The ultimate guide to modern data movement

An all-in-one guide to the Fivetran way of data movement

Fivetran

Table of contents

I. Data movement is the future

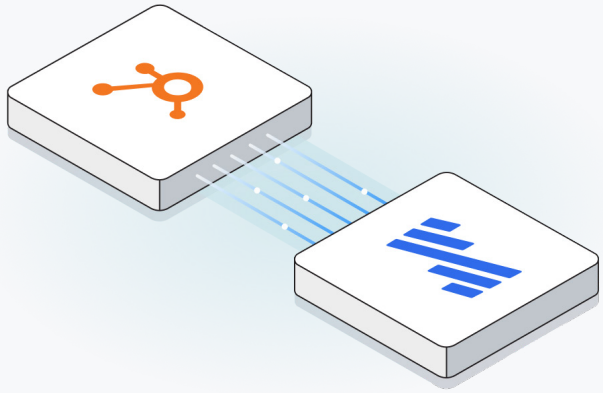
Since the advent of modern computing, organizations have sought to mine the data produced by their operations for insights to support decisions. The growth of the cloud has intensified both the need to harness data and the evolution of the tools directed to solve this problem. Use cases for data have grown from decision support to predictive modeling, direct support for large-scale operations and innovative, data-based products. These emerging use cases call for a change to the way we think and talk about the tools and processes used to prepare data for various end uses.

In light of the growing range of data use cases, data integration is needlessly limiting and is superseded by a broader conception of **data movement**. Data movement, a more general and inclusive concept than data integration, encompasses a wider range of important use cases for data.

For instance, there are operational use cases for data movement. To ensure performance, an organization might want to replicate data across operational systems that are distributed across different regions. Similarly, to prevent disruptions to operations in the event of failures, an organization might want to replicate data in real time from production servers to failover instances. Other use cases for data movement require productizing or activating data by moving data models back into applications and operational systems.

As data-related use cases become ever more complex and varied, additional concerns crop up regarding how to scale data usage safely and responsibly. These concerns include governance, security and extensibility. Answering these concerns requires a much more comprehensive suite of capabilities than can be provided through a single point solution.

In this guide, we will discuss the Fivetran approach to data movement and how an automated data movement platform can support your organization's future needs as you scale your operations and pursue new products. If you are a data professional with a hand in designing or building your organization's data architecture, particularly a data engineer or data architect, this guide is essential reading.



Data integration challenges

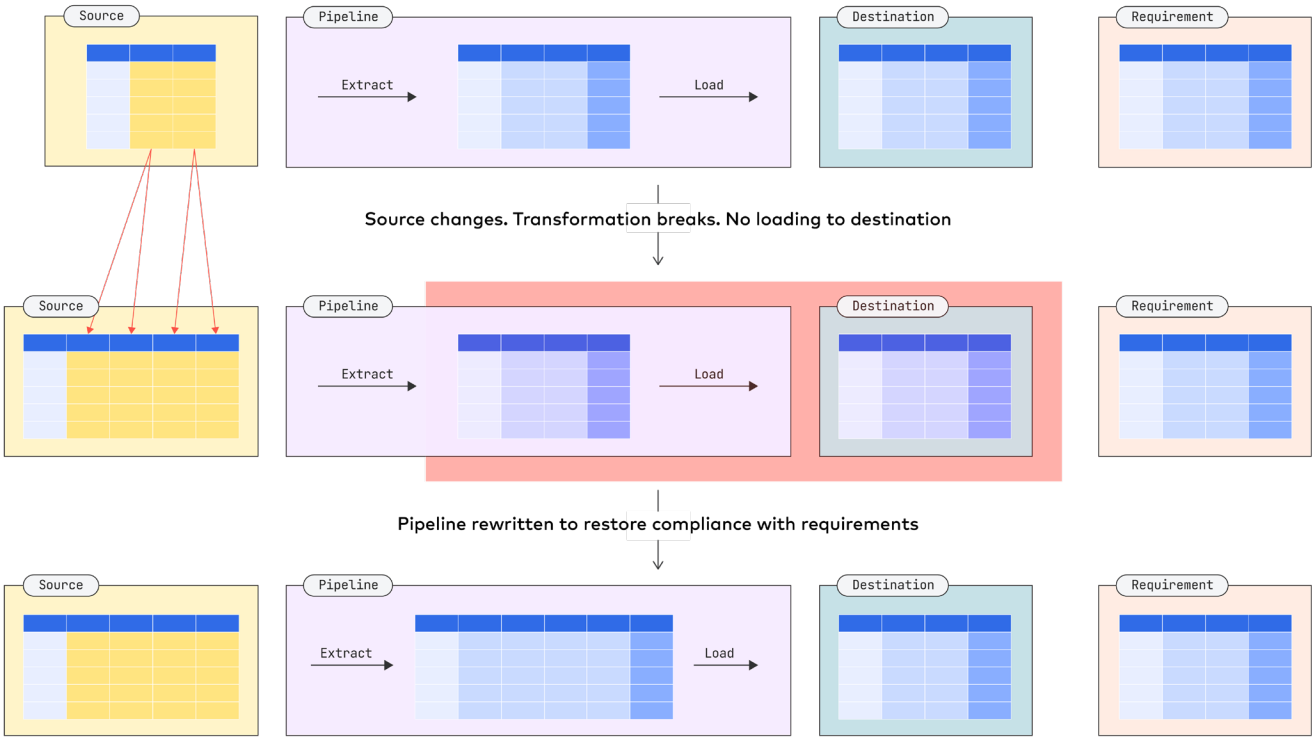
Data integration is a data management process that focuses on extracting raw data from a variety of disparate sources and combining it into a single, unified view for analysis and management. The growth of the cloud and cloud-based technologies have radically changed the realities of data integration, enabling the easy movement of data from any source to any destination.

