Contents

[Build data lake house from data in a web page 1](#_Toc135927067)

[Goal of system 1](#_Toc135927068)

[**What data do I need?** 1](#_Toc135927069)

[**What is the data lake house?** 1](#_Toc135927070)

[Technique used in this system. 2](#_Toc135927071)

[Workflow of system: 2](#_Toc135927072)

[Set up: 3](#_Toc135927073)

[Explain dags(parts) in this system: 5](#_Toc135927074)

[How to use system? 5](#_Toc135927075)

# Build data lake house from data in a web page

**Goal of system**

* The system monitors changes in the price of many popular coins in the world.
* What does the system really do?
  + Extracting data from a [web page](https://coinmarketcap.com/).
  + Transforming raw data.
  + Store clean data in a ‘data lake house’.

**What data do I need?**

* + Name
  + Symbol
  + Extract data about price
  + volume\_24h
  + 1h%, 24h%, 7d%
  + marketCap
  + Circulating Supply

**What is the data lake house?**

It is a new architecture which provides ability of both data warehouse and data lake. It can store structured, unstructured, and semi structured data. With architecture divide into 3 layers:

* + Bronze layer: store raw data.
  + Silver layer: store cleaned data.
  + Gold layer: store data which is enriched and ready to make report or use to other purpose.

**Technique used in this system.**

* **Selenium**:  is a free (open source) automated testing framework used to validate web applications across different browsers and platforms. You can use multiple programming languages like Java, C#, Python, etc to create Selenium Test Scripts.
* **Bs4** - beautifulSoup: is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.
* **Pandas**: is a Python library used for working with data sets.
* **Spark**: Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size. It provides development APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads—batch processing, interactive queries, real-time analytics, machine learning, and graph processing.
* **Minio**: is a high-performance, S3 compatible object store. It is built for large scale AI/ML, data lake and database workloads. It runs on-prem and on any cloud (public or private) and from the data center to the edge. MinIO is software-defined and open source under GNU AGPL v3.
* **Airflow**: is an open-source platform for developing, scheduling, and monitoring batch-oriented workflows. Airflow’s extensible Python framework enables you to build workflows connecting with virtually any technology. A web interface helps manage the state of your workflows.
* **Docker**: Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

**Workflow of system:**

* start point: web page.
* using selenium to get a html source.
* using bs4 to extract needed info.
* using pandas to transform data from raw to clean.
* using python to store clean data locally. It like a backup when you need.
* using python to upload data from local to minio - data will store in bronze layer.
* using spark to connect minio → read data in csv file → incremental load into a parquet file - data store in silver layer.
* using spark to read only new part in silver layer → transform → load into dim and fact table - data store in gold layer.
* end point: gold layer of delta lake

A picture containing text, screenshot, diagram, design

Description automatically generated

**Set up:**

1. Download my folder - pull my project in github from [here](https://github.com/ckikriehanra/crawl-data-from-coin-market).
2. Download and install docker desktop [here](https://docs.docker.com/desktop/install/windows-install/).
3. Start docker desktop.
4. Download and install Visual studio code [here](https://code.visualstudio.com/download).
5. You can modify user account of Minio inside file with path ./minio/.env
6. You can modify your folder name. file name which will be used to save data in Minio inside file with path ./my\_lib/config.py
7. Pull my image:
   1. Open Docker Desktop
   2. Type Ctrl+K
   3. Enter key words: kiriharacken/custom-airflow
   4. A window open, you click pull to pull this image to local machine.
8. Open folder in vs code:
   1. Open vs code
   2. Type: Ctrl + K + O
   3. Choose your folder need open which you downloaded named: Crawl\_Webpage\_docker
9. Download .jar for spark. To know more information you need to read file .txt in ./spark/resources/jars/

A screenshot of a computer

Description automatically generated with medium confidence

1. Open terminal: Ctrl + J
2. Start project by scripts:
   1. docker-compose up airflow-init
   2. Open new terminal by (Ctrl + Shift + `) → type: docker-compose up -d
   3. Open new terminal by (Ctrl + Shift + `) → type: docker container ls
   4. Wait several minutes and retry: docker container ls. It ok if you see:

