Raven Analytics

Project 3 - Group 1

Jordan Dass
Adam Freeman
Mitchell Langdon
Tracey Martin
Marcus Whitelock

Motivation & Summary

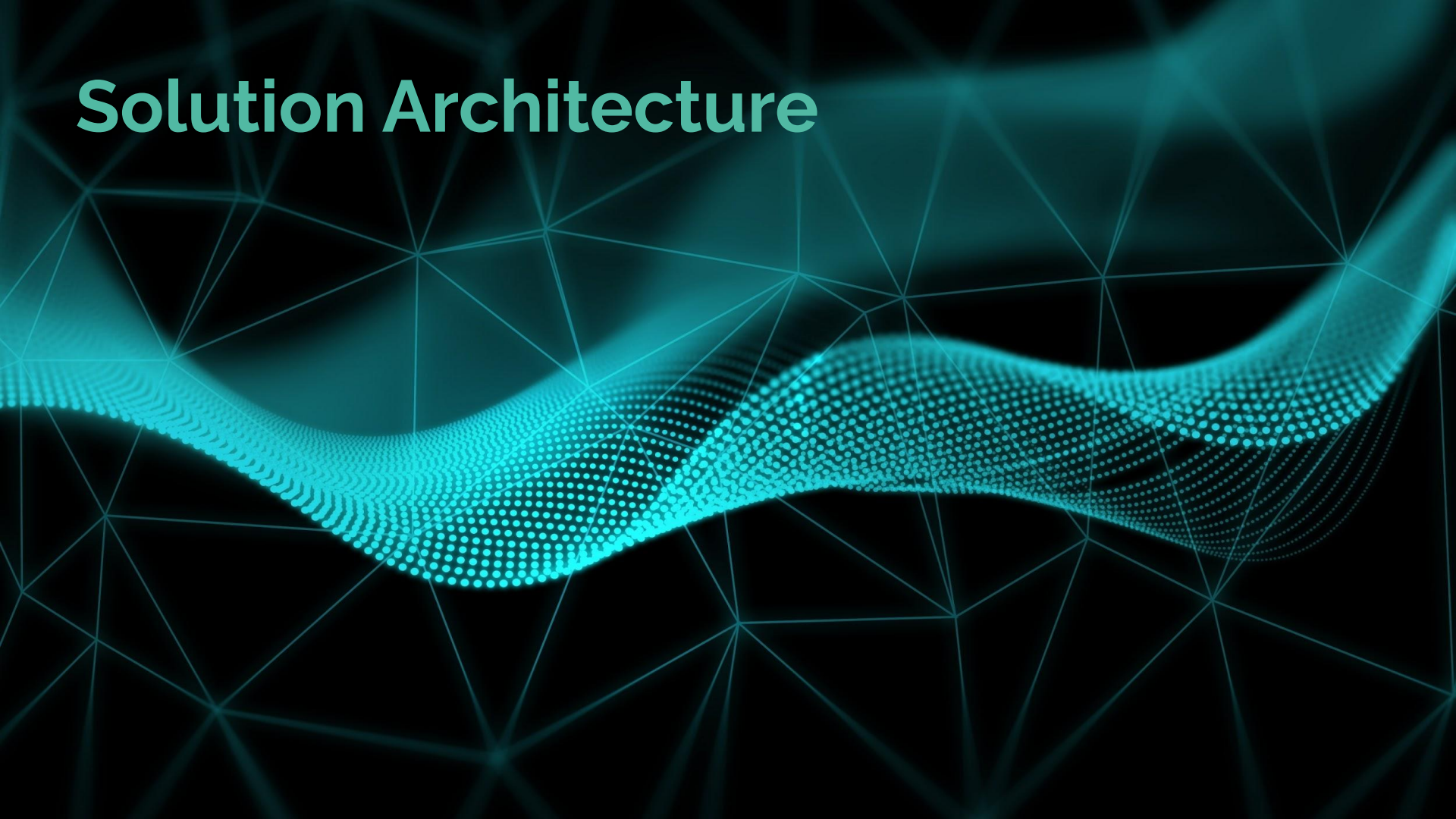
Problem	• Analysis is only as good as the data and we are drowning in data. How do we get the data we want in the format we want it
Solution	• Create our own data collection engine where we control the source and how the raw data is processed.
Value	• Full control of customizable, scalable and agile solution

