1 Online extraction and Offline extraction.

Question 1: What are the differences between online extraction and offline extraction?

Answer: Online extraction and offline extraction are two methods of extracting data based on their sources, timing, and how the data is processed and consumed.

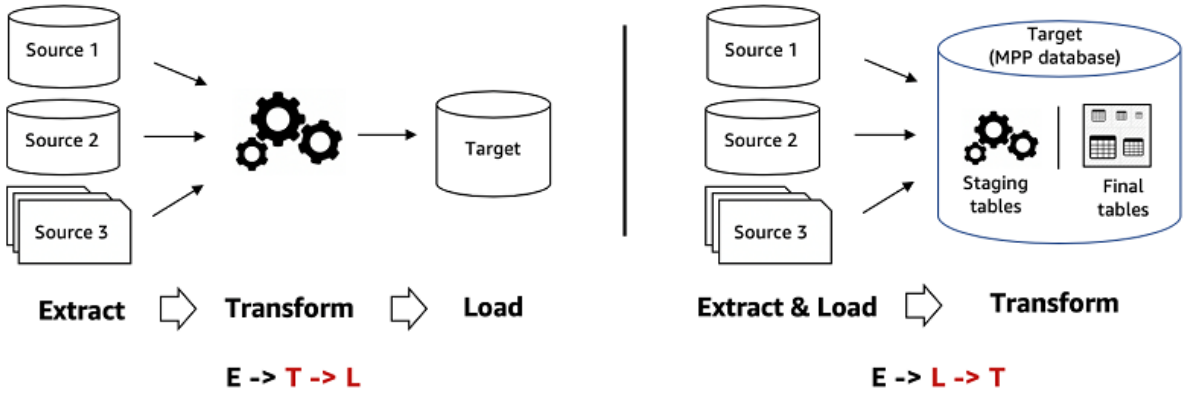
Aspect	Online extraction	Offline extraction
Timing	Real-time / Near real-time	Scheduled intervals for batch processing
Latency	Low-latency, highly available	High-latency
Volume	Small chunks of data	Large amounts of data
Use Cases	Fraud detection, real-time dashboards	BI reports, trend analysis, ML models
Architecture	Real-time infrastructure	Batch-oriented architecture
Tools	Kafka, RabbitMQ, Spark, real-time APIs	Hadoop, Spark, ETL pipelines

2 ETL and ELT.

Question 2: What are the differences between ETL and ELT. When and why we should use each?

Answer: In ETL (Extract, Transform, Load) process: data is extracted, transformed in a staging area, and then loaded into the destination system, typically a data warehouse. Otherwise, in ELT (Extract, Load, Transform) process: data is extracted, loaded directly into the destination system (e.g., cloud data warehouse), and then transformed there using its processing power.

Hình 1 ETL and ELT process, source: AWS



Aspect	ETL	ELT
Definition	Extract, Transform and Load	Extract, Load and Transform
Transform and Load position	In a secondary processing server	In the target data warehouse
Suitable for	Structured data	Semi-structured and Unstructured data
Velocity	Slower than ELT	Faster than ETL because ELT can make use of the internal resources of the data warehouse
Cost	Can be time-consuming and costly to set up, depending on the ETL tools used	More cost-effective, depending on the ELT infrastructure used

$\Rightarrow$ We should choose ETL for more *controlled, predefined pipelines* and choose ELT for *flexibility and scalability* in modern architectures.

3 Docker.

Question 3: Deploy one or more docker containers that can be run with airflow, ETL (using python, pandas...), and can connect to datasources and clouds. Guideline:

(i) Install Docker.

(ii) Install python, airflow (prefect, or dagster), selenium, requests, beautifulsoup, AWS or Azure CLI (if it is too easy for you, install pyspark) on a single container, or on different containers (it depends on how you orchestrate and use these services).

Answer:

4 Docker-compose.

Question 4: Why docker-compose?
Answer:

- Simplified control: Docker Compose allows you to define and manage multi-container applications in a single YAML file. You can start, stop, or restart an entire application stack with a single command (*docker-compose up* or *docker-compose down*).
- Efficient collaboration: Compose ensures that everyone on your team works in the same environment, avoiding "it works on my machine" problems. *docker-compose.yml* file makes it easy to replicate environments across local development, staging, and production.

- Rapid application development: Compose caches the configuration used to create a container. When you restart a service that has not changed, Compose re-uses the existing containers. Re-using containers means that you can make changes to your environment very quickly.
- Portability across environments: Compose supports variables in the Compose file. You can use these variables to customize your composition for different environments, or different users.
- Extensive community and support: Docker Compose benefits from a vibrant and active community, which means abundant resources, tutorials, and support. This community-driven ecosystem contributes to the continuous improvement of Docker Compose and helps users troubleshoot issues effectively.

5 Multistaging-build.

Question 5: How to reduce the size of Docker images, containers? Keyword: multi-staging build.

Answer: A multi-stage build allows you to use multiple FROM statements in a single Dockerfile. It separates the build environment from the runtime environment, ensuring that only the essential files are included in the final image. We use Multi-Stage Builds to reduce Docker image size by:

- Installing only production dependencies:

```
pip install --no-dev
```

- Minimizing layers:

```
RUN apt-get update && apt-get install -y \
    build-essential libffi-dev && \
    apt-get clean && rm -rf /var/lib/apt/lists/*
```

- Using Slim or Alpine base images: use `python:3.9-slim` or `alpine` instead of full-sized images like `python:3.9`.
- Removing unnecessary files:

```
RUN pip install --no-cache-dir -r requirements.txt
RUN rm -rf /tmp/* /var/tmp/*
```

- Including only required files:

```
node_modules
.git
tests
*.log
```