Before the modern cloud, the standard approach to data integration was a data pipeline featuring Extract, Transform, Load (ETL), often using on-premise hardware. ETL demands considerable engineering effort in planning, design, construction and continued maintenance. This makes ETL an engineering-centric process, imposing a barrier between analysts and the functionality they depend on to model data.

Its bespoke nature makes it difficult to scale and expand, as well as slow to respond to changing business needs. It features two major failure states that require redesigns and rebuilds of the data pipeline.

1. Upstream schemas at the source may change, breaking downstream processes that depend on specific configurations of fields, tables, etc.

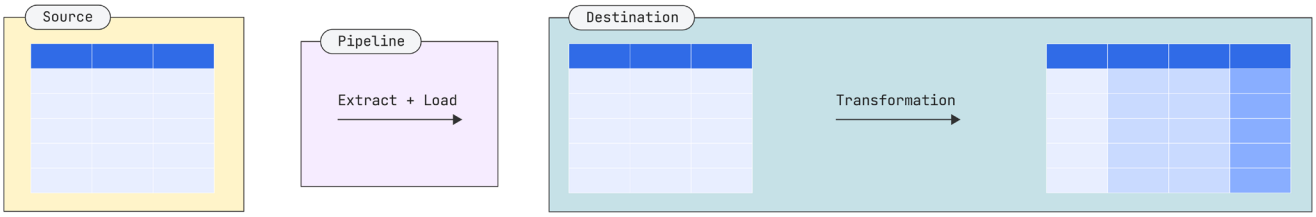
2. Downstream business and analytical needs may change, requiring new transformations to produce new data models.



These two common failure states make ETL extremely brittle, leading to downtime and stale data, absorbing precious engineering hours and limiting the effectiveness of an organization's analytics.

The automated data movement platform

The modern approach to data movement is cloud-based Extract, Load, Transform (ELT). In contrast to ETL, ELT defers transformations to the end of the process. ELT architecture loads data with minimal alteration from source to destination, while transformations are decoupled from the data pipeline and performed within the destination. This prevents the two failure states commonly imposed by ETL (i.e., changes to upstream schemas and downstream data models) from interfering with extraction and loading, leading to a simpler, more robust and more scalable data pipeline. A simpler and more robust data pipeline results in lower costs, downtime and engineering burden as well as fresher data.



There are several implications to shifting data modeling and transformations from the pipeline to the destination. The most important is that decoupling extraction and loading from transformation enables the data pipeline to be fully outsourced and automated, as there is no longer any need to build a bespoke solution for every downstream data model needed by an organization. Another implication is that transformations are now accessible to analysts. Finally, loading raw data from source to destination enables access to a singular, unadulterated source of truth that isn't obscured by transformation.

ELT is the central mechanism of a broader suite of tools and processes used to efficiently move data through the cloud, called the automated data platform. With the power of a modern data platform, organizations can not only integrate data but migrate critical data infrastructure to the cloud, promote self-service analytics and turn data assets into new products.

The modern data platform features four key pillars:

Data movement

An organization may need to move data from applications and databases to data warehouses, across clouds, from lakes to warehouses and more.

Governance

As organizations and their data grow, it becomes all the more important for data teams to understand what data is being loaded so it can be protected and monitored for compliance.

Security

The growing volume and complexity of data carries a high risk of exposure and misuse, complicating regulatory compliance, proper access and deployment practices.

Extensibility

Cloud-based data movement is enhanced by programmatic control and coordination with other cloud-based tools, and can also serve as the foundation for products.

