A GUIDE TO

ETL vs ELT data pipelines
Analysts and data scientists use SQL queries to pull data from the data storage underbelly of an enterprise. They mold the data, reshape it, and analyze it, so it can offer revenue-generating business insights to the company.

 BUT ANALYTICS IS ONLY AS GOOD AS THE MATERIAL IT WORKS WITH.

That is, if the underlying data is missing, compromised, incomplete, or wrong, so will the data analysis and inferences derived from it.

To understand the end result - data - we need to understand the different data pipeline processes that bring the data to life. Or at least to the analysts' hands.
1. ETL process deep-dive: design, challenges, and automation
   1.1 The benefits of a well-engineered ETL process
   1.2 Extract explained
      1.2.1 Extract architecture design
      1.2.2 Extract challenges
   1.3 Transform explained
      1.3.1 Transform architecture design
      1.3.2 Transform challenges
   1.4 Load explained
      1.4.1 Load architecture design
      1.4.2 Load challenges
2. ETL vs ELT: 11 Critical differences
   2.1 What is the difference?
      2.1.1 ETL process overview
      2.1.2 ELT process overview
   2.2 ETL vs ELT: 11 critical differences
      2.2.1 Data structure in storage
      2.2.2 Data volumes
      2.2.3 Pipeline speed
      2.2.4 Flexibility
      2.2.5 Scalability
      2.2.6 Implementation
      2.2.7 Maintenance
      2.2.8 Compliance
      2.2.9 Storage requirements
      2.2.10 Technology
      2.2.11 Maturity
   2.3 Which one to pick?
3. Build scalable ETL and ELT pipelines with confidence
   3.1 How can Keboola help with ETL data pipelines?
   3.2 How does ETL/ELT pipeline build look like in Keboola?
   3.3 What sets Keboola apart?
1. ETL process deep-dive: design, challenges, and automation

The Extract-Transform-Load process (ETL for short) is a set of procedures in the data pipeline. It collects raw data from its sources (extracts), cleans and aggregates data (transforms), and saves the data to a database or data warehouse (loads), where it is ready to be analyzed.

*The following diagram represents the data flow through the different ETL stages.*

![Diagram of data flow through ETL stages]

1.1 The benefits of a well-engineered ETL process

The ETL process is engineered in such a way that your data pipelines and analytics provide *business* value to your company. A well-engineered ETL pipeline has several benefits:

01. **Information clarity.** During ETL transformations, data is cleaned and joined across sources before it is saved in the database, where you can then analyze it. These operations allow you to work with clear information and disambiguate unclear raw data.

02. **Information completeness.** A well-designed ETL pipeline includes all of the business sources which are relevant to your operations in a single place (the destination [data warehouse/database](#)). All of the information is complete, so there are no missing puzzle pieces.

03. **Information quality.** ETL processes validate data at extraction or correct/discard data at the transformation. This ensures that the quality of data is
always controlled before it is analyzed, thus increasing trust in the analysis and giving you the confidence to use data for business intelligence and data-driven decision-making.

**04. Information velocity.** ETL processes can be designed to trigger the entire ETL pipeline whenever new data arises in the sources or when existing data is changed. You can therefore control the ‘freshness’ of the data, as well as the speed at which you make decisions based on signals in the outside world.

**05. Novel business insights.** The entire ETL process brings structure to your company’s information. This allows you to spend more time analyzing novel questions and acquiring new insights, rather than trying to perform procedures to get valuable data at each stage.

Let’s deep dive into each of the ETL stages and understand the engineering tradeoffs and considerations.

### 1.2 Extract explained

The “Extract” stage of the ETL process involves collecting data from its data sources. This data will ultimately lead to the rows and columns of your analytic database.

Traditionally, extraction meant getting data from Excel files and Relational Management Database Systems, as these were the primary sources of information for businesses (e.g. purchase orders written in Excel).

With the increase in Software as a Service (SaaS) applications, the majority of businesses now find valuable information in the apps themselves, e.g. Facebook for advertising performance, Google Analytics for website utilization, Salesforce for sales activities, etc.

