# Data Solutions

## DAQ - Application programing Interface

Data Analytics (DE)

Daasdt

# Steps

1. Projects Requirements
2. Concept Explain
3. Design ( Architecture)
4. Solution ( Implementation)
5. Coding
6. Git

# Requirements

1. Collect and publish the data
2. Implement Incremental loads
3. Use Python
4. Customized Stock data
5. Testing code
6. Manage the code in git

Requirement is to create the data engineering project to collect the data from API , parse the data and store it in RDBMS. Sample dataset will be a stock company level index data and this is small data set.

Daasdt

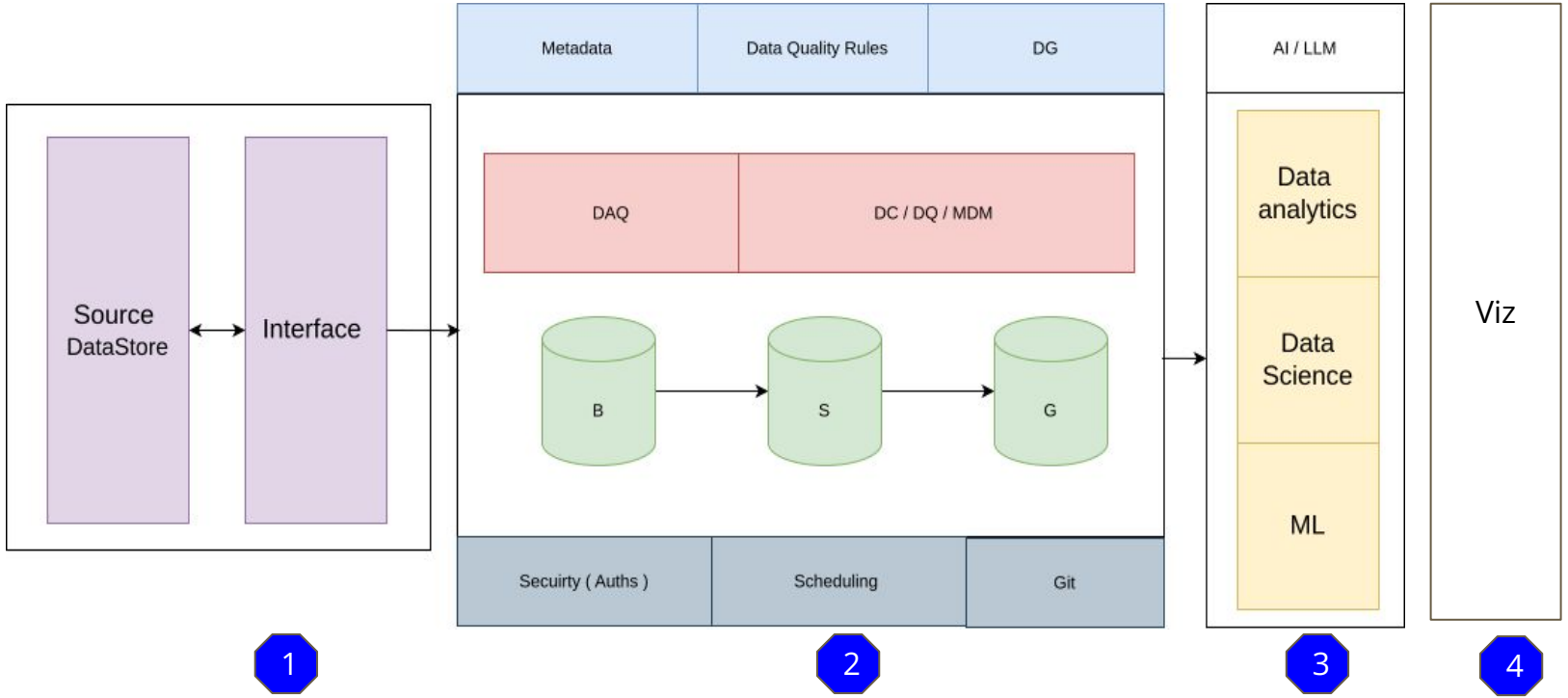
# Key Concepts Using

The concepts ( Technical and General) are going to use in the projects are here.