CHAPTER 1

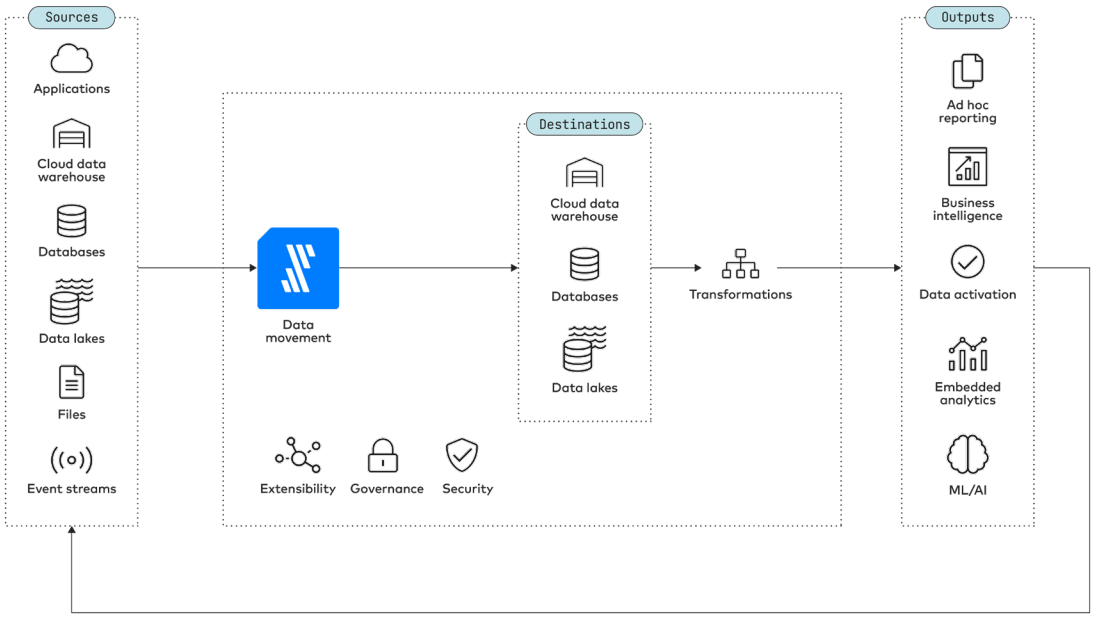
II. Data movement

Data movement refers to moving data from any source to any destination, across any type of deployment – cloud, on-premise or hybrid.

Data movement supports all uses of data, including:

1. Business intelligence and decision support
2. Predictive modeling, machine learning and AI
3. Real-time exposure of data, business process automation and data-driven products

As illustrated by the diagram below, data movement can be multidirectional. The sources can include applications, files, event streams and databases as well as data warehouses and data lakes. Data from sources is either aggregated or replicated in destinations including cloud data warehouses, databases and data lakes.



Most commonly, data is moved from applications, databases and file systems into data warehouses in order to support analytics. Centralizing data into a single destination is necessary because analytics should not be performed on source systems, which are heavily loaded with operational tasks. Moreover, many analytics tasks require data from a variety of different sources to be consolidated. For instance, to analyze marketing data, you may need to combine data from a variety of advertising applications and operational systems in order to construct customer journeys and compare the efficacy of different marketing initiatives.

You may also move data from one database to another, typically for reasons of redundancy or performance. Your production architecture might involve replicating a production database to a failover instance for disaster recovery and to prevent stoppages to operations. Or, you might handle traffic across multiple server regions, and rather than route all traffic to a server on one continent you might replicate to servers across the world so that your users can access servers closer to them with less latency.

Moving data from source to destination requires a high-performance system that can be deceptively complex to engineer. Considerations include properly scoping and scaling an environment, ensuring availability, recovering from failures and rebuilding the system in response to changing data sources and business needs. Many common data integration tools provide frameworks for approaching these tasks but still require a considerable degree of configuration and engineering work from end users.

These complications impose significant costs in time, labor and money on organizations that attempt to move data using an in-house solution. **An automated data movement solution obviates the need for an organization to build such a solution in-house.**

Automated data movement

At Fivetran, we believe that data movement should be fully managed and automated from the standpoint of the end user. From the perspective of the end user, the ideal data movement workflow should consist of little more than:

Step 1:
Selecting connectors for data sources from a menu

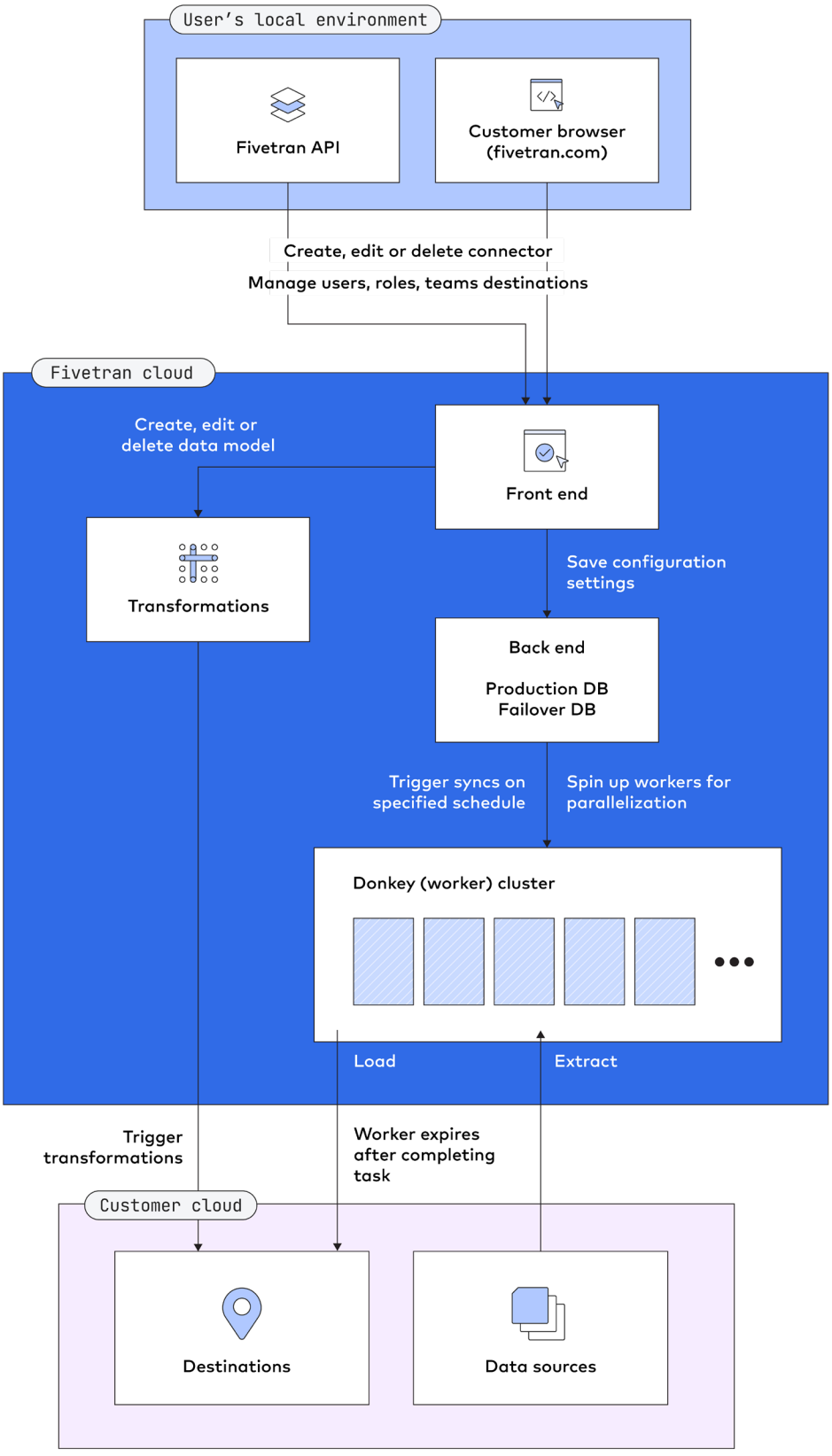
Step 3:
Specifying a schedule

Step 2:
Supplying credentials

Step 4:
Pressing a button to begin execution

The simplicity of this workflow belies considerable complexity under the hood. The Fivetran architecture is strictly divided between a user's local environment, Fivetran cloud and customer cloud. This division is essential for ensuring both security and performance. In terms of security, the strict separations between the front end, back end and customer cloud ensure there are no ways sensitive data can be exposed through the front end. For the sake of performance, as a cloud-native tool, Fivetran makes extensive use of on-demand parallelization.

The following architectural diagram lays out the Fivetran approach to automated data movement:



Note that this diagram illustrates the standard cloud-based solution, which is a low-maintenance, fully managed experience appropriate for most cases. For organizations with security requirements that limit the ability to use cloud-based SaaS solutions, Fivetran also offers hybrid and on-premises architectures.

A typical workflow follows these steps:

1. The user accesses the Fivetran front end through the Fivetran.com dashboard or API.
2. The user creates and configures connectors.
3. The user's choices are recorded in the Fivetran production database.
4. Based on the settings saved in the production database, the Fivetran backend spawns a number of workers on a schedule.
5. Each worker extracts and loads data, with some light processing.
6. Transformations to produce analyst-ready data models are separately triggered and run on the destination.

The smooth operation of this workflow involves a number of design considerations that aren't easily captured in an architectural diagram. In the coming sections we will discuss:

Incremental updates
Ensuring timely updates and minimal disruption to source systems.

Schema drift handling
Accurately representing data even as sources change. Schema drift handling also involves data type detection and coercion, balancing accurate replication and preservation of data with the reliable functioning of data pipelines.

Idempotence
The ability of the data pipeline to easily recover from failed syncs. Idempotence depends on deduplication, which we will also discuss.