Data Sources

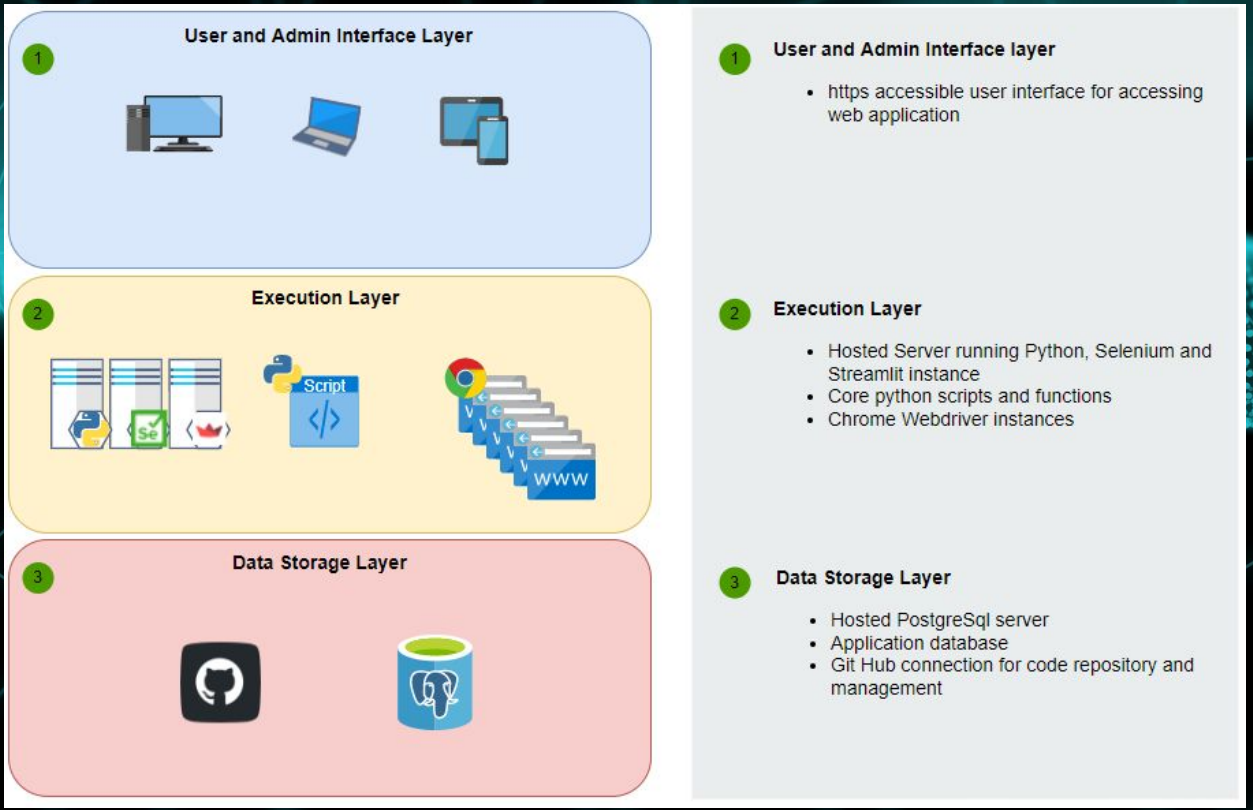
The Raven analytics tool has been designed to source data from any publicly accessible websites. For the initial iteration the following sources have been used.

- Hotcopper.com.au: Australia's largest stock trading and investment forum
- Marketindex.com.au: Financial portal for the Australian stock market.