1. API , Open API
2. Docker
3. Datasets & Database
4. Python
5. JSON , FAST API and Async
6. Testing

1. Securities
2. XML , WSDL , SOAP
3. Database Connections

# Design - ARCHITECTURE




1. Source Interface , 2. Data Transformation , 3. Data consumption, 4. Visualization

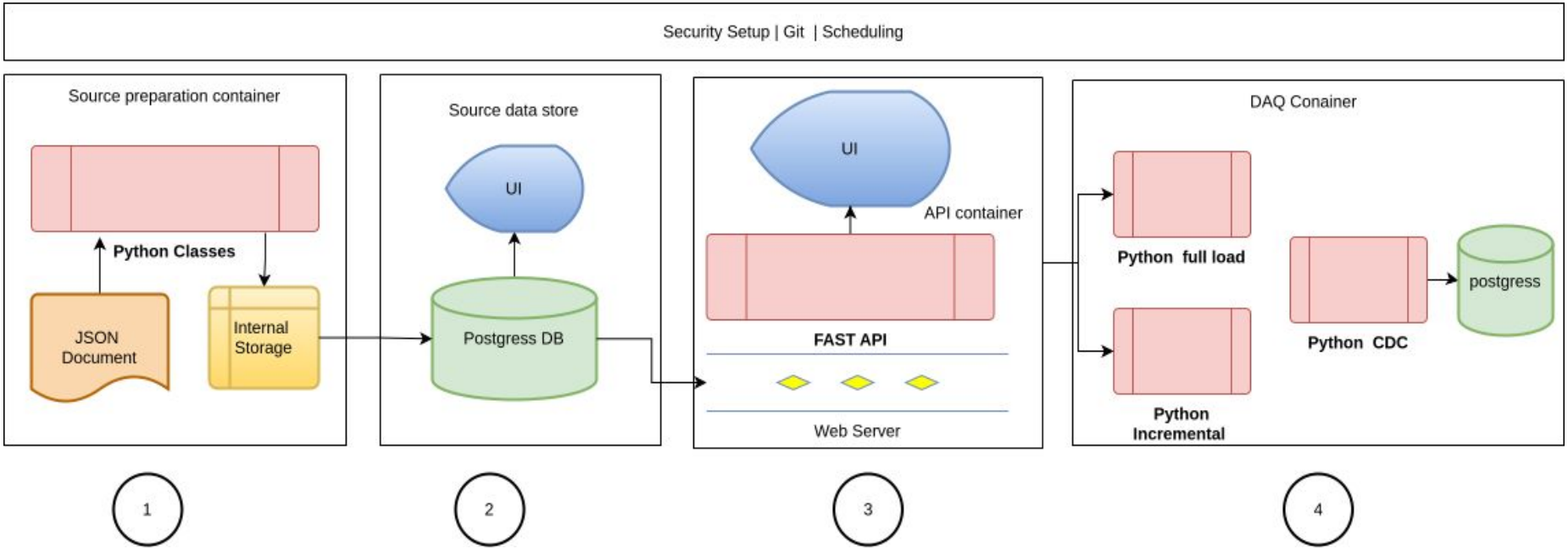
# Design - ARCHITECTURE - Details

1. Source Interface : Has two components
    a. Source Data Set ( RDBMS , NOSQL , FLAT FILES , SEMI /Unstructured data)
    b. Interface to expose the data to outside world. ( API , MSQ, STREAM)
2. Data Transformation layer ( all ELT , ETL Components)
    a. DAQ - Data Acquisition Process
    b. DE - Data Engineering process ( DQ, DC , MDM)
    c. Metalian Architecture data sets ( Bronze , Sliver , Gold )
    d. Metadata
    e. Data quality rules
    f. Data governance principle
    g. Security - Connecting to source interface
    h. Scheduling
    i. Git