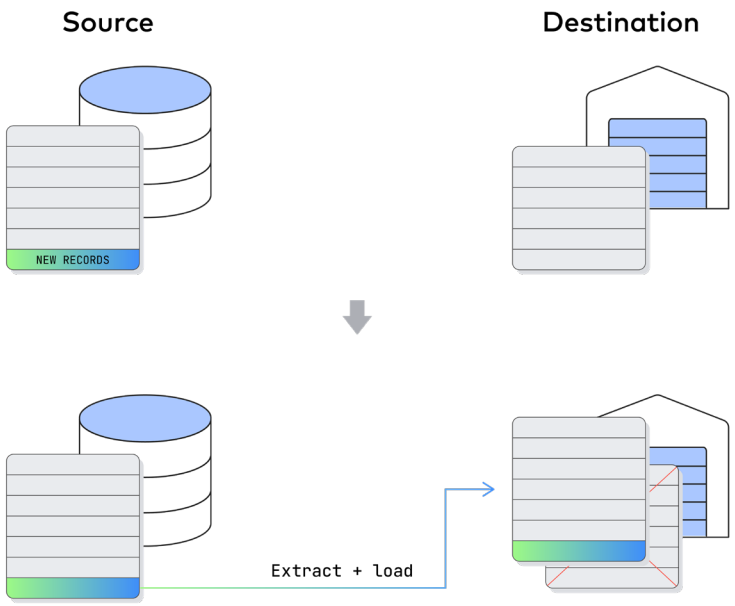
Pipeline and network performance
Minimizing latency and performance bottlenecks.

We will also explore how Fivetran handles **transformation.**

Incremental updates

Whenever possible, Fivetran incrementally updates records from every data source. That is, instead of extracting and loading the entire data source, Fivetran detects new or modified records and reproduces the changes at the destination. Incremental updates are a key feature of an efficient data pipeline.

Full syncs are necessary to capture the full data set during an initial sync, and occasionally to fix corrupted records or other data integrity issues, i.e. resyncing tables or columns.

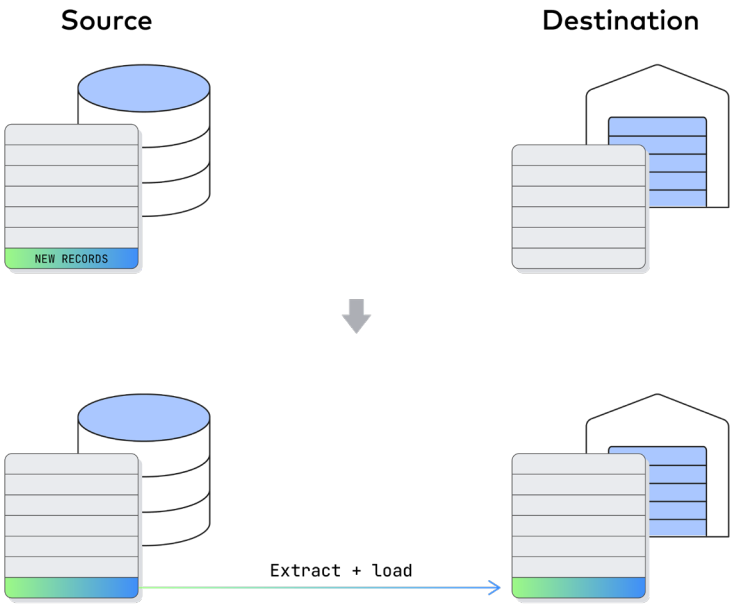


However, full syncs are inefficient for routine updates because they:

1. Often take a long time, especially for large data sources. This makes full syncs incapable of supporting timely, real-time updates. This point is exemplified by the common practice of daily snapshots, where "daily" itself describes the slow rate of syncs.
2. Can bog down both the data source and the destination, consuming resources otherwise required for operations and analytics, respectively.
3. Consume excessive network bandwidth and compute expense.

As data sources grow and add records, the problems listed above naturally grow, as well.

Incremental updates require the ability to identify changes made to a previous state of the source. This practice is often referred to as **change data capture** (CDC). There are several methods of change data capture, but two of the most prominent methods are the use of logs and last-modified timestamps. At small enough increments or batches, change data capture enables real-time or streaming updates.



INCREMENTAL UPDATES FROM A CHANGE LOG

Log-based change data capture (CDC) is most commonly used to make incremental updates from databases. The initial sync from a relational database is always a full sync using a SELECT * query, as it must return all records and all values. By contrast, subsequent syncs should only retrieve new records and values.

Fivetran database connectors parse the transaction log or **change log**. Change logs contain a full history of updates, new records and deleted records as a list of updates.

These updates may be whole-row, in which a record's unique primary key and all values are tracked, offering a full snapshot of each record. They may also be partial-row, in which only a primary key and changed values are tracked.

The sync workflow looks like so:

1. Import a full table using a SELECT * query. This initial sync provides a snapshot of the table's entire contents, primary keys and all, at a particular moment in time. Make note of the primary key column; this will later be essential for designating the correct updates and preventing duplicate or conflicting records.
Suppose we imported a table (right) at 2022-07-30 13:23:04:

id	username	status
1	"D_artagnan"	"inactive"
34	"Goeman"	"active"
45	"Ichabod_Crane"	"active"
94	"jon_rico"	"inactive"
101	"Drosselmeyer"	"active"

2. After the specified interval has elapsed, query the change log and identify all the changes that occurred since your import. Our update corresponds with all transactions before transaction_id = 3, so we'll start from there.