Today, data extraction is mostly about obtaining information from an app’s storage via APIs or webhooks.
1.2.1 Extract architecture design

When designing the software architecture for extracting data, there are 3 possible approaches to implementing the core solution:

**01. Full-extraction.** Each extraction collects all data from the source and pushes it down the data pipeline.

**02. Incremental extraction.** At each new cycle of the extraction process (e.g. every time the ETL pipeline is run), only the new data is collected from the source, along with any data that has changed since the last collection. For example, data collection via API.

**03. Source-driven extraction.** The source notifies the ETL system that data has changed, and the ETL pipeline is run to extract the changed data. For example, data collection via webhooks or change data capture.

Below are the pros and cons of each architecture design, so that you can better understand the trade-offs of each ETL process design choice:

<table>
<thead>
<tr>
<th></th>
<th>Full-extraction</th>
<th>Incremental extraction</th>
<th>Source-driven extraction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROS</strong></td>
<td>Data is guaranteed to be fresh.</td>
<td>Good balance between data freshness and computational resources is needed.</td>
<td>Lower computational resources needed.</td>
</tr>
<tr>
<td><strong>CONS</strong></td>
<td>Extremely computationally expensive.</td>
<td>Deleted records from the source could be missed by the extraction system (unless the system checks every record from the database against the source – computationally expensive).</td>
<td>Webhooks often fail, and cannot be relied on for mission-critical data collection.</td>
</tr>
<tr>
<td><strong>BEST FOR</strong></td>
<td>First extraction cycle.</td>
<td>The majority of mission-critical data.</td>
<td>Data that changes often, but is not mission-critical.</td>
</tr>
<tr>
<td></td>
<td>Short-lived data (e.g. logs).</td>
<td></td>
<td>Data for which speed of availability is more important than data quality.</td>
</tr>
<tr>
<td></td>
<td>Sources with small amounts of data.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.2.2 Extract challenges

The data extraction part of the ETL process poses several challenges. A lot of the problems arise from the architectural design of the extraction system:

**01. Data latency.** Depending on how fast you need data to make decisions, the
extraction process can be run with lower or higher frequencies. The tradeoff is between stale or late data at lower frequencies vs higher computational resources needed at higher frequencies.

**02. Data volume.** The volume of data extraction affects system design. The solutions for low-volume data do not scale well as data quantity increases. With large amounts of data, you need to implement parallel extraction solutions, which are complex and difficult to maintain from an engineering perspective.

**03. Source limits.** You need to be aware of the source limitations when extracting data. For example, some sources (such as APIs and webhooks) have imposed limitations on how much data you can extract simultaneously. Your engineers need to work around these barriers to ensure system reliability.

**04. Data validation.** Either you validate data at extraction (before pushing it down the ETL pipeline), or at the transformation stage. When validating data at extraction, check for missing data (e.g. are some fields empty, even though they should have returned data?) and corrupted data (e.g. are some returned values nonsensical, such as a Facebook ad having -3 clicks?).

**05. Orchestration.** Based on your choices of data latency, volume, source limits, and data quality (validation), you need to orchestrate your extraction scripts to run at specified times or triggers. This can become complex if you implement a mixed model of architectural design choices (which people often do to accommodate for different business cases of data use).

**06. Monitoring.** You need to monitor your extraction system on several levels:

- a. **Resources.** How much computational power and memory is allocated?
- b. **Errors.** Have any errors resulted in missing or corrupted data?
- c. **Reliability.** Have the extraction scripts run at all?

**07. Disparate sources.** Working with different data sources causes problems with overhead and management. The variety of sources increases the data management surface by increasing the demands for monitoring, orchestration, and error fixes.

With the increasing dependency on third-party apps for doing business, the extraction process must address several API challenges as well:

**01. Variability & inconsistency.** Every API is designed differently, whether you are using apps from giants like Facebook or small software companies. The
variability and inconsistencies between their designs will cause your engineers to spend a lot of time implementing the extractors for each source.

**02. Lack of documentation.** APIs are notoriously lacking in proper documentation. Every time your use case deviates from the core minimum (which is documented), engineering hours need to be allocated to understand how to execute your vision for the extractor within the API.

**03. (Breaking) Changes.** APIs change frequently and break your extractors. Whether it’s because of version updates or expiring access tokens, APIs require a lot of monitoring and maintenance hours.

