FSA Data Engineering

Assignments

Nguyen Hong Phi - PhiNH14@fpt.com

1. Online extraction vs offline extraction

Aspect	Online Extraction	Offline Extraction
Timing	Real-time or near real-time	Scheduled (batch processing)
Latency	Low	High
Source System Load	Higher (due to continuous interaction)	Lower (scheduled during off-peak)
Data Updates	Incremental	Bulk
Use Cases	Fraud detection, personalization	Reporting, historical analysis

2. ETL, ELT. When to use? Why?

Aspect	ETL	ELT
Transformation Timing	Before loading into the target	After loading into the target
Target System	Traditional data warehouses	Modern data lakes/cloud warehouses
Processing Power	Requires ETL tools resources	Uses target system's resources
Data Structure	Predefined schema (schema-on-write)	Flexible schema (schema-on-read)
Advantages	- Data Quality Control: Ensures data is clean, consistent, and meaningful before loading. - Reduced Target Load: Since data is already transformed, the target system experiences minimal processing overhead.	- **Flexibility**: Raw data remains accessible for various transformations and use cases. - Scalability: Designed for big data workloads with modern distributed systems.
Disadvantages	- Limited Scalability: Not ideal for big data scenarios where scalability and **flexibility** are critical.	- Data Quality Challenges: Requires careful management to ensure raw data is transformed correctly later.

2. ETL, ELT. When to use? Why?

Aspect	ETL	ELT
When to use?	- Regulated Industries: Where compliance mandates data transformation before storing it. - Data Quality Prioritized: When ensuring high data quality before storage is critical.	- Big Data Analytics: For processing large datasets where scalability and speed are critical. - Exploratory or Ad-Hoc Analytics: Where flexibility is needed to support diverse queries.

3. Airflow with Docker (Github)

- **Dockerfile:**
  - Base image: `apache/airflow:2.10.3`
  - Installed **AWS CLI**
  - Successfully use AWS CLI inside containers with credentials stored in .env file
  - Installed python dependencies: requests, pyspark
  - Successfully installed and set up JAVA for running Spark inside the container
- **docker-compose.yml:**
  - Used ***Anchors & and Reference <<: ****
  - Used ***container components:***
    - environment
    - volumes
    - user
    - depends_on
    - expose
    - healthcheck
    - ports
    - command
- Modified **airflow login credentials** using .env

3. Airflow with Docker (Github)

- **Airflow:**
  - Task extract & transform: `PythonOperator`
    - Extract data of 10 books from tiki.vn, using requests
    - Transform with Pyspark (standalone mode)
    - Write to local file system
  - Task upload to Amazon S3: `BashOperator`
    - Load data saved in local file system to S3 using AWS CLI

3. Airflow with Docker (Github)

- .env

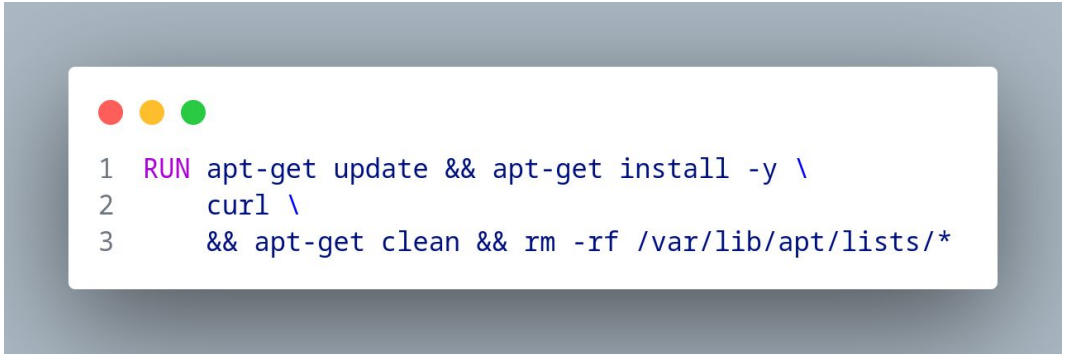
```
AWS_ACCESS_KEY_ID=[redacted]
AWS_SECRET_ACCESS_KEY=[redacted]
AWS_DEFAULT_REGION=ap-southeast-1
AWS_OUTPUT_FORMAT=json
AIRFLOW_UID=1000
AIRFLOW_GID=0
AIRFLOW_IMAGE_NAME=phinguyen/airflow:latest
_AIRFLOW_WWW_USER_USERNAME=phinguyen
_AIRFLOW_WWW_USER_PASSWORD=phinguyen
```

4. Why docker-compose?

- **Key benefits**
  - ***Simplified control:***
    - allow us to define & manage multiple-container applications in a single YAML file.
    - simplify the complex task of orchestrating & coordinating various services
  - ***Efficient collaboration:***
    - *docker-compose.yml* files are easy to share among teams
  - ***Portability across environments:***
    - Compose supports variables -> customize the composition for different environments

5. How to reduce the size of Docker images, containers?

- **Use small base image**: alpine (5Mb) or *-slim instead of full-featured images.
- **Use multistate-build**: Separate build-time dependencies from runtime dependencies to avoid including unnecessary files in the final image (example in next slide)
- **Remove Unnecessary Files**: `RUN rm -rf awscliv2.zip aws`
- **Clean Up Temporary Files**:
  - Remove build-time caches and temporary files in the same RUN layer to keep the image clean.



```dockerfile
FROM node:lts-alpine as builder
WORKDIR '/app'

# npm need package.json to install needed dependencies to build the app
COPY package.json .

# install dependencies listed in package.json
RUN npm install

# copy codebase of the app
COPY . .

# build the app using the codebase
# the result of this command is /app/build/ folder (and its content), nginx needs only this to run the app
RUN npm run build

FROM nginx

EXPOSE 80

# just copy the /app/build of the previous image (and remove the rest: node, npm, dependencies, ...)
# to limit the size of the result image
COPY --from=builder /app/build /usr/share/nginx/html

# now we have the image with nginx and /app/build, this is enough for running the application in production
# nginx image has the default command of starting nginx server
```