transaction_id	transaction	timestamp_completed	table	row_id	column	value
1	INSERT	2022-07-30 13:23:02	user	45	status	"active"
2	UPDATE	2022-07-30 13:23:03	user	45	username	"Ichabod_Crane"
3	UPDATE	2022-07-30 17:45:30	user	1	status	"active"
4	DELETE	2022-07-30 17:59:03	user	34	NULL	NULL
5	DELETE	2022-07-30 18:12:59	user	94	NULL	NULL
6	INSERT	2022-07-30 18:23:03	user	13	status	"inactive"
7	INSERT	2022-07-30 18:23:03	user	13	username	"capt_ahab"

3. Start reading the change log from the position you identified in the previous step. As you progress through the change log, based on unique row identifiers:
   - Merge updates with the corresponding records in the destination
   - Add new records as new rows with new primary keys
   - Remove deleted records from the destination

id	username	status
1	"D_artagnan"	"active"
13	"capt_anab"	"inactive"
101	"Drosselmeyer"	"active"
45	"Ichabod_Crane"	"inactive"

The incremental data sync workflow depends on idempotence, which we will discuss later. Incremental updates often involve a small amount of backtracking to ensure the complete replication of data. Without idempotence and the ability to properly attribute rows using primary keys, replication can introduce several versions of the same row, depending on how many updates occurred in the interval between the log and the snapshot. This introduces a serious danger of duplicate and conflicting rows.

INCREMENTAL UPDATES FROM A LAST-MODIFIED TIMESTAMP

Data sources that expose data through an API may not have change logs, but may instead have timestamps that indicate when a record was last changed. The goal is to identify, extract and load records that have been added or updated since the last sync.

The process of incrementally updating from a timestamp looks like so:

Step 1:	Step 2:
For each entity, find the "last modified timestamp," "last modified date," or a similar attribute that denotes when a record was last updated.	After the initial update, make subsequent updates based on the timestamp being "greater than" the initial timestamp value, called a cursor. This cursor is updated with every subsequent update.

Timestamps offer a simpler approach to incremental updates than change logs, may be considered more intuitive and are more commonly used. The downsides to timestamps are that:

1. Unlike change logs, which track actions, timestamps may not track deletes, because a deleted record may simply be absent (if it isn't soft-deleted).
2. Since you have to query and scan an entire table or feed, it imposes a heavy load on the source, leading to potential slowdowns.
3. Updating from a timestamp only allows you to access a snapshot of data from any given time. If you ever need to reconstruct every change ever made to a record, you can't use this method without taking and separately storing a succession of snapshots. Even then, there is a possibility that the same row may have been updated multiple times between each snapshot, leading to missed values.

HOW TO SOLVE THE INCREMENTAL UPDATE CHALLENGE

Incremental updates are challenging to properly execute in several regards.

One is of scale. If a data source grows quickly enough, even incremental syncs can become so large and lengthy that the duration of a sync exceeds the specified interval for updating, creating a form of growing data sync debt. Suppose, for instance, that a data connector is scheduled to sync incrementally every five minutes, but the size of the sync is such that it will take seven minutes. Once seven minutes have elapsed, the pipeline is two minutes late starting the next sync. This means that two minutes of extra data has accumulated at the source. This adds additional records to the next sync, which will then take even longer than seven minutes. The sync interval will continue to snowball as more and more data accumulates. The solution to this problem is to write more performant code and more efficient algorithms to minimize sync times.

Another challenge concerns the granularity of timestamps. Suppose the "last modified" field in an API feed is only accurate to the second, and your latest timestamp was 5:00:01 PM. You may not know where between 5:00:01 PM and 5:00:02 PM you ended. If you use the "greater than" approach described earlier, your next sync is likely to miss some records that were updated in the interval from 5:00:01 PM to 5:00:02 PM. These missing or outdated records are easily overlooked until they cause serious problems because few records tend to be affected at any particular time. It can be impossible to recover the missing data. Here, as with change logs, you must use "greater than or equal" logic, erring on the side of introducing duplicate records into the workflow.

Sometimes, APIs contain records that are modified at a particular timestamp but aren't available to read until after that timestamp has already elapsed, leading them to be missed. You may have to use your discretion to determine how many records you are willing to resync in order to catch late arrivals.

Finally, sometimes parts or all of a data source will lack change logs or timestamps. In the best-case scenario, you might be able to identify a reasonable proxy for a timestamp, like a transaction or serial number. In the worst-case scenario, you may be forced to schedule full syncs for part or all of the data source.

Incremental updates and duplication lead us to our next topic, idempotence.

Idempotence

Idempotence means that if you execute an operation multiple times, the final result will not change after the initial execution.

One way to mathematically express idempotence is like so:

```
f(f(x)) = f(x)
```

An example of an idempotent function is the absolute value function abs().

```
abs(abs(x)) = abs(x)
```

```
abs(abs(-11)) = abs(-11) = 11
```

A common, everyday example of an idempotent machine is an elevator. Pushing the button for a particular floor will always send you to that floor no matter how many times you press it.