Solution Architecture



Architecture Diagram



1 **User and Admin Interface layer**

- https accessible user interface for accessing web application

2 **Execution Layer**

- Hosted Server running Python, Selenium and Streamlit instance
- Core python scripts and functions
- Chrome Webdriver instances

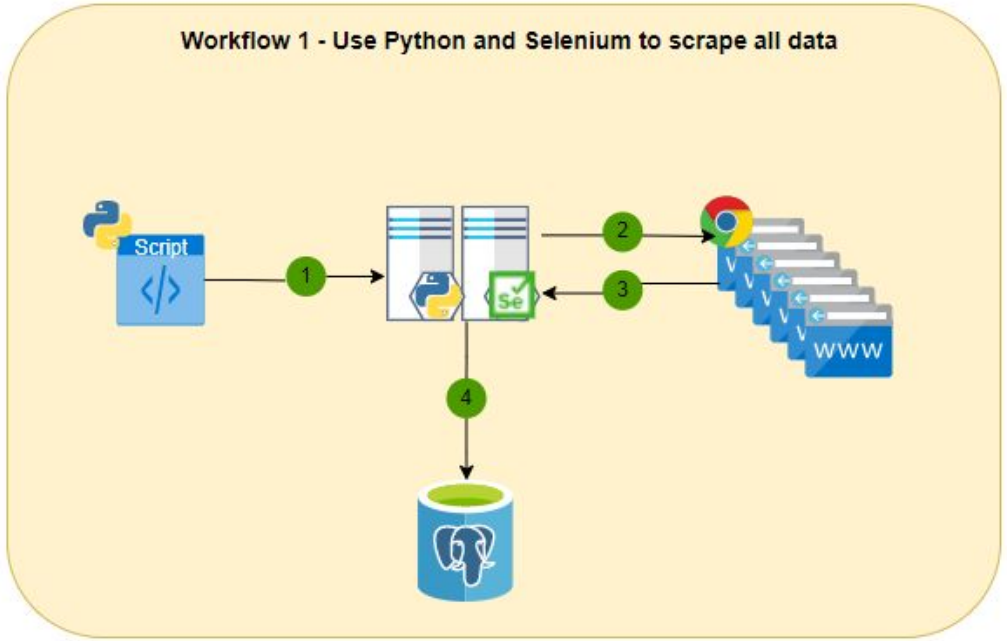
3 **Data Storage Layer**

- Hosted PostgreSql server
- Application database
- Git Hub connection for code repository and management

Operational Workflows

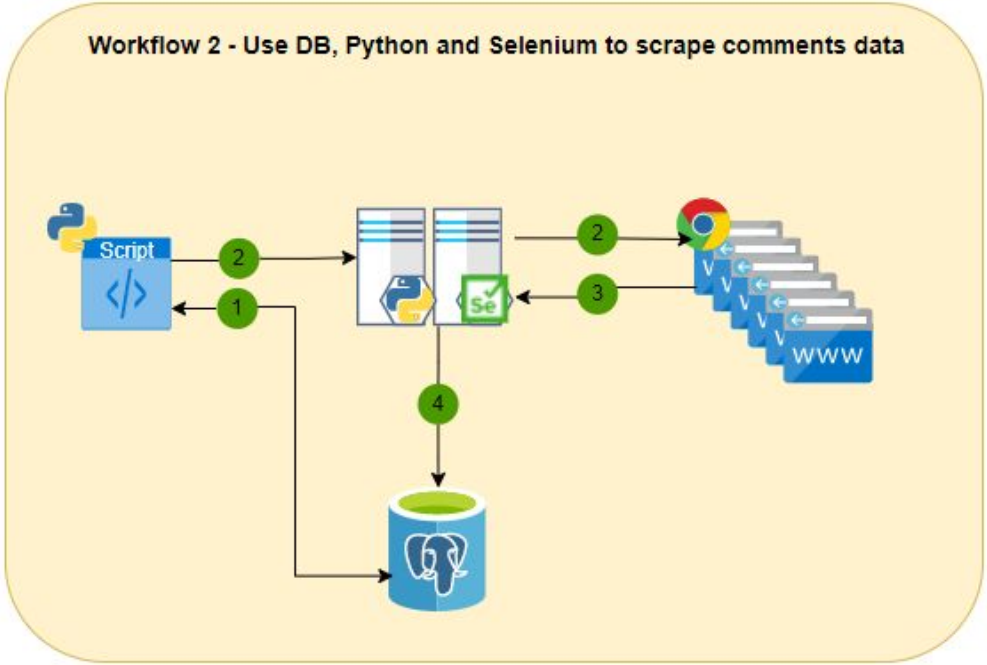
The following slides demonstrate some of the key workflows as a brief summary of the solution operation

