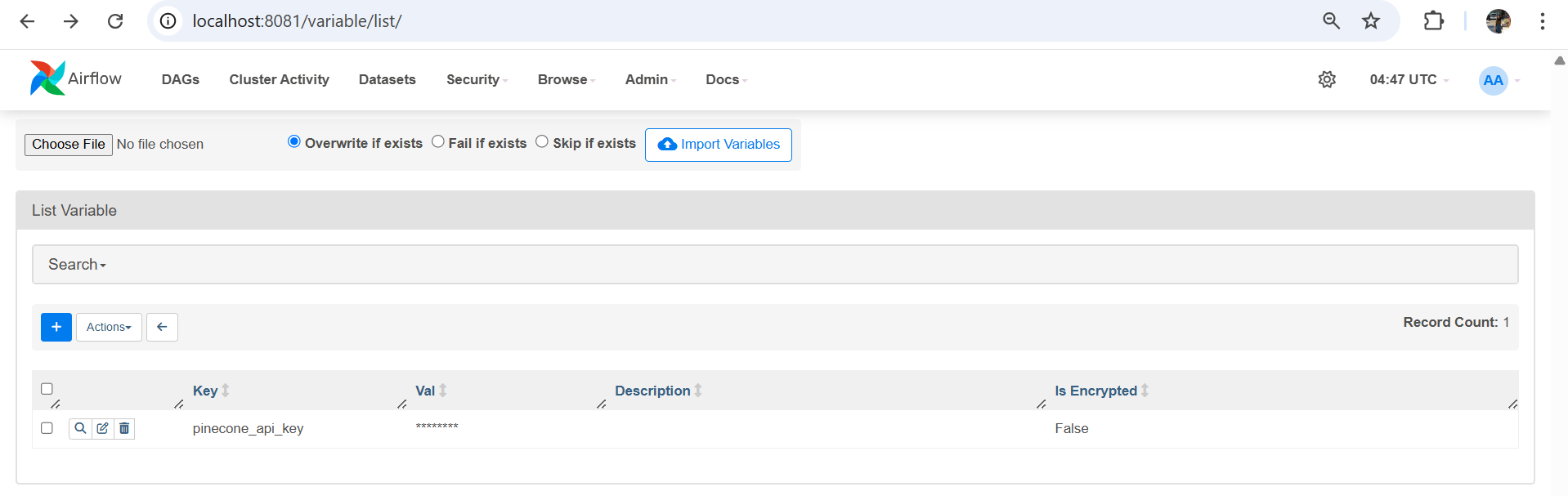
    test\_search\_query(final\_index\_name)

restarting the docker

1. Configure Pinecone (account), get the