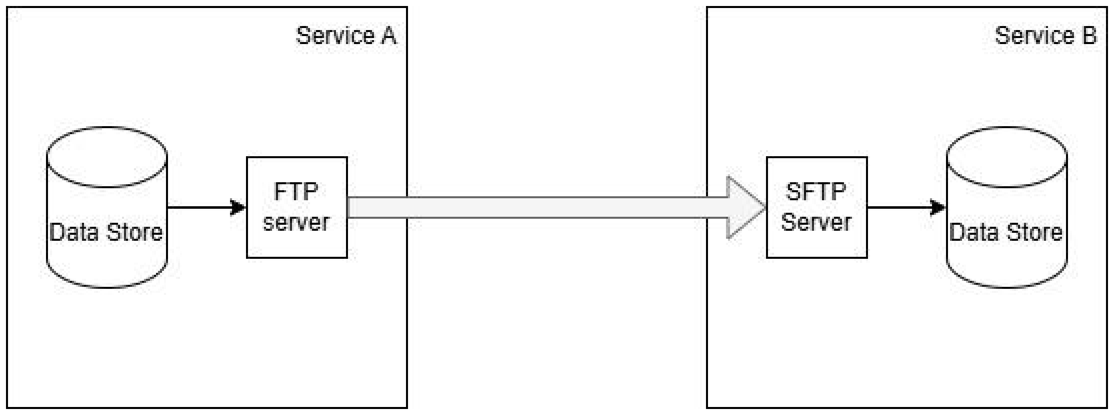
1. Context

In real-life scenarios, secure file transfers are typically managed by a centralized Managed File Transfer (MFT) system. MFT solutions consist of centralized servers and agents installed on each source and destination. They usually provide features related to data security, availability, scalability, recovery, integration, and monitoring. Due to time constraints, this assignment is designed to partially simulate file transfer using SFTP in an assumed scenario: a one-directional transfer from a client (or server) to SFTP servers, as illustrated in the diagram below.



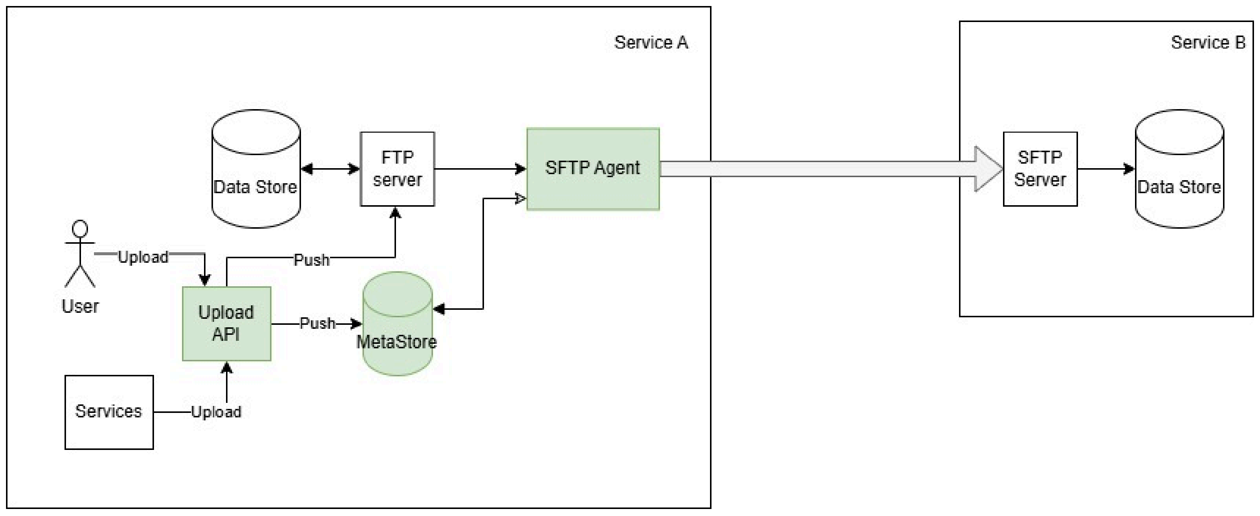
2. Assumed requirements:

- Transfer files from service A to service B using a secure file transfer protocol (SFTP).
- Have mechanism to detect new file in service A
- Provide a mechanism to monitor and manage the status of file transfers.
- Support file sizes up to 1 GB.
- Allow lightweight transformations to be applied before transferring files.
- Integrate with an Apache Airflow DAG
- Handle abnormal files

What is not included :

- The availability of the server and client during the transfer process.
- The scalability of the client prior to initiating the transfer.
- The connection between the data store and ftp/sftp server.
- Other aspects related to security, recovery, integration, and monitoring that are outside the scope of the SFTP connection between the client and the SFTP server.

3. Solution:



3.1 Summary

To manage files which are stored in sftp server in service A also to manage status of file transfering, I propose a **Metastore** which can be backed by file storage or sql database. Whenever a user or service wants to upload a file to sftp server A, they have to use the provided **upload API** . The upload api upload file to the SFTP in service A and also push the metadata about the files to MetaStore which can be used later in **SFTP Agent** to detect new file need to deliver to service B. The SFTP Agent acts as a middleware layer, it has a feature which is **uploading file in chunks** to handle big file. Additionally it has to update status of the file upload to the Metastore for monitoring purposes. Finally SFTP Agent **can be attached to a DAG** to meet requirements.

3.2 Metastore

Metastore consists of 2 tables: file_registry, transfer_logs.

File_registry:
Consist information about a file that upload to sftp server(in service A). this information will be inserted by the Upload API along with the uploading process. Those information will be used later in sftp agent(dag) to select files to transfer to service B

Column Name	Data Type	Description
id	string	Uuid of a file in db
path	string	Path of a file that uploaded to sftp server

updated_date	date	Updated or inserted date of a file
checksum	string	Checksum of a file for validation

Transfer_logs:
Which contain information of the transfer process for monitoring. It will be inserted by SFTP Agent after transfering data

Column Name	Data Type	Description
id	string	Uuid of a file in File_registry
transfer_status	string	successful/failed/in_progress
transfer_date	date	Date time after file are transferred successfully to the destination

3.3 SFTP Agent

The service job that have responsibility to transfer data from service A to service B. This service can be attached to a DAG for better intergration and monitoring. It also be designed to attach transform code before pushing

Logical flow:

1. On scheduled time the agent query the File_registry db to get list file that uploaded from the previous day
2. The file must not exists in the transfer_logs table with status successful or in_progress
3. If it include transformation code, then it re-calculate checksum for each file
4. Then the agent make a **control file** that contain list file to be upload and checksum of each file in json format
5. It insert status in_progress to table transfer_logs along with other metadata
6. Then it pushes each data file in chunks to handle memory overload when the file is big.
7. After transferring all data file, then it push a control file to predetermined destination
8. The control file help destination server to verify if it recieve enough and correct data or not
9. If no error happen and the transfer was completed it update status in table transfer_log to successful
10. If an error happens then it inserts failed status .

3.4 Upload API

Whenever a user/service wants to upload data to the service A then they have to use the upload api. The upload api calculates checksum of the files and other metadata to file_registry along with uploading file to service A.

4 Implementation

Due to time limitation, the code repo is made only to demonstate for the idea of this design. I use files storage as a backed storage for the MetaStore and assuming File_registry already contains a list of files and checksum needed to push. The Upload API **is not implemented** in this repo

5 Expansion

This implementation has some problems that need to improve:

1. Race condition if multiple agent query to the meta db may cause data duplication in the service B
2. Lack of retry or mitigation action (like rollback ..) if upload process is failed
3. Lack of scaling mechanism when the file is big
4. Lack of verification process to handle abnormal file
5. …