Workflow 1 Diagram



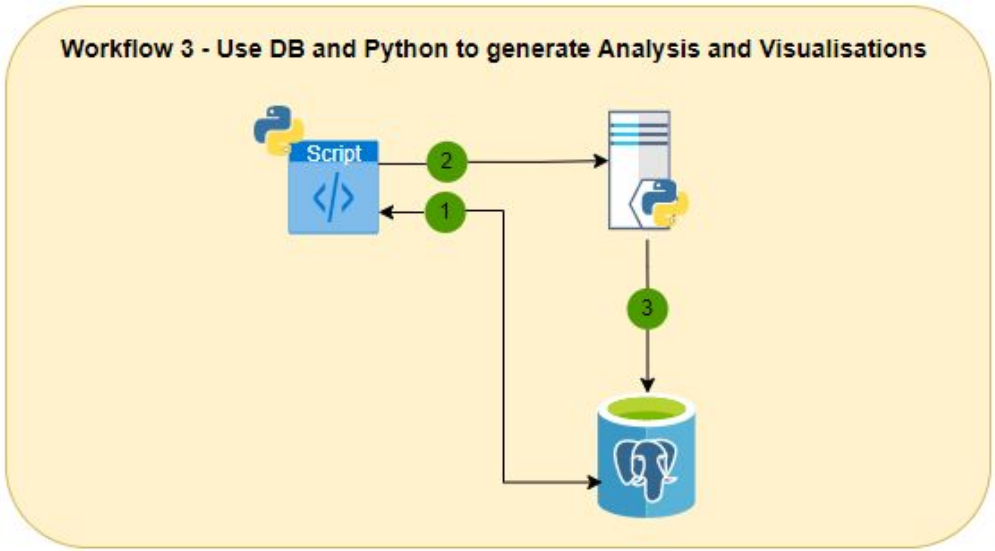
1. Execute the hot_copper.py with embedded scrape_asx_tickers.py python scripts on the server.
2. The script execution uses a number of libraries including Selenium, SQL Alchemy, Beautiful Soup, URLLIB and Pandas from the Servers Python environment to execute the data collection
3. The Python script execution code calls the Chrome Webdriver to establish a connection the the target website and scrape the targeted web page elements. This is returned to the code execution engine and a Pandas DataFrame is created
4. The returned DataFrame which includes the following fields Ticker, Subject, Poster, Likes, Date, HREF_Link, Ticker_Filter are written to the PostgresSQL Database table - **hc_stock_sum**. The returned DataFrame which includes Tickers is written to the PostgresSQL Database table - **hc_ticker_list**.

Workflow 2 Diagram



1. Execute the scrape_comments.py python script on the server. This connects to the Database and provides the **hc_stock_sum** table as a Pandas Dataframe to the code execution engine. .
2. The script execution uses a number of libraries including Selenium, SQL Alchemy, Beautiful Soup, URLLIB and Pandas from the Servers Python environment to execute the data collection using the **HREF_Link** returned from the DB.
3. The Python script execution code calls the Chrome Webdriver to establish a connection the the target website and scrape the targeted web page elements. This is returned to the code execution engine and a Pandas DataFrame is created
4. The returned DataFrame which includes the following fields HREF_Link and Comments are written to the PostgresSQL Database table - **hc_top_likes.** The returned DataFrame contains only HREF_Links and Comments for posts with >= 20 likes.