API token and create Airflow Variable (1pt)

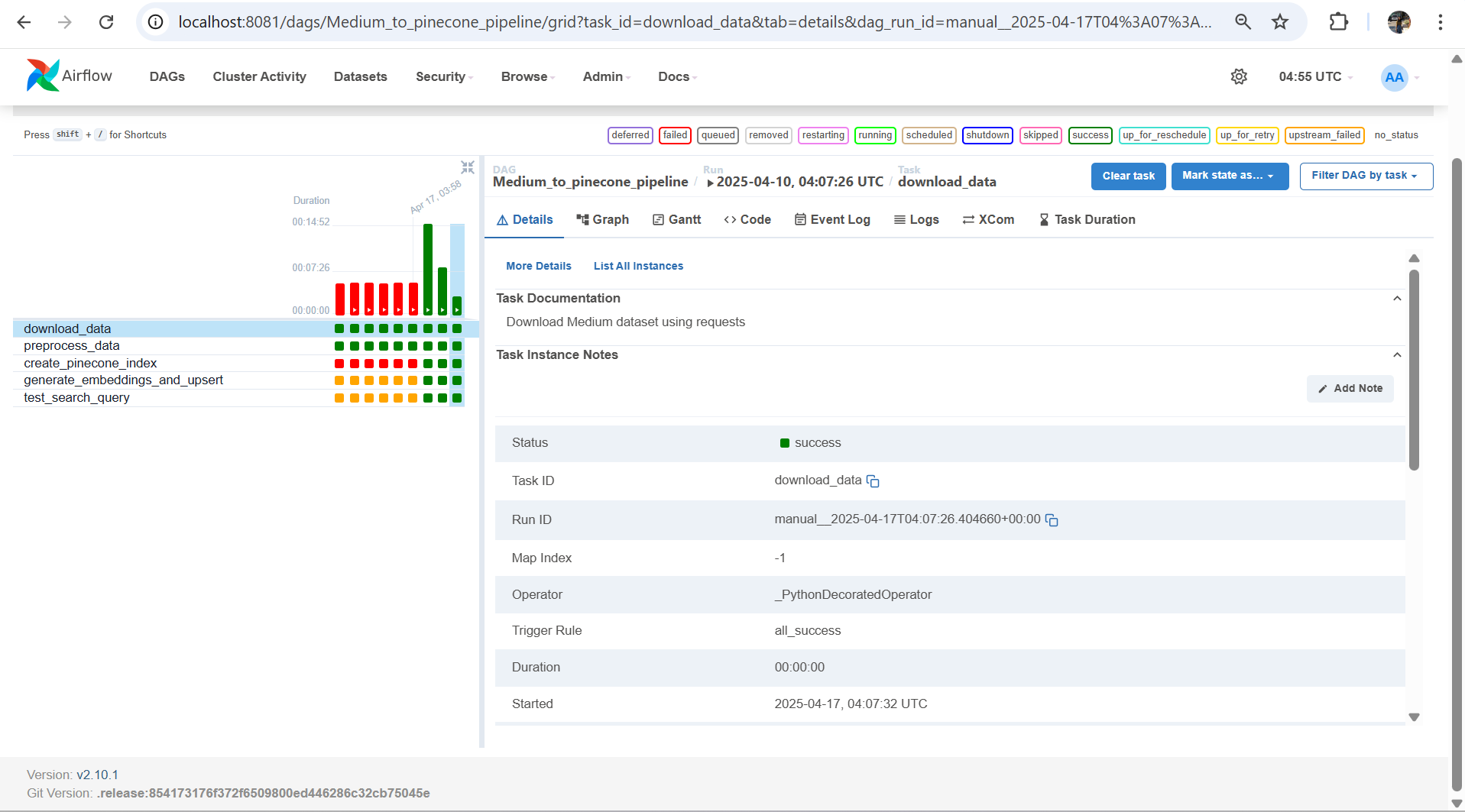
* Created Pinecone account to get the API token and added the same