**04. High complexity.** APIs expose data in multiple locations (endpoints) and complex aggregations. This is especially true for SaaS apps, which cover multiple business use cases and collect a lot of different data. You need to budget engineering hours for navigating this complexity.

Before you even start extracting data from its raw sources, you need to consider several challenges and design or architect your data engineering solutions to them.

But once you successfully extracted the data, you can move to the Transform stage of the ETL data pipeline.

**1.3 Transform explained**

The “Transform” stage of the ETL process takes the data that has been collected at the extractor stage and changes (transforms) it before saving it to the analytic database. There are multiple transformations:

**01. Data cleaning.** Data cleaning involves identifying suspicious data and correcting or removing it. For example:

- Remove missing data
- Recode missing data into NULLs or 0s or “#NA”
- Remove outliers
- Recode different versions of the same data to a common denominator. For example, “M”, 1, “male”, “masculine” to “Male”
- One-hot encode categorical data
• Convert data types to standard forms. For example, convert DateTime objects and Unix timestamps to the same data type

02. Data enriching. Data enriching involves adding new information to the raw data already collected.
• Join different sources. For example, create customer information blobs, which join information from a variety of purchasing apps.
• Deduplication. Identify which information is duplicated and remove the copycat.
• Calculated fields. For example, calculate the lifetime value of the customers at import, or the number of their consecutive purchases.

Transforming data means preparing data to be in the right shape, format, and size for analysts to quickly dive in and extract insights from the data without having to first spend hours cleaning data.
Check out our extensive guide to data cleaning for a comprehensive deep dive into the subject.

1.3.1 Transform architecture design
When designing the architecture of data transformation, there are multiple things to consider:

01. Order of operations. The order in which transform rules are applied to incoming data can affect the end result. For instance, imagine we have two transform scripts. The first one processes data to compute the consecutive number of purchases made by a customer. The second transformation process drops purchase information from the data pipeline unless there is a shipping address. If we drop the row for a customer with a missing shipping address before we calculate the consecutive order, the end result is going to be two different purchase orders.

02. Business logic. Transforms often implement business logic, such as calculating a customer’s lifetime value or their consecutive orders. Your architecture needs to be designed so it can handle missing or corrupt data and transform orders, thus supporting business logic implementation.
03. **Algorithmic efficiency.** Because transforms go through the extracted data, sometimes they need to handle heavy loads. Algorithmic efficiency in the design of transforms can make a difference in the time needed for a transform to execute or whether your transform will time-out your system. As a simple example: implementing a dictionary solution for 1M rows transformation vs a for loop results in a difference of a couple of orders magnitude.

04. **Quality assurance.** Transformations are often the place where data is validated against a set of criteria (e.g. do not import customer information unless we have their email) and monitored for data quality. A lot of ETL processes are designed with alerts at this stage to notify developers of errors, as well as of rules, which prevent data passing on the data pipeline unless it matches certain criteria.

Before you devise your transform pipelines and scripts, you have to keep an eye on the specific challenges of this step to avoid any potential pitfalls.

1.3.2 Transform challenges

There are several challenges when dealing with transformations:

01. **Lack of business logic.** Oftentimes, it becomes clear that there is a lack of business logic given the data we receive from the extract phase. As an example: the business rule for determining a new customer is the date of their first product purchase. But what do we do for customers who paid for shipping, but not for a product?

02. **Changing business logic.** As company operations evolve, business definitions change. Even small changes to business logic can have multiple effects on transforms, especially if the change in one transform affects other transforms that depend on it.

03. **Hard business logic.** Sometimes, implementing something trivial from a business perspective can be challenging from an engineering perspective.

04. **Changing source data.** APIs can change their response payloads, data can become corrupted, or your system might migrate to a new SaaS… so you need to implement a different transform logic. In addition to the decoupling issues, changing source data requires constant monitoring and maintenance of the transform stage.
05. Scaling complexity. Transforms present challenges when the ETL processes evolve. The more transforms you implement, the harder it is to keep track of their mutual effects.

Once you master the solutions to the transform challenges, you can deep dive into the tradeoffs of the ETL’s load stage.