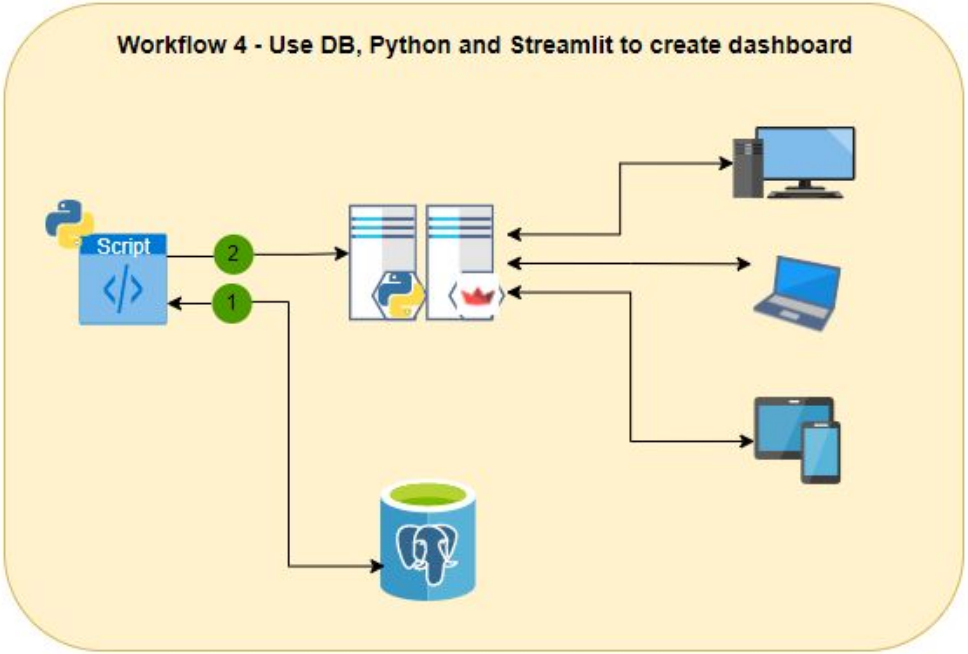
Workflow 3 Diagram



1. Execute the appropriate python SQL scripts on the server. This connects to the Database using the SQL Alchemy library and provides the SQL query results table as a Pandas Dataframe to the code execution engine. .

2. These Python SQL scripts are tested on the Servers Python engine and then created as functions that can be called by the Streamlit Dashboard. Any new queries, tables or views can be saved to the PostgresSQL DB for re-use.