in the Airflow variables

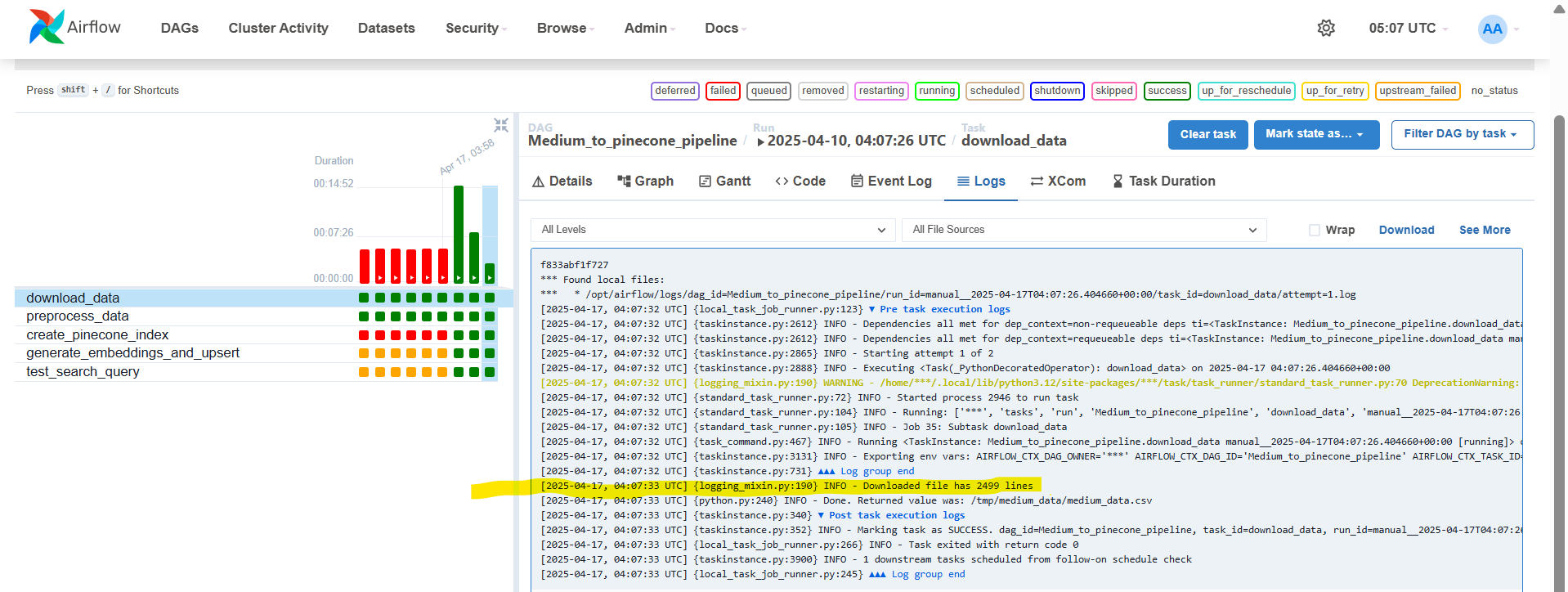




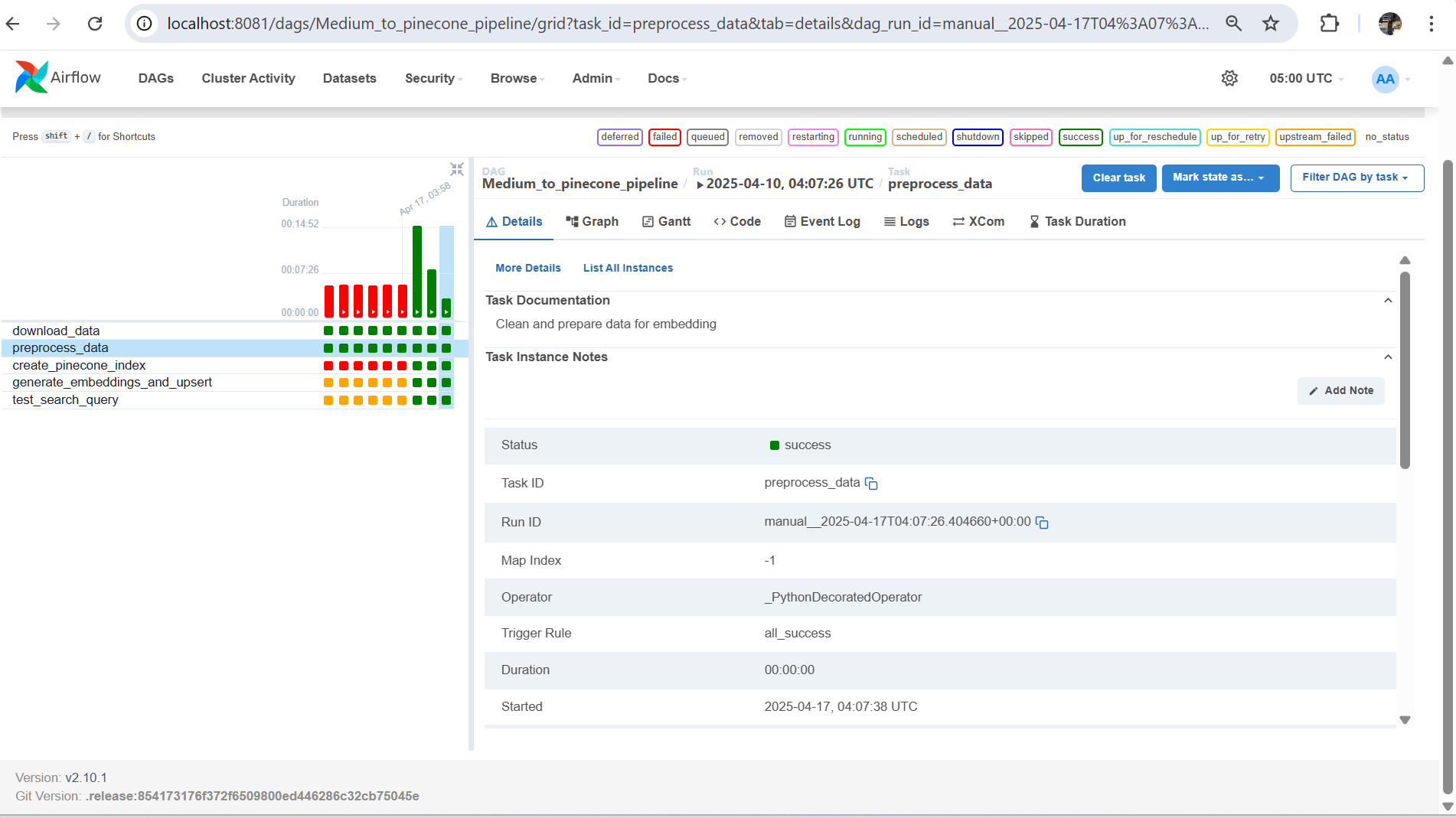
1. Download, Process and Generate an input