# Physical architechture:

Development machine is a Silicone based mac

There are 3 RDBMSs used in this system for data excluding the airflow metadata db

Oracle Cloud free tier hosted Oracle 23ai database:

A locally hosted docker image of an sql server database:

Microsoft SQL Server 2022 (RTM-CU19) (KB5054531) - 16.0.4195.2 (X64) Apr 18 2025 13:42:14 Copyright (C) 2022 Microsoft Corporation Developer Edition (64-bit) on Linux (Ubuntu 22.04.5 LTS) <X64>

A cloud hosted postgres db v17.6. Hosted for free on the below

<https://console.aiven.io/>

Running a conda distribution of python v3.10.9 for the application code

With Java version and path set using the below in /Users/stephenpir/.zshrc and .bash\_profile

export JAVA\_HOME=$(/usr/libexec/java\_home -v 17)

export PATH=$JAVA\_HOME/bin:$PATH

This was to resolve using pyspark to connect to databases

Attempted running airflow version 3.1.0 in localhost over a second conda environment in python v3.10.18

conda activate airflow\_env

# To start the airflow application run:

airflow standalone

# URL to the airflow UI

http://localhost:8080/

This had many issues in and ARM environment and is unstable.

In the end used a docker compose image of airflow 2.9.3 as baseline for the airflow environment. It was then extended via dockerfile and yaml files to give all the additional python modules and java runtime required by the system.

I used Visual Studio Code (VS code), with several extensions to cater for the multiple db types. – In the end dropped it down to just the one for SQL Server as the extensions had conflict. Now using SQL Developer for Oracle, and TablePlus for Postgres

Finally, all the code was configured using github.

## Scenario 1:

Load data from multiple different sources to one postgres table and fill out missing data, so it’s ready for analysis. The initial set of files is directly from the national lottery website and while the quality of data can be considered the highest, the quantity is small.

1st set of Source files, with formats:

euromillions-draw-history\_20210416-20211012.csv

DrawDate,Ball 1,Ball 2,Ball 3,Ball 4,Ball 5,Lucky Star 1,Lucky Star 2,UK Millionaire Maker,DrawNumber

euromillions-draw-history\_20221223-20230616.csv

DrawDate,Ball 1,Ball 2,Ball 3,Ball 4,Ball 5,Lucky Star 1,Lucky Star 2,UK Millionaire Maker,European Millionaire Maker,DrawNumber

1. Create target db structure for data

The columns for the two files are almost identical, the only difference being the “European Millionaire Maker” column. As such this column will not be created in the db taget table (euromillions\_draw\_history) and removed from the dataset after reading and prior to load.

Script:

create\_EuroMillions\_lottery\_table\_PG.psql

1. Read csv files into dataframes, consolidate into a single dataframe and load into the euromillions\_draw\_history\_PG table

Script:

load\_source1\_Euromillions\_lottery\_data\_to\_postgres.py

1. There are only 103 data points so a new data source from elsewhere will be used. This data source is of slightly different format but has significantly more data from a different period

2nd Source file and format:

EuroMillions\_numbers\_20040110-20211201.csv

Date;N1;N2;N3;N4;N5;E1;E2;Winner;Gain

Immediate observations are;

* There are different columns, but the important ones of date and numbers drawn are there
* There is no draw number in the new dataset but that can potentially be generated using the existing dataset as an anchor point
* The file is semi-colon delimited and not comma delimited
* There is date overlap with data from euromillions-draw-history\_20210416-20211012.csv

Firstly, a column will be added to the euromillions\_draw\_history\_PG table to include a marker for the source and populated for the already loaded data. The primary key and other constraints that the new data will violate are to be removed as well.

To keep it simple the existing structure of the euromillions\_draw\_history\_PG table shall remain unaltered other than the change above, and the new data will be loaded with obvious column mappings between the 2 datasets.

Once loaded a data comparison where there are overlaps will be conducted, and appropriate action taken.

1. On examination of the data, the overlap are not duplicates but spurious dates which need to be examined further/ removed.

