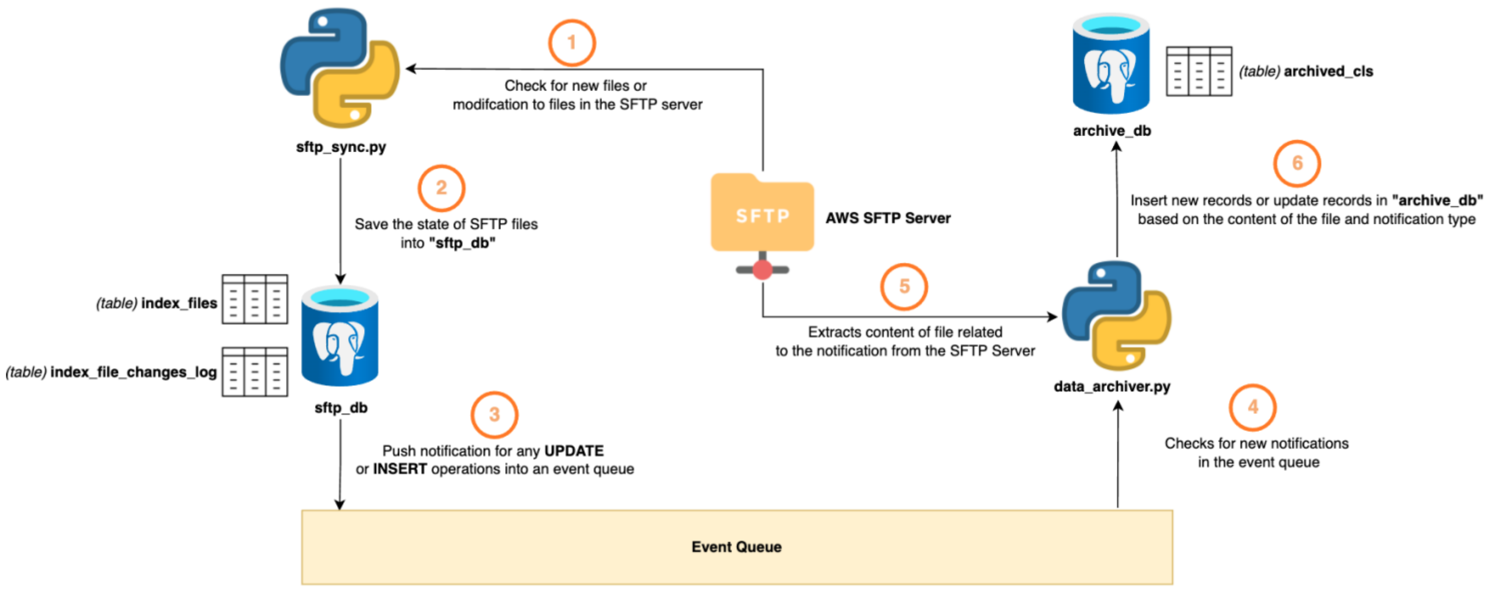
Project Implementation Design

Table of Contents

Overview

The diagram below showcases the high-level design overview of the solution implemented for this project.


sftp_sync.py
1
Check for new files or modifcation to files in the SFTP server
2
Save the state of SFTP files into "sftp_db"
(table) index_files
(table) index_file_changes_log
sftp_db
3
Push notification for any UPDATE or INSERT operations into an event queue
SFTP
AWS SFTP Server
5
Extracts content of file related to the notification from the SFTP Server
archive_db
(table) archived_cls
6
Insert new records or update records in "archive_db" based on the content of the file and notification type
data_archiver.py
4
Checks for new notifications in the event queue
Event Queue

The order of operation in the diagram is denoted with **orange numbers enclosed in orange circles** where **1** is the first operation that will execute and **6** is the last operation that will happen in the workflow.

The whole architecture can be broken down into 3 parts:

- sftp program:
  - **“sftp_sync.py”**:
    this program will keep checking for new “**ASX300_NCS_CLS”** files or modification to these files in the SFTP server at regular intervals
  - **“sftp_db”:**
    contains **index_files** table which stores the state of the SFTP server files of interest.
- Event queue:
  - Contains a queue of notifications generated by **INSERT** and **UPDATE** operations on “**index_files”** table in “**sftp_db”**
- Data archiver program:
  - **“data_archiver.py”:**
    This program is the consumer of event queue. It is designed to continuously obtain a notification from the event queue (if there’s any available) and processes the notification accordingly based on the type of event that created the notification.
  - “**archive_db”:**
    Contains **“archived_cls”** table where records of all “**ASX300_NCS_CLS”** files in the SFTP server are saved into.

The next section of the document will go through each component of the program in more details.

Components

sftp_sync.py

First, the program creates a connection to the SFTP server. Once the connection is established, the program will:

- Obtain a list of files in the SFTP server which contains the substring "**ASX300_NCS_CLS"**
- Get metadata for each of these files
- Go through each file in the list and:
    - Insert the metadata of the file into "**index_files**" in "**sftp_db"** if the file name does not exist in the table OR
    - Update the metadata of the file in **"index_files"** table in **"sftp_db"** if the filename already exists in the table but only if either of these 2 conditions is fulfilled:
        - The **"last_modified_time"** of the file stored in the table is older than the modified time of the same file obtained from the SFTP server
        - There is a difference in file size of the file stored in the table and the file size of the same file obtained from the SFTP server

sftp_db

This database contains 2 tables:

- **"index_files"**:
  This table contains metadata of all files read by **"sftp_sync.py"** fro the SFTP server including their filesize and last modified time.

  A trigger function is attached to this table such that any **INSERT** or **UPDATE** operation on this table will create a new notification object, which will then be pushed to the event queue. Any **INSERT** or **UPDATE** operation on the table will also generate a new record inside **"index_file_changes_log"** which contains records of modifications or insertions of file metadata in **"index_files"** table.

- **"index_file_changes_log"**:
  This table keeps a record of various modifications and insertions of file metadata made to **"index_files"** table. This is very useful for us to know when each record in the table was modified or inserted.

  Each record in this log file contains:
    - The old modified time of the file before the file metadata record is updated
    - The new modified time of the file after the file metadata record is updated
    - The old file size of the file before the file metadata record is updated
    - The new file size of the file after the file metadata record is updated

data_archiver.py

This program executes itself in a loop and it will keep checking for new notifications in the event queue. If a new notification is detected, the program will detect whether the event that generated the notification is an **UPDATE** operation or if it's an **INSERT** operation.

Once the program determines the type of operation, it will read the content of the file related to the notification from the SFTP server and perform data validation to ensure that the content of the file is in the correct format.

Should the validation fails, the program will not upload the content of the file to **"archive_db**" database. Instead, it will output a log letting us know which line of the file contains an error and why exactly the validation fails.

However, if the validation is successful, the program will upload the content of the file to the database. The way the program uploads the content of the file to **“archive_db”** is going to be different depending on the notification type:

- If the notification was generated by **INSERT** operation in **“sftp_db”** database, then each row of the file will be inserted into a new row in **“archived_cls”** table in **“archive_db”** database.
- If the notification was generated by an **UPDATE** operation in **“sftp_db”** database, the program will update the records in **“archived_cls”** table that correspond to rows of the file

archive_db

This database has a single table, **“archived_cls”** which contains an aggregration of all records extracted from any **“ASX300_NCS_CLS”** files in the SFTP server. The primary key of each row in the **“archived_cls”** table is the **“id”** column and it’s set to **yyyymmdd<ticker_symbol>** for each record in the index file.