file to Pinecone (2pt)

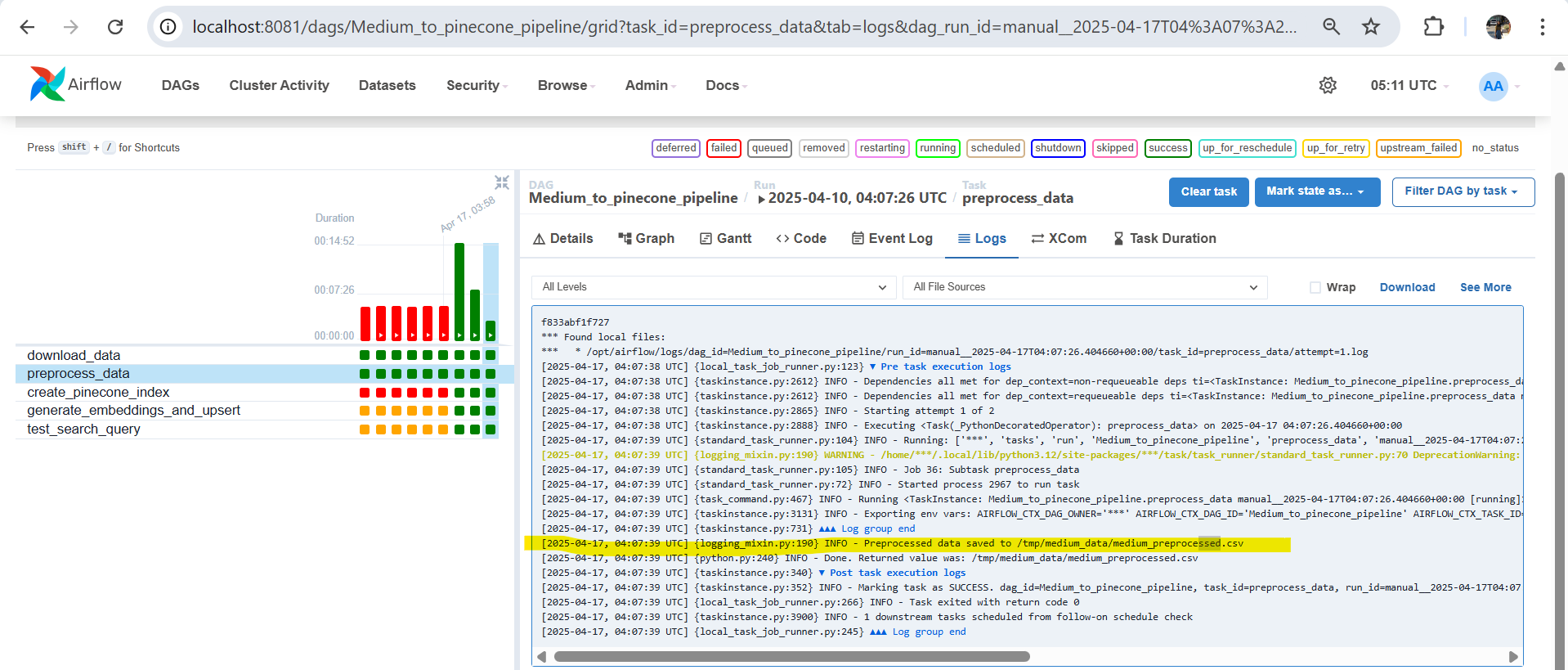
* The dataset was downloaded and saved. Line count was printed to confirm download.



