Deploy Code For Production

Capstone Step 9

Approach

Created Docker Container Packaging with Scripts to Deploy:

- ➔ Azure Data Lake Gen2
- ➔ Azure MySQL Database
- ➔ Azure Databricks
- ➔ Azure Data Factory (ADF)

Exported pipelines from ADF workspace as json configs

Decisions

First Attempted Coordinating Pipelines between Airflow and Azure Data Factory (ADF)

- Connectors from Airflow to ADF were riddled with bugs

Then Attempted with ADF, using Batch jobs for processing data extract scripts:

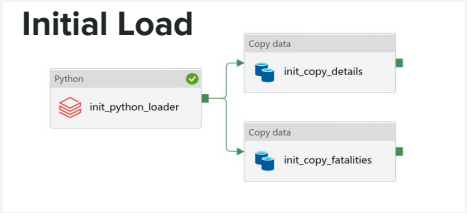
- Realized spinning up pools/jobs/tasks was overkill complexity for ETL
- I actually did get this fully working before abandoning it entirely
- Moved files to Step8/azurebatch_approach, as this could be considered failed testing

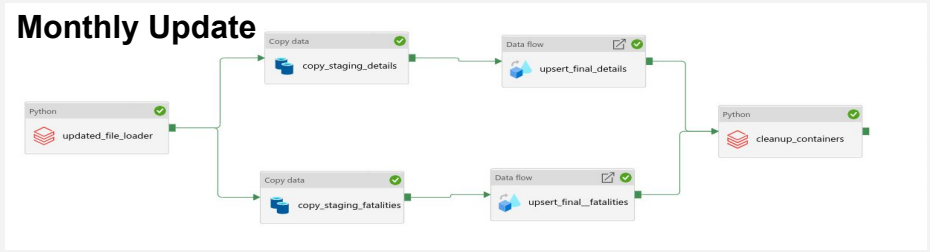
Finally settled on calling scripts using Databricks Python activity in ADF:

- Possibly over-resourced, but future-proofs against later uncertainties

Azure Data Factory

Each pipeline tailored for different use case based on source data release cadence



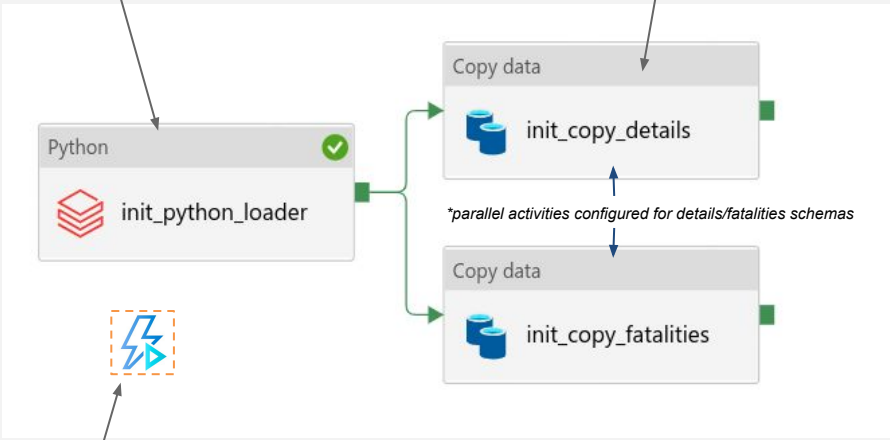


“Initial Load” Pipeline

Use Case: All details/fatalities csvgz files with post y2k data from ncei.noaa.gov severe weather data repo (source) require ETL to MySQL (ignores files with year of record < 2000, and all “location” files as it is entirely redundant with “details” files)

Databricks Python Activity:
run: `initial_files_to_blob.py`
extract: all csvgz files on source url with date of record > 2000
from: source url
to: data lake “allfiles” container

Copy Data Activity
copy: new 2022 csvgz file
from: data lake “newfiles” container
to: mysql staging table



“Monthly Update” Pipeline

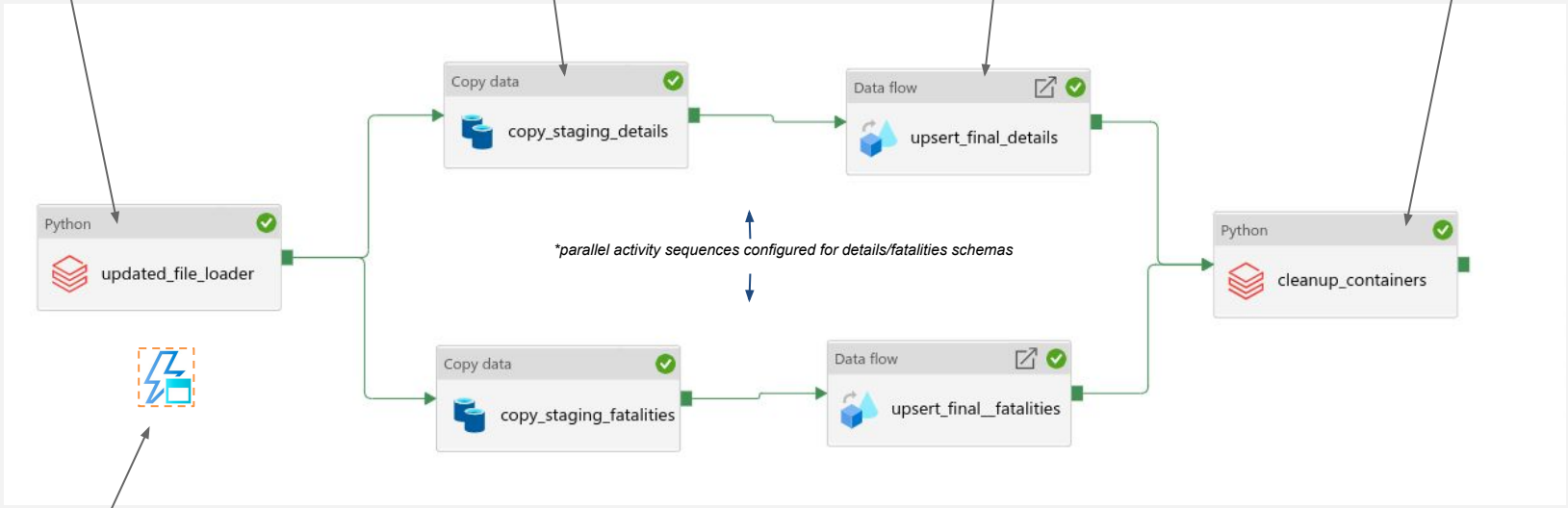
Use Case: 15th of each month, ncei.noaa.gov severe weather data repo (source) drops updated csvgz file for current year+month with recent data appended and requiring ETL.