In the context of data movement, idempotence ensures that if you apply the same data to a destination multiple times, you will get the same result. This is a key feature of recovery from pipeline failures. Without idempotence, a failed sync means an engineer must sleuth out which records were and weren't synced and design a custom recovery procedure based on the current state of the data. With idempotence, the data pipeline can simply replay any data that might not have made it to the destination. If a record is already present, the replay has no effect; otherwise, the record is added.

Fivetran offers an idempotent approach to data movement. The Fivetran approach to idempotence leverages:

1. Cursors to track progress through batches of data syncs, including failed syncs
2. Row identifiers (such as primary keys) in order to identify and remove duplicate or conflicting records

HOW DATA SYNCS FAIL AND WHY FAILURES MATTER

Idempotence comes into play whenever data syncs fail. Data syncs can be interrupted or otherwise fail at several stages of the data movement process.

Source

A data source unexpectedly becomes unavailable, interrupting the sync. This can be the result of a network stoppage or a failure at the source itself.

Destination

Failed queries to a data warehouse can interrupt syncs, as can upgrades and migrations. This may happen because of resource constraints, such as busy nodes and scheduled downtime.

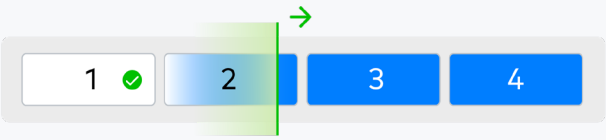
Pipeline

Sometimes the pipeline itself hiccups and stops working. There are many reasons pipelines can fail:

- Infrastructure failures, i.e. servers going down
- Wrong or missing credentials
- Resource limitations, i.e. memory leaks
- Software bugs

A data pipeline will inevitably experience failures over a sufficiently long period of operation. In every such case, data values that are already loaded may be reintroduced to the destination, creating duplicate or conflicting records.

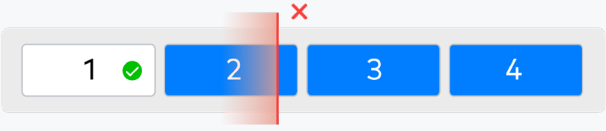
Suppose you have four micro batches of data to be synced to a data warehouse. The cursor, symbolized by a green checkmark, is basically a bookmark that records progress. In the below example, it indicates that the first batch has been completed. The pipeline is partway through the second batch:



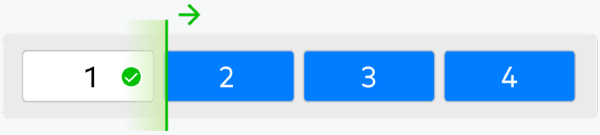
All records from the first batch ("foo," "bar," "baz," "qux," and "corge"), and some from the second batch ("grault," "garply," and "waldo"), have been loaded:

1	foo
	bar
	baz
	quz
	corge
2	grault
	garply
	waldo

Then, the sync is interrupted:



When the data pipeline resumes, it resumes from the last location indicated by the cursor, recognizing that the first batch has been completed but the second has not.



As the second batch is completed, the cursor is updated to the end of the second batch.

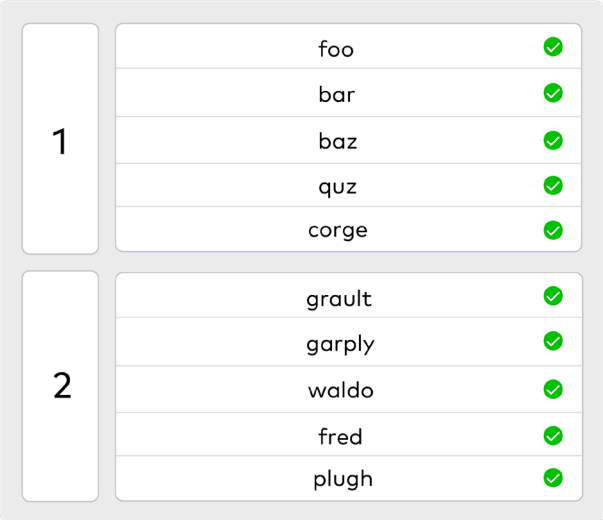
Records that were previously synced from the second batch (in this case, "grault," "garply," and "waldo") are reintroduced to the destination. Without idempotence, the records are duplicated (red rows):

1	foo
	bar
	baz
	quz
	corge
2	grault
	garply
	waldo
	grault
	garply
	waldo
	fred
	plugh

Duplicate records can pose serious problems for business intelligence and analytics:

1. **Misleading or erroneous insights –** you will not have correct rank orders, counts or sums of records. Duplication may also alter medians, averages, distributions and other summary metrics.
2. **Broken operations downstream –** any systems that rely on a one-to-one correspondence between records and identifiers may throw an error or produce the wrong behavior.
3. **Wasted storage space and computational bandwidth –** extra records don't add value yet still have to be accounted for in queries (and disk space).

