

Apache Airflow Implementation for Instagram Data Collection

GitHub Repository URL: github.com/COMP4651-24fall/project-apache-airflow-ig-data-collection

By TSANG Po Yin, YEUNG Kwong Wo, Cheng Ho Chak

Introduction

Overview

This project is designed to explore cloud computing technologies by implementing a data collection pipeline using Apache Airflow, deployed on an Azure Virtual Machine (VM). The system is integrated with Azure Cosmos DB for MongoDB and Azure Blob Storage to manage and store data. The primary focus is collecting fashion-related data from public Instagram accounts, including profile bios, images, captions, and other relevant information.

Apache Airflow allows us to design and orchestrate the data collection pipeline actions with clear dependencies and relationships between tasks as directed acyclic graphs (DAGs). This approach will enable us to efficiently organize, schedule, and activate the pipeline, ensuring a streamlined process for managing data workflows.

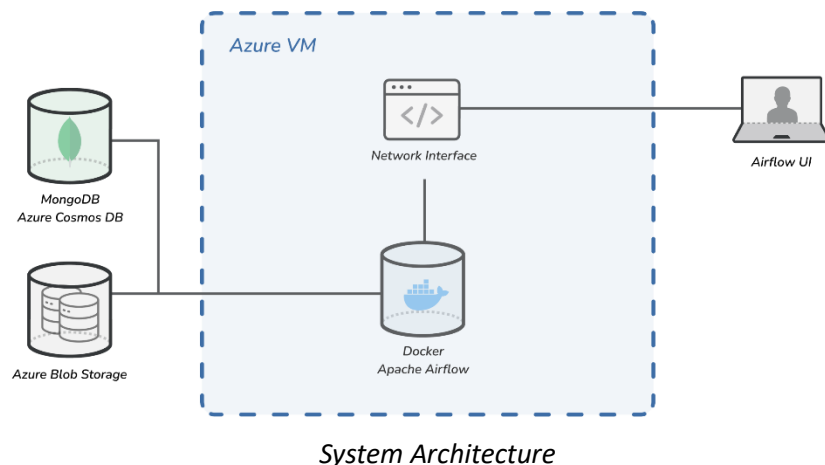
Objectives

1. **Cloud Integration:** Utilize Azure services to deploy and manage a scalable data scraping pipeline.
2. **Data Collection:** Scrape and store fashion-related data from Instagram for further analysis.
3. **Data Storage:** Use MongoDB and Blob Storage for efficient data management and retrieval.

Methodology

System Design

The system is designed to be modular and scalable, leveraging cloud services to handle large volumes of data efficiently. All the resources are created within a resource group and are deployed in East Asia (as the primary location). The architecture consists of the following components:































1. **Azure Virtual Machine:**

Hosts the Airflow instance as docker, providing the computational resources needed for the project. The source image for the VM is Linux, ubuntu-24_04-lts. For stable hosting for the Docker application, we chose Standard B2ms deployment, with 2 vcpus and 8 GiB memory. To allow internet connectivity, a network interface is configured. The public IP allows users to access the Airflow UI for seamless management and control.

In the Airflow, 2 worker containers are configured, where each worker can handle 4 tasks. I.e., 8 scripts can be executed in parallel.

2. **Azure Cosmos DB (MongoDB):** The DB stores two collections: “IGaccounts” stores the target accounts, and “IGposts” stores the collected data. As an experimental usage, we chose free tier as the cluster tier and set the storage size as 32 GiB.

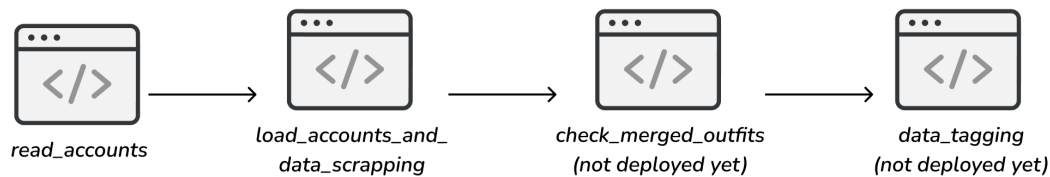
IGaccounts		IGposts	
 _id	objectid	 _id	objectid
 createdAt	isodate	 account_name	string
 dress_attire	array	 post_date	isodate
 occasion	array	 shortcode	string
 region	string	 createdAt	isodate
 style	array	 imageURLs	array
 vibe	array	 updatedAt	isodate
 __v	int32	 video_view_count	string
 fashion_lv	int32	 caption	string
 type	string	 isOutfit	string
 account_name	string	 reelURL	string
 updatedAt	isodate	 isOutfitReason	string
 gender	string	 like_count	int32
		 media_count	int32
		 merged_same_outfit	boolean