Databricks Python Activity:
run: `updated_file_to_blob.py`
if: source url's 2022 file shows modification since last ETL
then: extract updated 2022 csv.gz file
from: source url
to: data lake “newfiles” container

Copy Data Activity
copy: all data from updated 2022 csvgz file
from: data lake “newfiles” container
to: mysql staging table

Data Flow Activity
insert: only new rows based on primary key
from: mysql staging table
to: mysql final table

Databricks Python Activity
run: `clean_newfile_container.py`
delete old 2022 files
from: “allfiles” container
copy updated 2022 files
from: “newfiles” container
to: “allfiles” container
delete updated 2022 files
from: “newfiles” container



Trigger
schedule: @monthly on 16th

“Yearly New” Pipeline

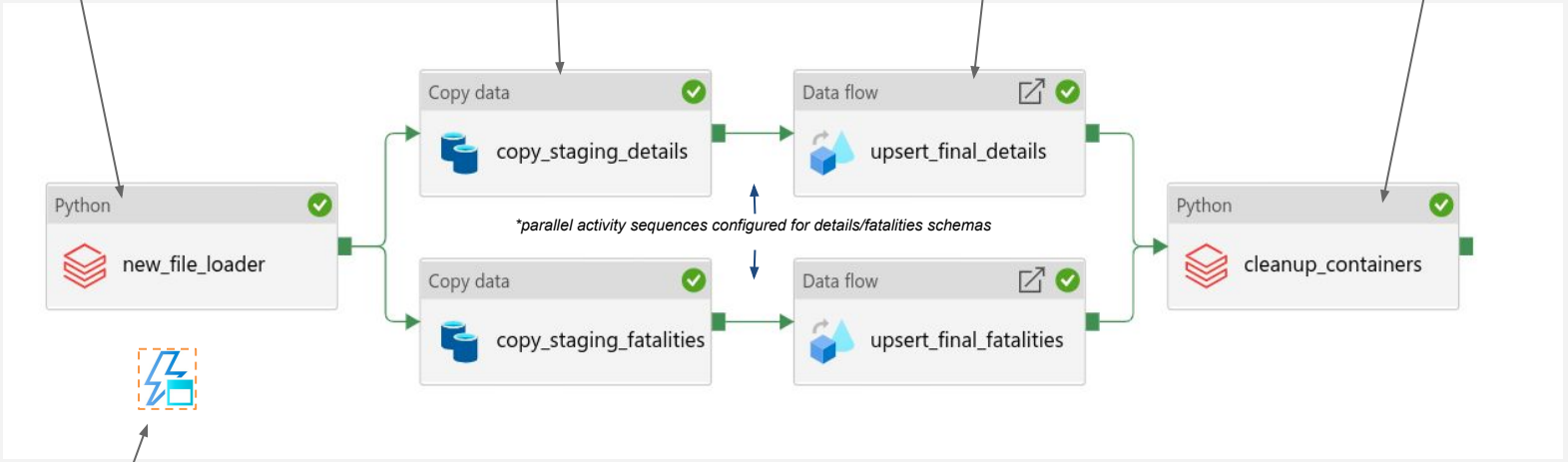
Use Case: April 15th of each year, ncei.noaa.gov severe weather data repo (source) drops new csvgz file with fresh January data of that year requiring ETL.
(it takes 75 days for severe weather event data to be recorded, submitted, and published)

Databricks Python Activity:
run: `new_file_to_blob.py`
if: source url has dropped a new filename for new year
then: extract new csv.gz file
from: source url
to: data lake “newfiles” container

Copy Data Activity
copy: all data from new year csvgz file
from: data lake “newfiles” container
to: mysql staging table

Data Flow Activity
insert: new rows based on primary key (equates to all)
from: mysql staging table
to: mysql final table

Databricks Python Activity
run: `clean_newfile_container.py`
delete old 2022 files
from: “allfiles” container
copy new 2022 files
from: “newfiles” container
to: “allfiles” container
delete new 2022 files
from: “newfiles” container



Trigger
schedule: @annually on April 16th

Testing Full Pipelines

I had already vetted the basic ingestion in Step 8 tests

This final deployment added tests for all edge cases and iterative updates