1.4 Load explained

“Load” involves taking data from the transform stage and saving it to a target data store (relational database, NoSQL data store, data warehouse, or data lake), where it is ready for analysis.

1.4.1 Load architecture design

There are three possible designs for architecting data being loaded into a destination warehouse/database. Here, we explore them alongside their pros and cons:

<table>
<thead>
<tr>
<th></th>
<th>Full load</th>
<th>Incremental batch load</th>
<th>Incremental stream load</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAIN MECHANISM</strong></td>
<td>Dumps all data into the database at once.</td>
<td>Inserts data into the database at regular intervals.</td>
<td>Inserts data into the database when new data arises or old data is updated.</td>
</tr>
<tr>
<td><strong>PROs</strong></td>
<td>Low implementation difficulty.</td>
<td>Less time and resources are needed.</td>
<td>Less time and resources are needed.</td>
</tr>
<tr>
<td><strong>CONs</strong></td>
<td>Needs more time and resources.</td>
<td>Medium expertise is needed.</td>
<td>High expertise is needed.</td>
</tr>
</tbody>
</table>

Irrespective of the architecture you chose, there are multiple challenges associated with the load stage that you should consider.
1.4.2 Load challenges

There are several challenges in the loading stage of the data pipeline:

01. **Order of insertion.** The order of insertion can affect the end result. If a table has a foreign key constraint, it might prevent you from inserting data into that table (and would probably skip it), unless you first insert matching data in another table.

02. **Schema changes.** The schema represents what the destination (database or data warehouse) expects the data to look like. As your business evolves, the schema is often updated to reflect changes in business operations. The resulting need for schema updates can lead to a waste of engineering hours, as well as unintended consequences for the entire system (e.g. data quality validations might break when the form of data breaks).

03. **Data quality.** Suspicious data is sometimes formatted in such a way that it circumvents all of your data validation at extraction and transformation. As a result, you need additional data quality monitoring to assure data quality in your database or data warehouse.

ETL is a comprehensive and mature data paradigm for providing analysis-ready data to stakeholders. But with the advent of big data, a new - and different - paradigm has entered the world of data engineering: ELT.

2. ETL vs ELT: 11 Critical differences

ETL and ELT refer to two patterns of data storage architecture within your data pipelines.

Both ETL (extract, transform, load) and ELT (extract, load, transform) processes help you collect data, transform it into a usable form and save it to permanent storage, where it can be accessed by data scientists and analysts to extract insights from the data.
2.1 What is the difference?

Well, the obvious distinction is in the order of operations: the ETL process transforms the data before committing it to a data storage, while the ELT design pattern prioritizes storage before the transformation logic is applied.

This tinny difference acts as a domino effect. Based on the architecture pattern you chose, your data pipelines offer different benefits and are optimized for different use cases. In this chapter, we will dive deeper into understanding the ETL process, ELT process, how the two differ, and which one to choose for your operational data needs.

2.1.1 ETL process overview

In case you missed Chapter 1, here's a short recap. The ETL process collects the raw data from various data sources (your CRM, ad accounts, ERP, email servers, ...) and saves them to the staging area.

Before data can be loaded in the target data warehouse or database of your choice, the data undergoes extensive transformations.

Depending on your business logic, you might mask sensitive personal information, remove outliers, or aggregate metrics to make your analysts' life easier, before finally loading data into the data storage.

2.1.2 ELT process overview

The ELT process is similar to the ETL one. There is also data extraction, data transformation, and data loading. But unlike the ETL process, ELT loads all the data into a data lake. Only later, you can apply transformation logic to the data before moving it to a data warehouse.

To re-iterate – the ETL process extracts data to a staging area and carefully picks what data gets loaded further, while the ELT process extracts all data, and only later applies the needed transformations.
2.2 ETL vs ELT: 11 critical differences

There are 11 crucial differences between ETL and ELT processes:

2.2.1 Data structure in storage
ETL processes store only structured - aka relational - data. In contrast, the ELT process stores all types of data structured as they appear in the source data. That includes structured data, but also semi-structured and unstructured data like raw data files (XML, JSON), videos, images, audio clips, sensory telemetry, etc.