A picture containing screenshot, text

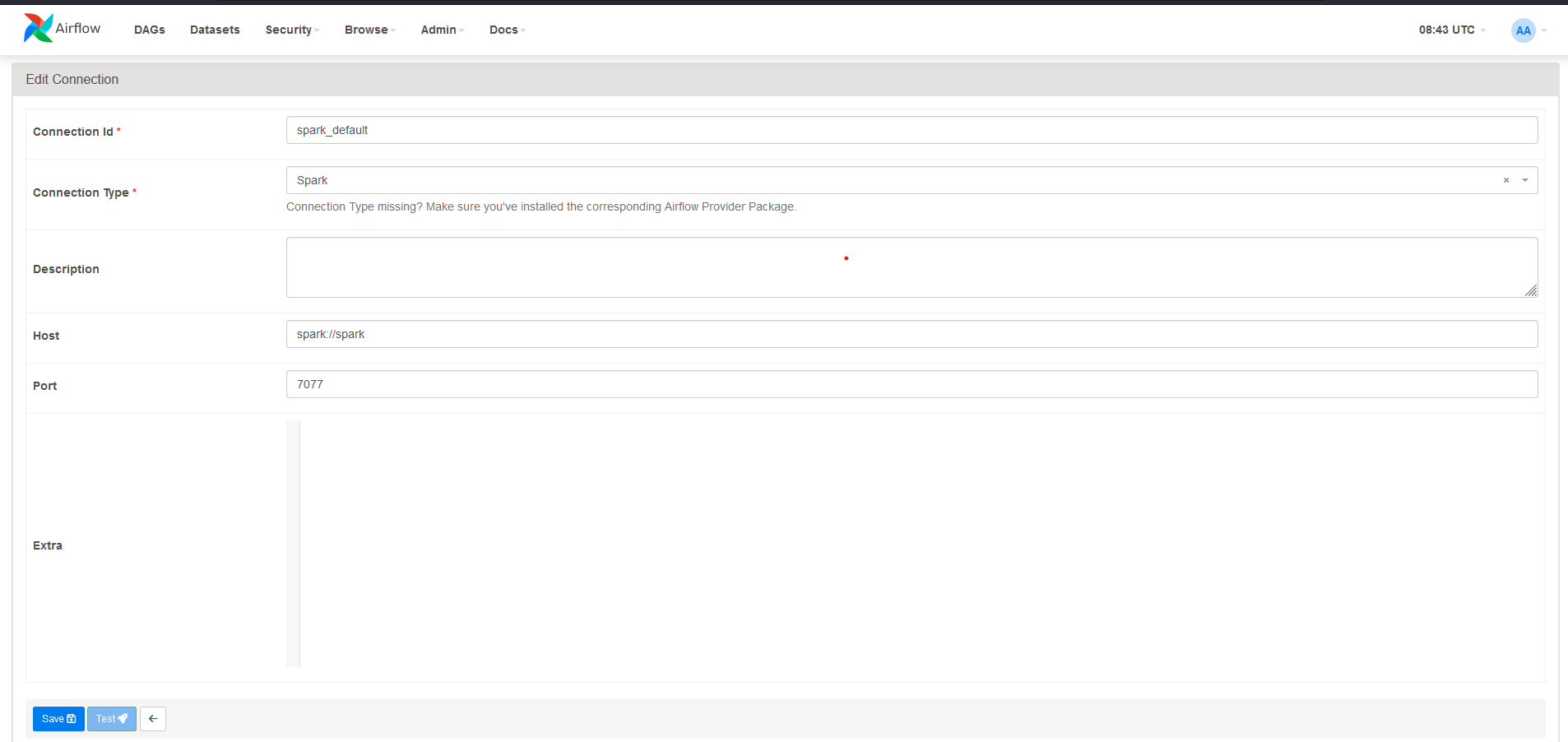
Description automatically generated

* 1. Open browser, in search bar enter: <http://localhost:8080> → Sign in. It is ok if you see console like image:

A screenshot of a computer

Description automatically generated with medium confidence

1. In header, you choose Admin → choose connection. Now you need to configure your connection to help airflow interact with spark:
   1. Click Add a new record.
   2. Add a connection with information then click save.



# Explain dags(parts) in this system:

* Load\_data\_into\_dim\_date: used to full load data into dim\_date, can be triggered by handle.
* Crawl\_data: crawl data from web page, transform and load data into data lake house.
* Load\_data\_into\_fact\_price\_hour: transform data into average value and load data into fact\_price\_hour. This value beyond latest last hour.
* Load\_data\_into\_fact\_price\_day: transform data into average value and load data into fact\_price\_day. This value beyond lastest last day.
* Check: used to check data quality, check number of rows in table, check first 20 rows of tables.

# How to use system?

* Before running this project, you need to check data lake:
  + Open browser and enter <http://localhost:9090>.
  + Sign in by username: kirihara, password: minioadmin. You can change this account by going inside ./minio/.env.
  + If you see existed bucket then it’s ready to start. Like this:

A screenshot of a computer

Description automatically generated with medium confidence

Else you need to create a new bucket with name “cken-coins-data”. You can set specific name by going inside file with path ./my\_lib/config.py.

* Firstly, dag needs to be run is ‘Load\_data\_into\_dim\_date’. With this dag you only need to trigger once time.
* Second dag needs run is ‘crawl\_data’. This dag set time to automatically runs every 5 minutes.
* After 1 hour first, you can trigger dag ‘load\_data\_into\_fact\_price\_hour’.
* After 1 day first, you can trigger dag ‘load\_data\_into\_fact\_price\_day’.
* Dag ‘check’ will be used with goals to check number of records in each layer in data lake house, 20 first rows of each layer. Otherwise, it shows numbers of rows in silver layer.

How to use this dag?

Each time you want to check results, you go inside dag ‘check’ and trigger it.

* + - If you want to see number of rows in silver layer, by minute, by hour, by day then you can see in file summary.csv with path ‘cken-coins-data/summary.csv’ in [minio](http://localhost:9090).
    - If you want to see 20 first rows of each type of data you need to go inside log of task ‘check’.