Schema of the two collections

3. **Azure Blob Storage:** Used for storing unstructured data, such as images and videos downloaded from Instagram. Hierarchical namespace is enabled, where images are stored in folders named by the account name. To enable later usages on data tagging workflows, we chose “Hot” as the access tier. We also enabled replication as read-access geo-redundant storage (RA-GRS). It will copy the data synchronously three times in the primary region (East Asia), and three times to a secondary region (Southeast Asia). With read-access enabled, the data allows read-access even during a fail-over process.

Implementation

Data Workflow (Apache Airflow DAG)



1. Read Accounts

The `read_accounts` stage is the initial step in the data processing pipeline. It is responsible for retrieving a list of Instagram accounts from a MongoDB database. This list is then saved to a CSV file, which serves as the input for subsequent stages. By using a Jupyter notebook executed with Papermill, this task allows for dynamic updates to the account list without requiring changes to the codebase. This flexibility ensures that the pipeline can easily adapt to new data sources as needed.

2. Load Accounts and Data Scrapping

The `load_accounts_and_data_scrapping` stage is the core of the pipeline, where the actual data collection takes place. This task utilizes the `instaloader` library to extract profile information, posts, and media from Instagram accounts. The collected data, including images and metadata, is stored in Azure Blob Storage and MongoDB for further processing. The task is dynamically created for each account, enabling efficient parallel execution and scalability. Various techniques are implemented to allow effective collection, including recording the “`short_code`” as a unique identifier for each post, and using “`is_completely_fetched`” to annotate whether all the images within one post is fetched successfully.

3. Check Merged Outfits (Not Deployed Yet)

As the motivation of the project is to collect fashion data for later processing, the `check_merged_outfits` stage enables concatenating the records that several images are referring to the same clothing item. It is planned to analyze the scraped data to identify and merge similar outfits using vision-enabled LLMs. While the script is completed, it is not deployed in the workflow due to cost considerations.

4. Data Tagging (Not Deployed Yet)

Same as above, the `data_tagging` stage will tag each item’s properties and descriptions for later analytic tasks. It leverages vision-enabled LLMs to perform tagging. It is not deployed in the workflow as well due to cost considerations.