Keep in mind that both processes can extract unstructured or semi-structured data. But only ELT saves the data in an unstructured or semi-structured format.

(Read more about the differences between structured and unstructured data, and why those differences matter for your data operations)

2.2.2 Data volumes
ETL usually operates in the ranges of MB or GB, while ELT works with orders of magnitude higher data volumes (PB, TB). This is why ELT is the preferred choice for big data applications.

2.2.3 Pipeline speed
Because ETL needs to transform data before loading it, the entire process experiences higher latency than its ELT counterpart.

Transformations can be extremely time-consuming, especially if they require multiple passes over the data, such as in complex aggregate queries, or complex data cleansing, such as transforming unstructured data into structured data.

This makes ELT pipelines faster.
2.2.4 Flexibility
Engineers building ETL pipelines specify in advance the data constraints applied before loading the data into the system. This makes the ETL process less flexible. The data in ELT, on the other hand, is loaded first, transformed later. This makes the ELT paradigm much more flexible.

For example, if you realize your machine learning algorithm needs an additional field, you can just re-run a slightly modified pipeline from the loaded data. In contrast, the ETL process would not give you this option, since non-approved data was never loaded in the database, to begin with.

2.2.5 Scalability
ETL scales slower than ELT since the transformation layer acts as a bottleneck. Simply saving data is fast. But if you transform it before loading it, you could delay data ingestion. This delay increases with sudden surges of data volumes or velocities, aka, whenever your system would need to scale.

2.2.6 Implementation
ETL incurs higher implementation costs than ELT, because the design of the target data warehouse, as well as the necessary transformations, need to be set in place before you start importing data.

2.2.7 Maintenance
Maintenance costs can be high in both systems, but they occur at different stages of the process.
With ETL, a lot of the transformation logic is applied in the extract phase. You only collect data you will need downstream, to speed up the entire pipeline. This causes maintenance overhead since raw data sources change all the time, and engineers need to re-write extraction scripts.

On the other hand, maintenance of ELT processes happens in two locations. The first load data storage can quickly become unmanageable because it acts as a dump. Regular cleanups and documentation efforts are set in place to tame the load beast.
Additionally, the same issues of changing raw data affect the ELT process. But maintenance of changing source data is handled by re-engineering transformation pipelines after loading the data. This gives your engineers more flexibility (no data is lost if a transformation script fails to adjust to the new incoming data structure), but it does not save them from the maintenance work.

2.2.8 Compliance
Regulatory compliance - such as GDPR, HIPAA, or CCPA - sets high standards in place when working with personal or sensitive data.

The ETL pipeline is easier to keep compliant. Sensitive data is either never imported into the target data warehouse or is masked before importing (e.g. an IP mask occludes the last octet to prevent identification).

ELT is much more prone to issues surrounding data privacy. Because all data is loaded, irrespective of its sensitive nature, ELT architects and engineers need to work around to clock to secure the data from privy eyes. Roles-based access control and data security become even more important and time-consuming to guarantee similar levels of regulatory compliance as in the ETL counterpart.

2.2.9 Storage requirements
ELT requires orders of magnitude more storage than the ETL process. Saving all the raw data in its non-filtered form increases the requirements for storage space.

The cost of storage is lower in ETL.
But with the advent of the cloud as a service, commodified storage is cheap, and the cost advantage of ETL over ELT is diminishing by the day.

2.2.10 Technology
The technology underpinning ETL processes revolves around databases and online analytical processing (OLAP) cloud data warehouses. Notable examples of data warehouses include Amazon Redshift, Snowflake, and Google BigQuery.

On the other hand, ELT handles the fast and massive data loads, by relying on a new technology called the data lake. Data lakes are specialized data storages that
can handle massive and fast loads. Examples of data lakes include Amazon S3, Apache Hadoop, and Microsoft Azure Data Lake Storage (ADLS). Read more about the differences between data lakes and data warehouses.

2.2.11 Maturity
ETL has been around for multiple decades and is much more mature. From tried-and-tested architecture patterns to devoted ETL tools, the ETL process is much more mature than its ELT counterpart.

This carries two consequences:

01. Availability of talent and tools is easier to source in ETL paradigms. Nowadays, it is much easier to find engineers who can handle an ETL pipeline than engineers who are skilled in ELT approaches.