# Design - ARCHITECTURE - Details

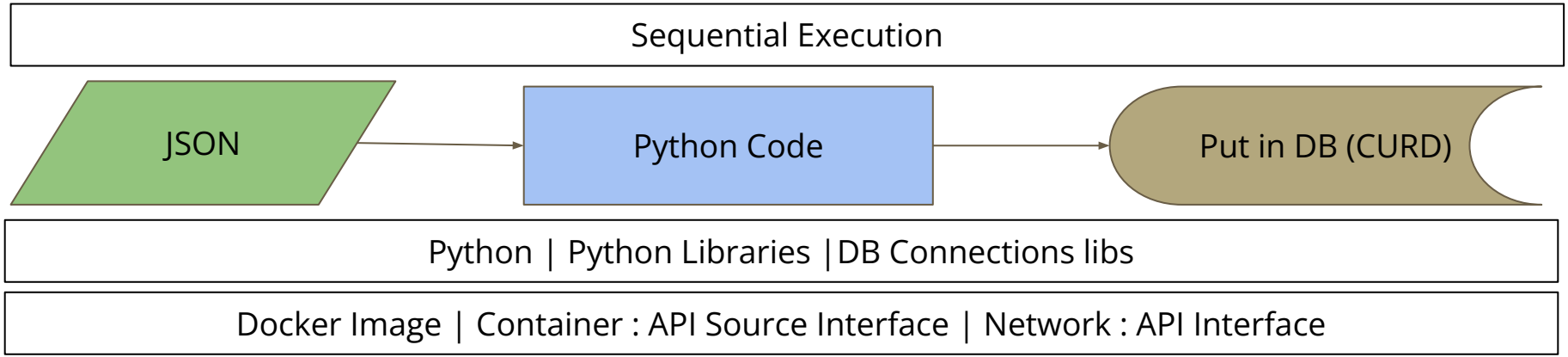
1. Data Consumption :
    a. Data Analytics
    b. Data Science
    c. Machine learning
    d. AI and LLM
2. Visualization & Reports
    a. Visual analytics
    b. Reporting