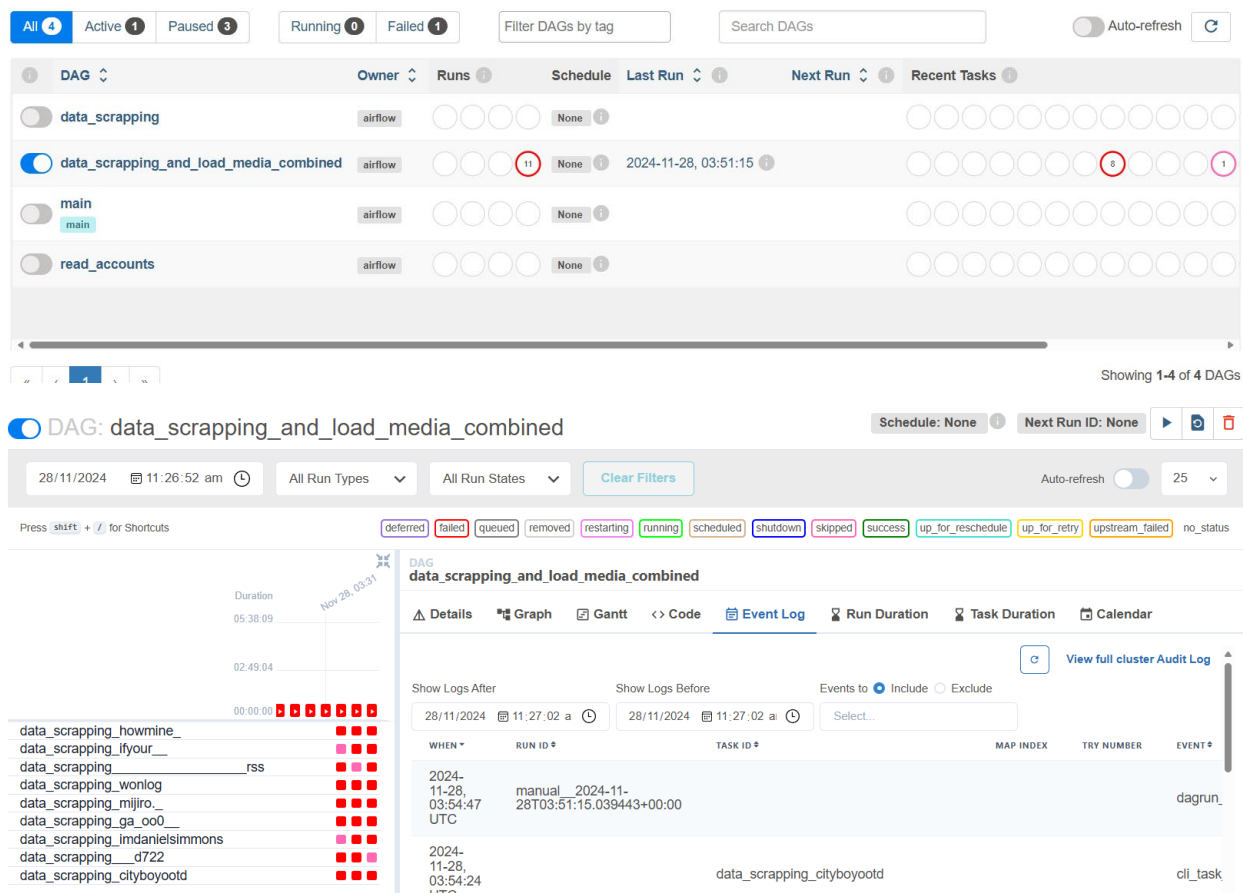
Deployment as Docker

Dockerizing Apache Airflow and deploying it on an Azure Virtual Machine (VM) provides a robust and efficient solution for managing and orchestrating data workflows. By using Docker, we ensure that the Airflow environment is consistent across different stages of development and deployment, eliminating the common “it works on my machine” issues. The Docker setup involves creating a Dockerfile to define the environment and a `docker-compose.yaml` file to manage the multi-container application, which includes components like the Airflow webserver, scheduler, workers, and a message broker like Redis.

This setup allows for easy scaling and resource management, as containers can be quickly started, stopped, or replicated as needed.

Deploying this Dockerized setup on an Azure VM offers flexibility in choosing the operating system and configuration, while also providing seamless integration with other Azure services such as Cosmos DB and Blob Storage. The VM acts as a host for the Docker containers, and once the setup is complete, the Airflow webserver can be accessed via the VM's public IP, providing a user-friendly interface for monitoring and managing workflows. This approach not only accelerates deployment but also enhances portability and resource efficiency, as containers share the same OS kernel, making them more lightweight compared to traditional virtual machines. Overall, this deployment strategy leverages the strengths of both Docker and Azure, resulting in a scalable, reliable, and easy-to-manage data processing pipeline.

DAGs



*Airflow Web UI, access with
stylematch-airflow.eastasia.cloudapp.azure.com:8080*

By now, we have collected 9 accounts, 2548 posts, 12725 images, at around 8 posts per minute while respecting the rate limit imposed by Instagram.

Future implementations

As the project transitions from a proof-of-concept to a real-world application, implementing scalable infrastructure becomes crucial to handle increased data collection and tagging tasks effectively. One of the primary strategies is to enable auto-scaling for Azure Virtual Machines. This approach allows the system to dynamically adjust the number of VM instances based on the current workload, ensuring that it can efficiently manage peak loads without incurring unnecessary costs during periods of low demand. By automatically scaling resources up or down, the system maintains optimal performance and cost-effectiveness, which is essential for a scalable and sustainable operation.

In addition to scalable infrastructure, optimizing data management and storage is vital for handling large datasets efficiently. Implementing data partitioning strategies in Azure Cosmos DB can significantly enhance query performance and manageability. By organizing data into partitions based on account names or other relevant attributes, the system can improve data retrieval speeds and ensure that the database remains responsive even as the volume of data grows. Furthermore, exploring cost-effective storage options, such as utilizing Azure Blob Storage's "Cool" or "Archive" tiers for infrequently accessed data, can help reduce storage costs while maintaining data availability. These tiers offer a more economical solution for storing data that does not require frequent access, thereby optimizing the overall cost structure of the data management system. Together, these strategies provide a robust framework for scaling the project to meet the demands of real-world applications.

Conclusion

This project successfully demonstrates the integration of cloud computing technologies to build a scalable and efficient data scraping pipeline. By leveraging Azure services, the system can handle large volumes of data. This setup not only serves the project's objectives but also provides a valuable learning platform for students to explore cloud-based data processing and management techniques.