02. Security can be a concern. The security vulnerabilities of the ETL pipelines are much more known. On the other hand, ELT can be more dangerous, because non-sanctioned data loading exposes Enterprise’s resources to malicious attackers.
Based on the 11 crucial differences, which paradigm should you choose?

<table>
<thead>
<tr>
<th>Transform Technologies</th>
<th>ETL</th>
<th>ELT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scripting languages, SQL procedures</td>
<td>Data warehouse specific solutions</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical space required to store data</th>
<th>ETL</th>
<th>ELT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower</td>
<td>Higher</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maturity</th>
<th>ETL</th>
<th>ELT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tested and proven</td>
<td>Novel and (sometimes) experimental</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Engineering expertise required</th>
<th>ETL</th>
<th>ELT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medium</td>
<td>High</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data type</th>
<th>ETL</th>
<th>ELT</th>
</tr>
</thead>
<tbody>
<tr>
<td>All, but best for structured (relational) data</td>
<td>All, but excels at unstructured data</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pros</th>
<th>ETL</th>
<th>ELT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simpler to deploy and maintain.</td>
<td>Can handle massive amounts of data.</td>
<td>Best for unstructured data.</td>
</tr>
<tr>
<td>A lot of (human and technical) resources available.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cons</th>
<th>ETL</th>
<th>ELT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scaling - Becomes increasingly more complex for large data deployments.</td>
<td>Needs a higher level of expertise to deploy and maintain.</td>
<td>Edge cases are not always polished for reliability.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Best for</th>
<th>ETL</th>
<th>ELT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic ETL processes</td>
<td>Data experts who know what they are doing.</td>
<td></td>
</tr>
</tbody>
</table>

2.3 Which one to pick?

ETL is best suited for fast analytics in smaller-to-medium data environments, where the source data and data operations are well-controlled and do not evolve constantly (do not need flexibility).

ELT, in contrast, is best suited for working with semi-structured or unstructured data, in big data environments, where the changing data operation requirements foresee a lot of needed flexibility.

But before you make your final choice, you need to consider what tools are available to make your job easier.
3. Build scalable ETL and ELT pipelines with confidence

Keboola is a Data Stack as a Service (DaaS) platform, which covers the entire data pipeline operational cycle.

From ETL jobs (extract–transform–load) to orchestration and monitoring, Keboola offers a holistic platform for data management. The architecture is designed modularly as plug-and-play, allowing for greater customization. On top of the expected ELT/ELT features, Keboola surprises with an advanced take on the data pipeline, such as one-click deployments of digital sandboxes and machine learning out-of-the-box features.

3.1 How can Keboola help with ETL data pipelines?

With Keboola, you can build, maintain, and automate your ETL/ELT pipelines with a couple of clicks:

01. Automate data collection from third-party apps (and databases) with over 200 Extractors (components that help you Extract data from its sources).

02. Automate data cleaning and transformations with Transformations and Applications (bonus: it comes with data versioning).

03. Flow data between different storages with Writers. You can send your data to a database, data warehouse, or data lake of your choice. But Keboola goes beyond that. It allows you to send data to 3rd Party Apps, like Facebook Ads or SalesForce, that help you enrich your data.

04. Schedule your data pipeline tasks with Orchestration. The set-it-and-forget-it logic helps you automate and scale your ETL/ELT pipelines.
3.2 How does ETL/ELT pipeline build look like in Keboola?

Building data pipelines is made simple with Keboola.

**PREREQUISITE:** Log into your (free) Keboola account

If you do not have an account already, you can make a free one here:
**STEP 1: Connect Keboola to your data source to automatically extract data**

Use Keboola’s Extract component (app) to automatically collect data from your data sources. For example, we navigate to **Components > Directory > Search**, and query for the Facebook Ads extractor. When clicking on the “Use this” button, we simply follow along the configuration wizard to give access rights to Keboola and the platform starts collecting Facebook Ads data automatically.
**STEP 2: Clean your data**

Data cleaning can be very involved. But not with Keboola. Go to Transformations > New Transformation to build a script that will clean the data you collected in the previous step:

You can pick and choose the transformation scripting language of your choice. For this example, we will join the Facebook Ads data we extracted with the Google Ads data.