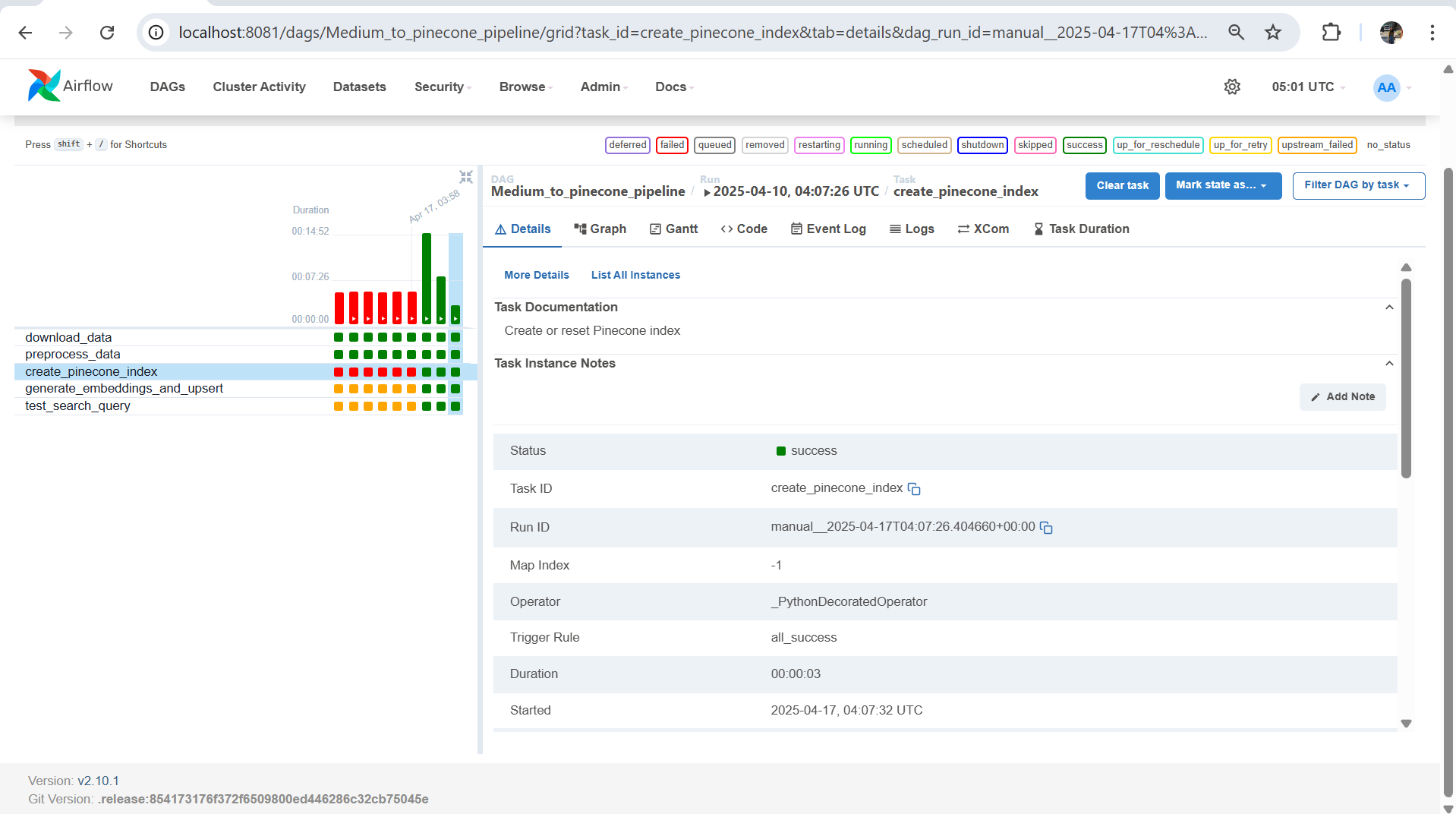


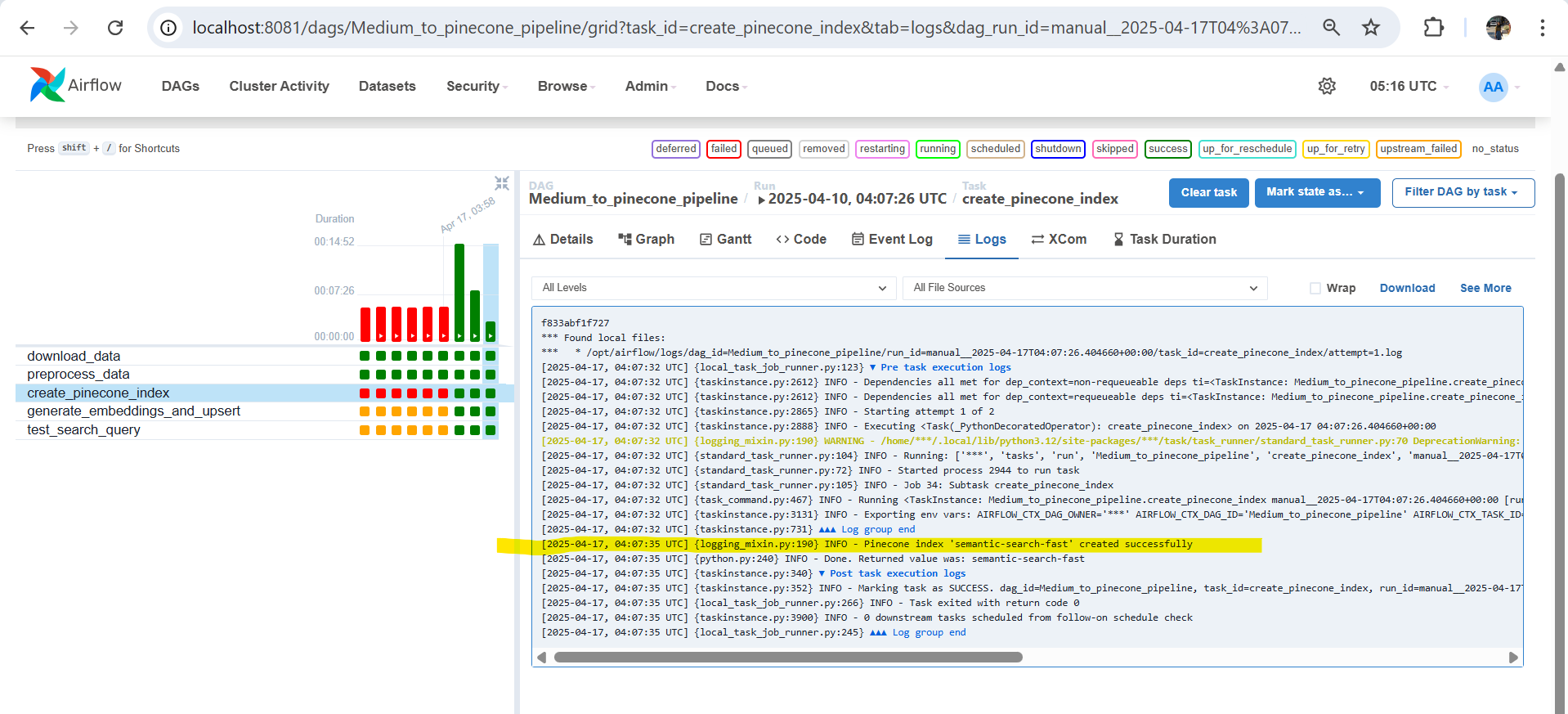
The preprocess\_data() task cleans the downloaded Medium CSV file. The preprocessed data is saved to /tmp/medium\_data/medium\_preprocessed.csv.