# Implementation - SOLUTIONS - Details




1. Docker + Python , 2. Postgres DB + pgadmin , 3. FastAPI 4. Python

# Implementation - SOLUTIONS (Docker + Python)



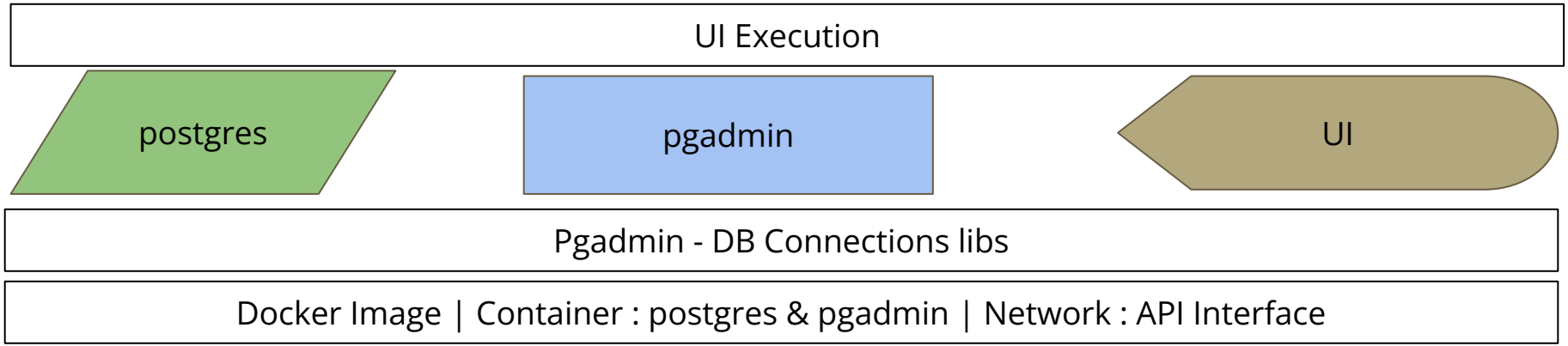

Component require to build this docker container

1. Docker - Ubuntu Image
2. Python - 3.9
3. Libs - JSON (Json document parsing ) , asyncio ( Put data in database) SQLAlchemy 2.0