Within the transformation step, we define (1) the name of the transformation, (2) what is the input data (hint: Facebook Ads and Google Ads data we extracted in the previous step), (3) the destination table where Keboola should save our clean data, and lastly (4) a simple SQL query that can do the transformation for us:
The data is automatically loaded into Keboola’s Snowflake, without you having to do anything else - no deployment, no maintenance, no additional heavy-lifting.

You can even automate the transformation script to automatically trigger every time new Facebook Ads and Google Ads data is extracted.

But let us say you want to go beyond the ETL pipeline, and move the cleaned data to a visualization software or another destination. What do you do?
**STEP 3: Send data to the destination of your choice**

The same way we extracted and transformed data with the use of simple components, Keboola allows you to integrate data into other data storages and 3rd Party Apps with just a couple of clicks.

For example, let us say we want to visualize Advertising data in a dashboard, to see

![Dashboard with components](image)

When we select PowerBI, Keboola’s wizard guides us through the configuration necessary, so we can send the cleaned data to Power BI and visualize it with the graph of our choice.

As you can see, the entire data pipeline is built and deployed in minutes. You can even automate it, so it always runs on a schedule to provide you with fresh data where you want it (pick your destination) and when you want it (on any schedule).
3.3 What sets Keboola apart?

Keboola is the right choice for your ETL tools for multiple reasons:

- **From idea to ETL deployment in minutes.** With the plug-and-play features, Keboola allows you to build ETL pipelines in minutes, instead of hours or weeks.
- **State-of-the-art architecture.** The engineering behind Keboola is superb. It is resilient, scales effortlessly as your data needs grow, and uses advanced security techniques to keep your data safe.
- **Fully customizable.** Keboola supports a variety of scripting languages (SQL, Python, R), giving you the option to fully customize the data pipeline on your own language terms. Additionally, it builds on top of open-source technologies like Singer, which allows you to create your custom applications (e.g. add a rarely used data source not already covered by their extractors).
- **Audit & security.** The platform offers an exhaustively auditable data pipeline, by versioning and fingerprinting changes. This is just one of the solutions built on top of the already advanced security measures, which leverages AWS’s best-in-class security standards.
- **Beyond ETL.** Keboola offers a suite of transformative technologies built on top of the ETL: scaffolds to deploy end-to-end pipelines with a couple of clicks, data catalogs that allow you to share data between departments (break those silos) and document data definitions, and digital sandboxes that allow for experimentation and prototyping without affecting the underlying engineering.
- **No vendor lock-in.** Keboola has a very generous always.-free tier. But even if you decide you want more and subscribe, the monthly fees keep your relationship with Keboola flexible; no hidden fees, no lock-in contracts. And unlike the majority of other vendors, it is easy to take your data and scripts out of Keboola and migrate to a different solution.
- **ELT by design.** If you want to build your pipelines within the ELT architecture instead of ETL, you can easily shift your operations within Keboola at no additional engineering costs. Keboola integrates data into its own Snowflake backend before sending it to other destinations, which can act as a de-facto data lake within your ELT pipeline.
But Keboola goes beyond the data pipelines by offering out-of-the-box solutions for a variety of data needs across your data operations:

**01.** Automate and orchestrate entire (end-to-end) data pipelines by deploying them in just a couple of clicks with [Keboola Templates](#). Templates or Scaffolds are reusable recipes that help you democratize, share, and accelerate your engineering.

**02.** With [Digital Sandboxes](#), you can experiment with new data models and develop new components without breaking the underlying ETL architecture.

**03.** Organize your data into [Data Catalogs](#) for information clarity, safe sharing, and controlled multi-level access.

**04.** Deploy machine learning models within Keboola, to prototype and build next-generation data products.

**TAKE KEEBOOLA FOR A SPIN. KEEBOOLA HAS AN ALWAYS-FREE, NO-QUESTIONS-ASKED PLAN. SO, YOU CAN EXPLORE ALL THE POWER KEEBOOLA HAS TO OFFER.**

Feel free to give it a go or reach out to us if you have any questions.