To identify what is happening and which dated need to be removed, more information is required. A day\_of\_the\_week field will be added to the data and populated with the day derived from the draw\_date field. As this is Euromillions data, any day that is not Tues or Fri should be investigated further.

Querying the outliers in the second dataset has shown the data is not correct. The earliest date in the data (10/JAN/2004) is before the Euromillions lottery started (13 February 2004[[](https://en.wikipedia.org/wiki/EuroMillions#cite_note-2)). The 10th of Jan 2004 was also a Saturday. Checking further it would appear that much of the date formatting in the second data set is incorrect, exchanging the day and month numeric components. E.g. The 10th of Jan 2004, represented as 2004-01-10 is actually meant to be 2004-10-01, the 1st of October. This can be reasonably assumed as checking the actual draw numbers for the 1st of October vs. the data set row for 10th of Jan 2004 showed they are the same.

Result: The second dataset can’t be trusted and another source for additional data should be found.

## Scenario 2:

A new comprehensive source for data has been found. The earliest draw date fits correctly with what is known about the Euromillions draw, and a cursory check of a random sample of data compared with that in the high-quality National Lottery data source has shown this data to be accurate.

Unfortunately, the data is not downloadable in csv format as the website is intended to check single instance results.

The aim is to Webscrape full dataset and load into another table where data can then be compared to validate what was initially loaded.

For academic purposes this system will be database heterogenous and this source will be loaded to an Oracle Cloud instance

URL to scrape:

https://www.beatlottery.co.uk/euromillions/draw-history/

1. Scrape numbers from website iteratively for each year and load into csv file add more details about the script

Script:

scrape\_euromillions\_data.py

Output:

euromillions\_draw\_history\_scraped.csv

1. Create target db structure for data

Script:

CREATE\_TABLE\_EUROMILLIONS\_DRAW\_HISTORY.osql

1. Read output csv file euromillions\_draw\_history\_scraped.csv. Sort by date and add a sequence to give the draw number, then load into target table using thick client. Thick client used for academic purposes to use a different method of connection, as for larger data loads performance would likely be better.

Thick Client:

<https://www.oracle.com/uk/database/technologies/instant-client/macos-intel-x86-downloads.html>

Script:

load\_euromillions\_data.py

1. Comprehensively validate the new data against the data from scenario one to ensure that the numbers for each date are correct and that the generated draw number for the scraped data also matches.

Script:

validate\_data.py

## Scenario 3:

Use Pandas to generate graphics for distributions etc

Use traditional methods to generate draw numbers from the existing data

Use ML to generate numbers from historic results

Script 1: analyze\_euromillions\_data.py

Uses pandas to provide graphics of the data, showing number frequency and distributions in pie chart and heat maps

Script 2: predict\_euromillions\_numbers.py

Looks at the data and provides several different strategies to generate draw numbers, most frequent, least frequent etc

Script 3: predict\_ml\_euromillions.py

Uses sklearn machine learning to generate a set of predicted draw numbers

Script 4: predict\_pytorch\_euromillions.py

Uses pytorch to do the same as the previous script i.e. machine learning to generate a set of predicted draw numbers

## Scenario 4:

Move data between dbs using intermediate json file and or parquet and airflow?

Script 1: export\_oracle\_to\_local\_files.py

Extracts all data from the oracle table euromillions\_draw\_history, and writes it out to both json and csv format

Script 2: load\_json\_to\_sqlserver.py

Conditionally creates a target table euromillions\_draw\_history\_sqlserver if it doesn’t exit and loads the json data created by export\_oracle\_to\_local\_files.py. Additionally it performs the load as an upsert

## Scenario 5:

Airflow pipeline for creating the tables – Environment initialisation batch

Pipeline process to scrape the results and load them to file or into the db – Scheduled for twice a week

Pipeline with a sensor?

## Scenario 6:

Use Kubernetes and Docker to set something up on all of this

Further enhancements:

* Adding constraints to the db tables PK, and Unique constraints on date, draw number
* A better way to handle the delta than getting all the results for comparison