Workflow 4 Diagram

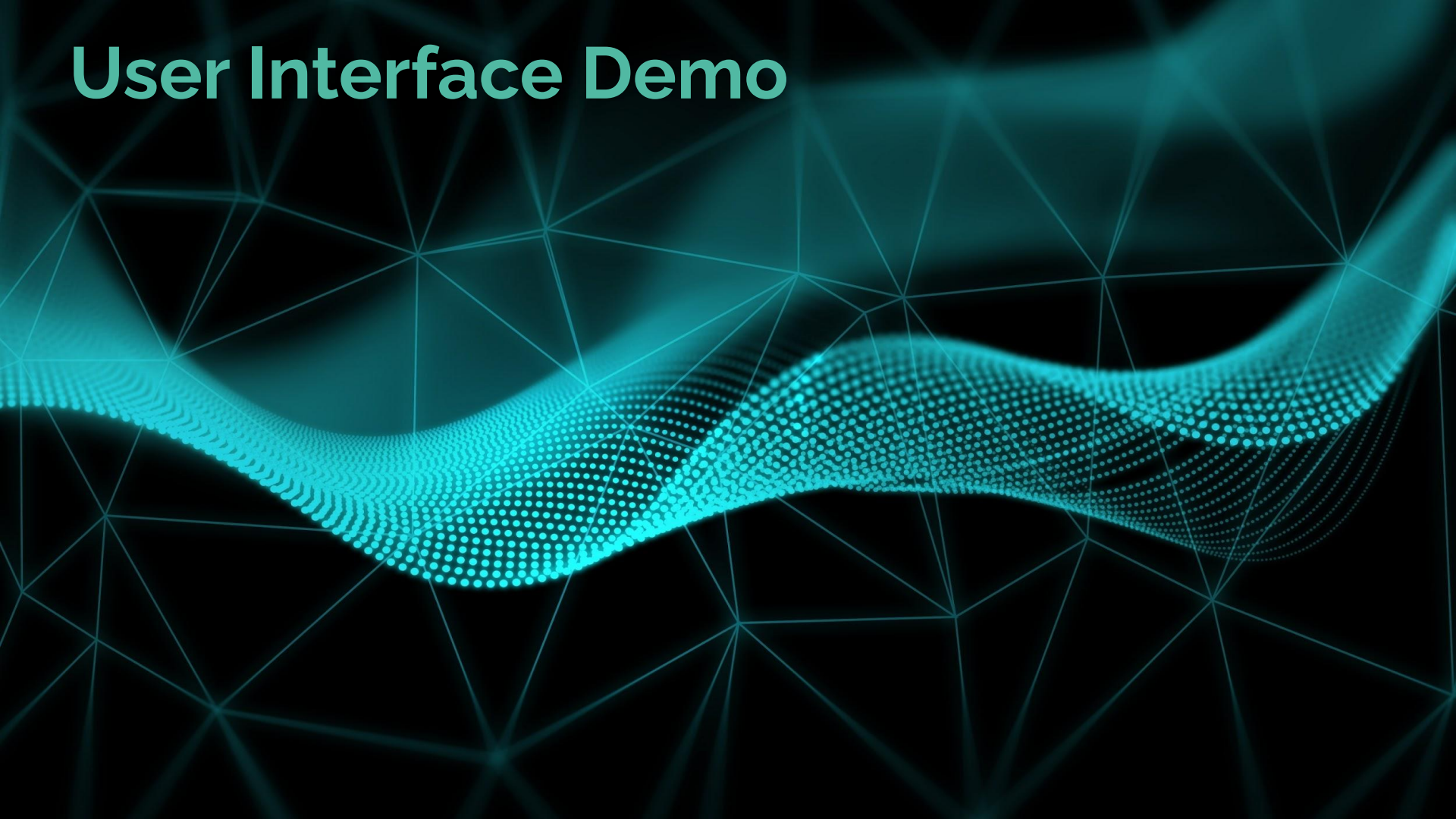


1. Data is retrieved from the database using the SQL Alchemy library along with the required SQL query embedded in the Python code.

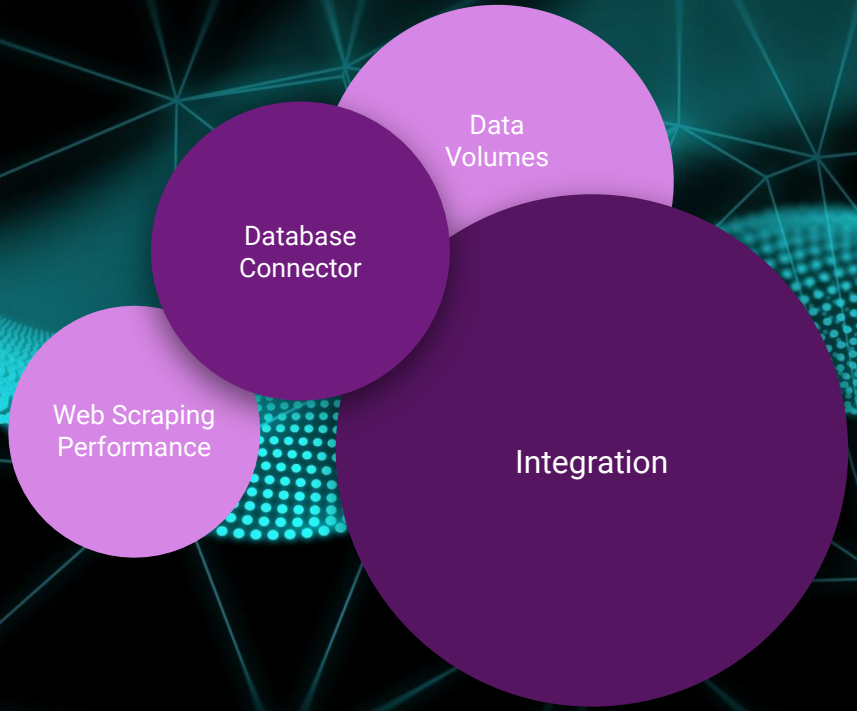
2. The required data is passed back to the script engine as a data frame that can be manipulated as required, then used as required in the Streamlit Dashboard.

3. End users are able to connect to the Streamlit dashboard hosted on the Server. This allows interaction with dashboard components like self service searches and reporting.

User Interface Demo



Challenges


Data Volumes
Database Connector
Web Scraping Performance
Integration

Questions