**Python Logic :**

1. Read JSON file
2. Parse the data
3. Do CURD Operation in DB

# Implementation - SOLUTIONS (Postgres)



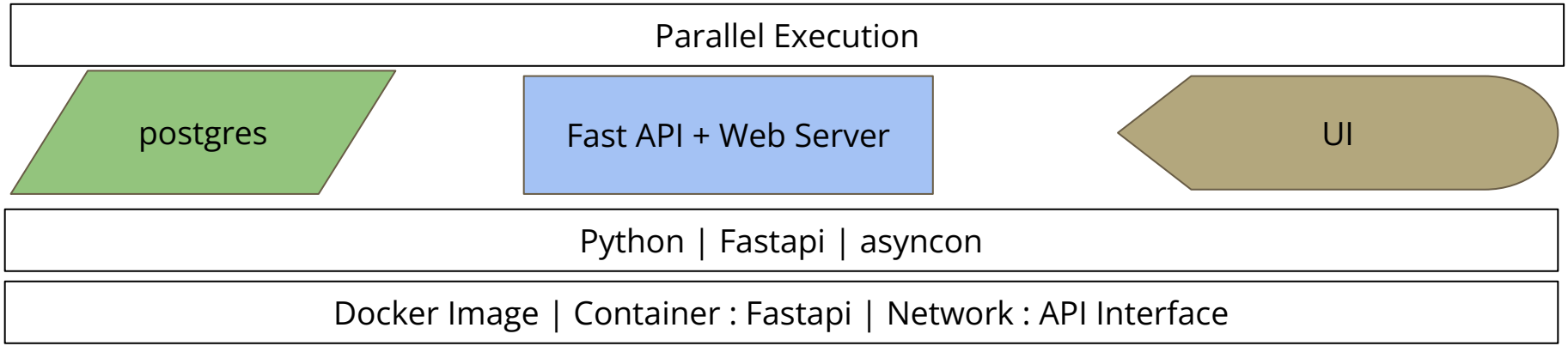

Component require to build this docker container

1. Docker - Ubuntu Image + Postgres
2. Docker - Ubuntu Image + pgadmin
3. BASIC AUTH details

Logic

1. Data Model
2. Table creation in Database
3. View & Store procedure

# Implementation - SOLUTIONS ( API Container)



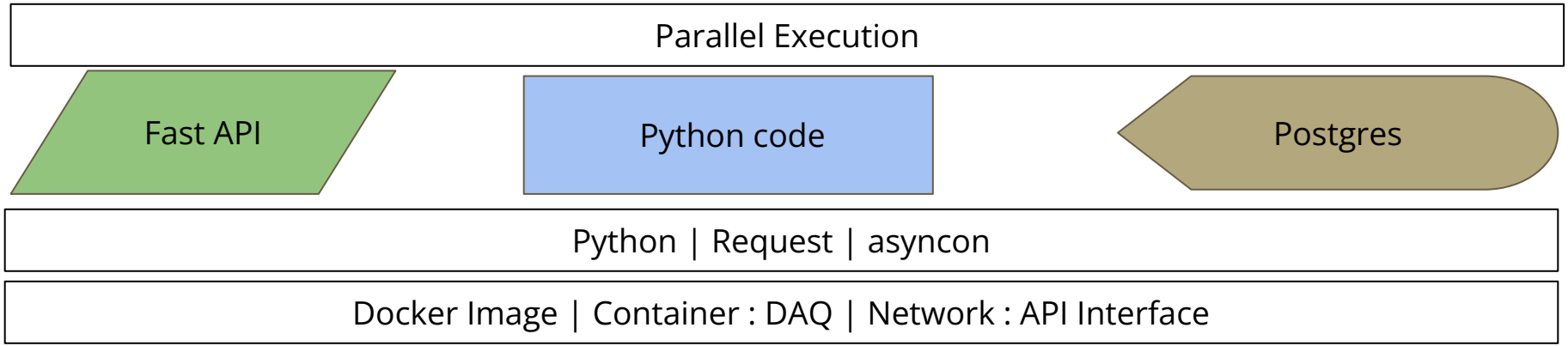

Component require to build this docker container

1. Docker - Ubuntu Image + Fast API
2. Oauth / Token

Logic

1. Database Connection
2. Create GET method for full fetch
3. Create GET method for incremental load
4. Expose the api to web server

# Implementation - SOLUTIONS ( DAQ)




Component require to build this docker container

1. Docker - Ubuntu Image + python
2. Oauth / Token

Logic

1. Read the data from API
2. Use full batch API for full load
3. Use incremental batch API for incremental load with datetime input
4. Prepare CDC operations
5. Stored in database.

# Coding

1. Implementation - SOLUTIONS (Docker + Python) :
   https://github.com/data-and-architecture/api_dataanalytics_dataload/tree/main/source

2. Implementation - SOLUTIONS (Postgres):
   https://github.com/data-and-architecture/api_dataanalytics_dataload/tree/main/db

3. Implementation - SOLUTIONS ( API Container)
   https://github.com/data-and-architecture/api_dataanalytics_dataload/tree/main/api

4. Implementation - SOLUTIONS ( DAQ)
   https://github.com/data-and-architecture/api_dataanalytics_dataload/tree/main/DataEngineering

# API

- Programming Interface to expose and receive the data through Internet (WEB) to / from outside world in a application.
- Architecture style to implement API
  - REST
  - SOAP
  - XML - RPC
- REST
  - Using Representational State Transfer design principle
  - Flexible lightweight way to implement microservices ( Web Services)

Reference :
https://www.ibm.com/topics/rest-apis
https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

# REST API

- Architectural Constraints ( Design Principle)
  - Uniform Interface ( Data belonging to only one URI)
  - Client-Server Decoupling ( Client know only URI)
  - Stateless ( Not required server side sessions)
  - Cacheability
  - Layered system Architecture
  - Code on Demand
- Using HTTP protocol
- Can data be in JSON , HTML , XLT , PYTHON , PHP or Plan text

Reference :
https://www.ibm.com/topics/rest-apis
https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

# REST API METHODS

- CURD ( supporting methods for create , update , read and delete)
  - GET
  - POST
  - PUT
  - DELETE
- Response code ( 1st digit : Class of Response)
  - 100 to 199 - information
  - 200 to 299 - Success
  - 300 to 399 - Redriect
  - 400 to 499 - Client Error
  - 500 to 599 - Server Error

Reference :
https://www.ibm.com/topics/rest-apis
https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

# OPEN API(3.0) and SWAGGER(2.0)

1. Standards formats for definitions
2. The OpenAPI Specifications provides a formal standard for describing HTTP APIs
3. The OpenAPI Specification (OAS) - exactly this transfer of knowledge from API provider to API consumer.
4. Standard of creating API's is open API.
5. swagger is the open API specification
6. Swagger definition will use YAML or JSON formats to create the API

https://www.openapis.org/
https://swagger.io/specification/

# Concepts DOCKER

1. A Platform as a service product
2. Containerization (Computing )
3. OS level virtualization
4. Can share the same OS for different application
5. Softwares
    a. Docker desktop
    b. Dockers Hub