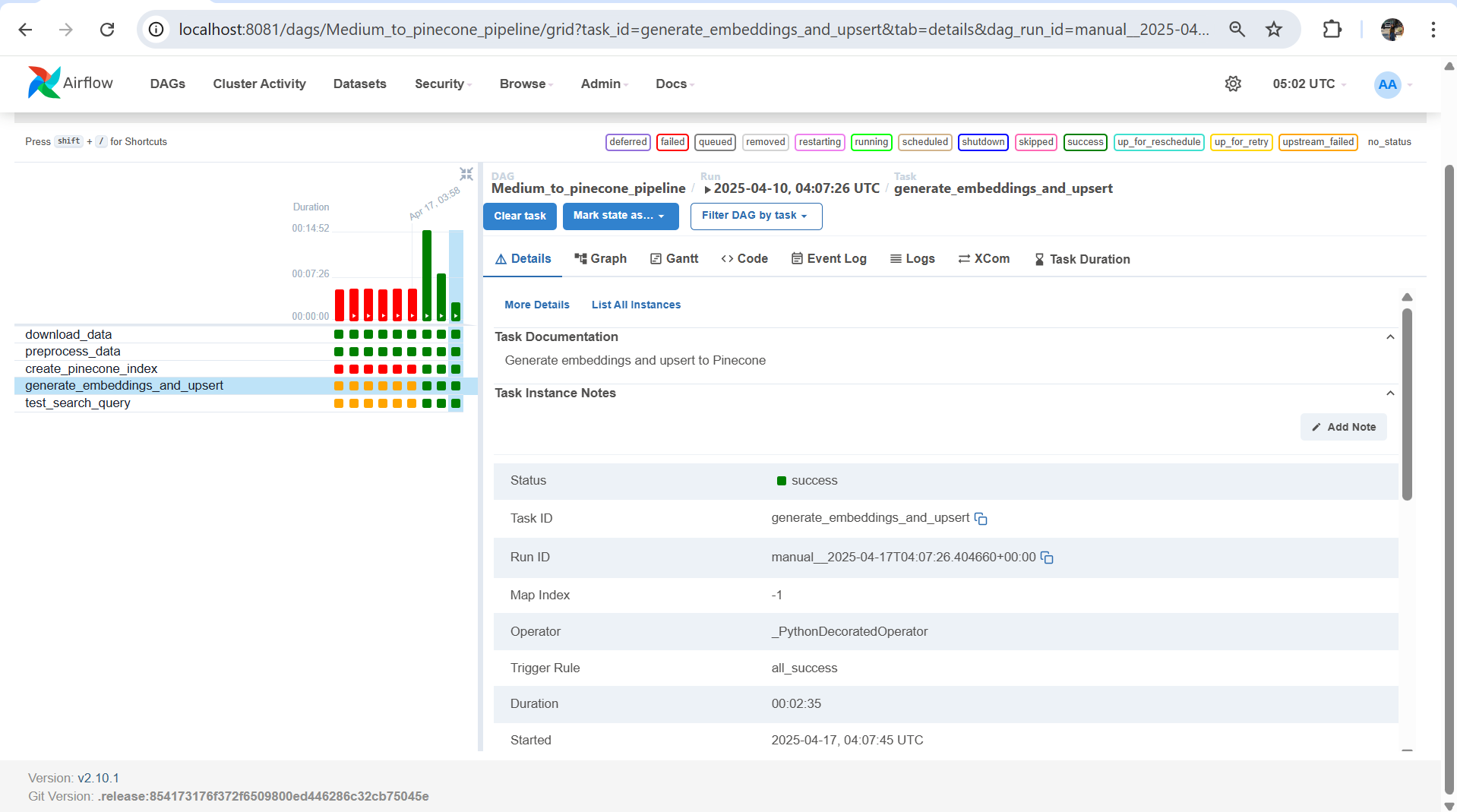
1. Create Pinecone index (1tp)

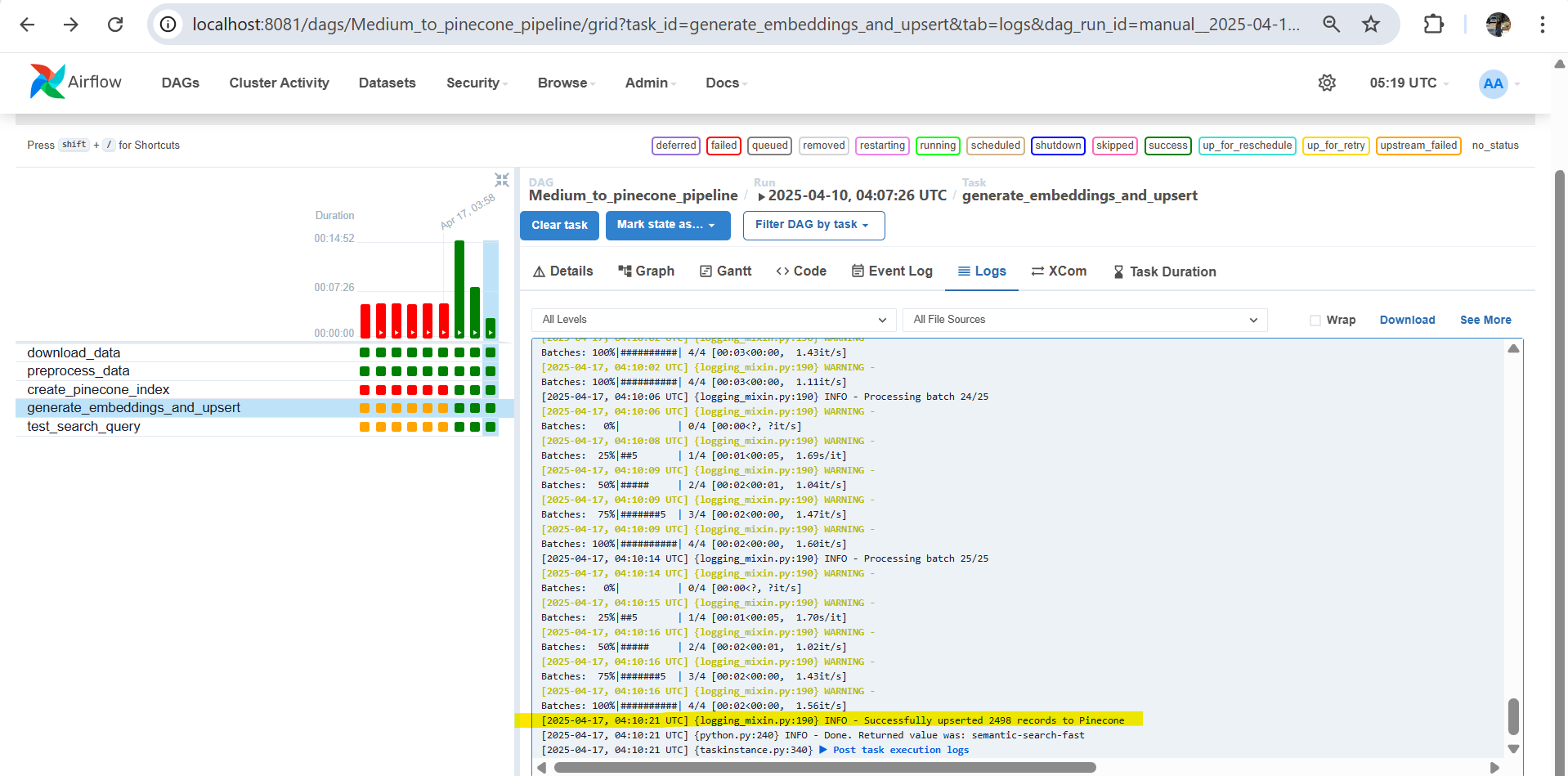
* The create\_pinecone\_index() task connects to Pinecone using an API key stored as an Airflow Variable. 



1. Convert the input file into embeddings and ingest them into Pinecone (2pt)

* The generate\_embeddings\_and\_upsert() task loads the cleaned Medium dataset from sentence-transformers to generate vector embeddings. This enables efficient semantic search over the article data.





1. Run search against Pinecone (1pt)

* The test\_search\_query() task simulates a user search experience by embedding a query string ("what is ethics in AI") into a dense vector. It then performs a similarity search on the semantic-search-fast index in Pinecone and retrieves the top 5 most relevant articles, displaying their titles and similarity scores.

