**Ex1: Online extraction, offline extraction.**

Online extraction is when you extract data directly from a source system in real-time. This means you can establish a live connection to the data source and can access the data as it is right now. Online extraction is useful when you need the most up-to-date information, like for real-time analytics or monitoring. This method is efficient for immediate data needs because it does not require any external storage.

Offline extraction is when you extract data from a copy or a separate place that is not the live data source. This can include data stored in files, logs, or databases that were created earlier through extraction routines. Offline extraction is useful when real-time access is not necessary, allowing you to process data in batches at schedule times, like generating weekly reports or analyzing historical trends. This method often requires more time for data preparation.

|  |  |  |
| --- | --- | --- |
| Feature | Online Extraction | Offline Extraction |
| Connection Type | Directly connect to the source system to get data | Retrieve data from an external storage |
| Data Freshness | Access real-time data | Access previously saved data |
| Use Cases | Used for real-time analytics and monitoring | Used for batch processing and scheduled reports |
| Infrastructure Needs | Need a stable connection to work well | Can work with saved data without a live connection |
| Efficiency | Efficient for getting current data | May be slower due to batch processing |

**Ex2: ETL, ELT. When to use? Why?**

ETL and ELT are methods used in data engineering to move and process data from various data sources into a storage system but they do so in different ways.

ETL stands for Extract, Transform, Load. ETL is a traditional data integration process that involves:

* Extract: Collecting data from different sources such as databases, APIs, or files.
* Transformation: Cleaning and transforming the extracted data into a desired structure or format. This can include changing data types, removing duplicates, or aggregating data.
* Loading: Loading the transformed data into a target system, typically a data warehouse.

When and why to use ETL

* When to use: ETL is good for
  + Small data sources that require complex transformations.
  + Cases when you want to make sure the data is accurate and safe before loading it.
  + Cases when there are strict rules about how data should be handled.
* Why use it: ETL helps ensure that only clean and well-organized data is loaded into the warehouse. This method is well-suited for structured data and small data sources.

ELT stands for Extract, Load, Transform. ELT is a modern approach that involves:

* Extract: Similar to ETL, data is gathered from various sources.
* Load: Raw data is directly loaded into the target system without transforming.
* Transform: Once the data is in the target system, transformations are performed as needed.

When and why to use ELT

* When to use: ELT is better when
  + Working with large volumes of unstructured or semi-structured data.
  + The target system has sufficient processing power (like cloud-based systems) to handle transformations efficiently.
  + Flexibility is required to adapt transformation logic later based on evolving business needs.
* Why use it: ELT allows for faster loading of large datasets and gives you more flexibility since you can decide how to transform raw data after it stays in the system.

|  |  |  |
| --- | --- | --- |
| Feature | ETL | ELT |
| Process Order | Extract 🡪 Transform 🡪 Load | Extract 🡪 Load 🡪 Transform |
| Data Transformation | Before loading into the data warehouse | After loading into the data warehouse |
| Data Handling | Best for structured data | Handle both structured and unstructured data |
| Speed | Low due to preprocessing | Faster due to direct loading |
| Flexibility | Less flexible, require transformations defined | More flexible, transformations can be defined later |

**Ex4: Why docker-compose?**

Docker Compose is a tool that helps you manage multi-container applications. You can define multiple containers (like web server, database, and API) in a single configuration file called “docker-compose.yml” which describes how each container should be built and how they interact with each other. Docker Compose also simplifies the process of defining and running these applications, allowing you to automate the setup and configuration of various services that work together. With Docker Compose, you can start, stop, and manage the defined containers with single command.

Why we use Docker Compose:

* Simplifies Development: If your application requires several services to run (for example, a web server, database, and caching layer), Docker Compose makes it easy to manage them all together. You don’t have to remember complex commands for each container because everything is defined in one file.
* Consistency Across Environments: Using Docker Compose ensures that everyone on your team runs the same setup. When someone clones your project repository, they can start the application with the same configuration by running a single command.
* Easier Configuration Management: You can specify environment variables, storage options, and network settings in the “docker-compose.yml” file, making it clear what each service needs to run properly.
* Supports Development and Testing: Docker Compose is great for local development, testing, and staging environments. It allows developers to quickly spin up their required services without worrying about individual setups.
* Automatic Networking: Docker Compose automatically creates a network for your containers, allowing them to communicate with each other easily without extra configuration.

**Ex5: How to reduce the size of Docker images, containers?**

Reducing the size of Docker images and containers is essential for improving performance, saving storage, and speeding up builds and deployments. There are several ways to minimize the size of Docker images and containers:

1. Use lightweight base images: choose minimal base images like alpine which are significantly smaller than full operating systems like ubuntu or Debian
2. Multi-stage builds: use multi-stage builds to separate the build environment from the final runtime environment, leaving only the necessary files in the final image
3. Remove unnecessary files: delete temporary files, cache, and logs that are not needed in the final image
4. Combine commands to reduce layers: combine run commands into a single line to reduce the number of layers in the image
5. Avoid installing unnecessary packages: install only the packages required for the application
6. Use .dockerignore: exclude unnecessary files and directories (for example .git logs, temporary files) from the Docker build context.
7. Use minimal application dependencies: audit and remove unused dependencies in your application before packaging it into the Docker image.
8. Use cache strategically: use Docker’s layer caching to avoid rebuilding unchanged layers
9. Use compressed or prebuilt binaries: replace heavy runtime environment with prebuilt static binaries
10. Use specific tags: use specific image tags (for example python:3.9-slim instead of python:latest) to avoid bloated images and ensure predictable builds
11. Clean up after installation: remove cache, unused files, and temporary files after package installation