The challenge is to identify every unique record and ensure there is no duplication so that the output of the process we illustrated above looks like so:



DEDUPLICATING DATABASE RECORDS

Deduplicating records depends on identifying unique entities within records. Databases may explicitly declare primary keys, enabling easy programmatic identification.

As previously discussed, incremental syncs of databases leverage change logs, which contain lists of updates. Without unique row identifiers and idempotence, the destination will create new rows for all log entries, duplicating some rows in the corresponding tables.

When primary keys are not explicitly declared, the data pipeline must use another identifier. One alternative is to impute a primary key by hashing the contents of the entire row; the drawback to this approach is that the data pipeline can only prevent exact duplication. It has no way to prevent new, conflicting rows from being created as individual values in a row change.

Change logs offer an opportunity to detect effective changes so that the data pipeline isn't forced to sequentially reproduce every single operation done to a record. That is, a record that goes through an UPDATE, then UPSERT, then DELETE is ultimately just deleted. Fivetran sorts change log records by DELETEs and primary keys and uses a number of rules to collapse the records into effective changes.

DEDUPLICATING SAAS RECORDS

There is typically no programmatic way to identify primary keys when extracting data through API endpoints for SaaS applications. Instead, they must be found in the documentation or reverse-engineered using knowledge of the source's underlying data model, then constructed from a field or combination of fields that are fixed and unique to a particular entity.

This can be highly error-prone, as seemingly fixed and unique fields, such as email addresses, may actually change. In order to determine a usable primary key, you have to understand the exact behavior of the application that uses the data, specifically which fields can and can't be changed.

For instance, a naive and tempting approach is to identify a user from contact information such as username or email address. But it isn't safe to assume that usernames and emails can't change. Consider the following basic user data:

There are plenty of practical reasons a person might change any of these (moving, marriage, new phone plan, etc.). You must use a fixed reference point for a unique identifier, and therefore determine exactly which fields are fixed within that particular system and which are not. In addition, you need to ensure that any fields you use as primary keys don't (and won't ever) contain null values.

Without documented primary keys, you will need to reverse-engineer the underlying data model and functionality of the application in order to identify the fields that really are both unique and immutable.

If it turns out that there are no unique or immutable fields, then you have no choice but to treat each full record as unique. In this case, the primary key would be imputed as a hash of the entire row. This at least prevents duplication on recovery from pipeline failure, but not from updates to existing records.

Wherever possible, Fivetran identifies primary keys for every table from every source. It can be burdensome to build and maintain a system that makes primary keys of paramount importance, especially as schemas at the source continue to change. It is tempting to simply read records from the source and dump them to destinations, but doing so introduces serious problems with duplication and general data integrity.

The more sustainable approach requires understanding the underlying data model of each source and assigning the correct primary keys. As sources grow in number and complexity, the challenges associated with identifying primary keys grow accordingly.

Schema drift handling

Schemas drift when an application's data model evolves and its columns, tables and data types change. Faithfully preserving the original values of data and ensuring its smooth passage from source to destination, even as the source schema changes, is a particularly vexing data movement challenge. Schema changes are one of the chief failure conditions for traditional ETL, often causing the data pipeline to fail and require an adjustment that requires scarce and highly specialized expertise.

NET-ADDITIVE DATA MOVEMENT AND LIVE UPDATING

One method Fivetran preferred in the past for avoiding pipeline breakages and ensuring faithful reproduction of data was net-additive data movement. The basic behavior is as follows:

- When a column or table is added to the source schema, it is added at the destination as well.
- When a column or table is removed, it is kept in the destination but no longer updated.
- When a column or table is renamed, it is duplicated at the destination under its new name, so that both the old version with the old name and the new version with the new name are present. The old version is subsequently no longer updated but retained for the sake of completeness.

The chief characteristic of net-additive data movement is that columns or tables are never removed despite schema changes. This means that schema changes never lead to lost data. This preservation of old data extends to individual records, as well. Rows that are deleted at the source are soft-deleted or flagged in the destination so that analyses that depend on old data still function.

Fivetran's currently preferred alternative to net-additive data movement is live updating. Here, the basic behavior is as follows:

- When a column or table is added to the source, it is added to the destination as well.
- When a column or table is renamed, it is renamed at the destination.
- When a column or table is removed, it is removed at the destination.

Live updating dispenses with the retention of old schema elements by perfectly matching the data model in the destination with the data model in the source. This extends to individual records as well. Rows that are deleted at the source are deleted at the destination.

New schemas, tables and columns may include sensitive information. For the sake of security, Fivetran supports the ability to exclude or include new schemas, tables and columns – more on that later!

Depending on whether the data source features a change log, some behaviors may not be possible. For both net-additive data movement and live updating, the ideal behaviors concerning renamed columns and tables are only possible if the data source **1) has a change log** and **2) tracks schema changes.** Otherwise, renamed columns and tables are usually recognized as a removal combined with an addition.

HISTORY MODE

One downside of both net-additive and live updating data movement is that values that change but are not deleted are updated in both the source and destination. Without an additional solution, this means the loss of values that change over time.

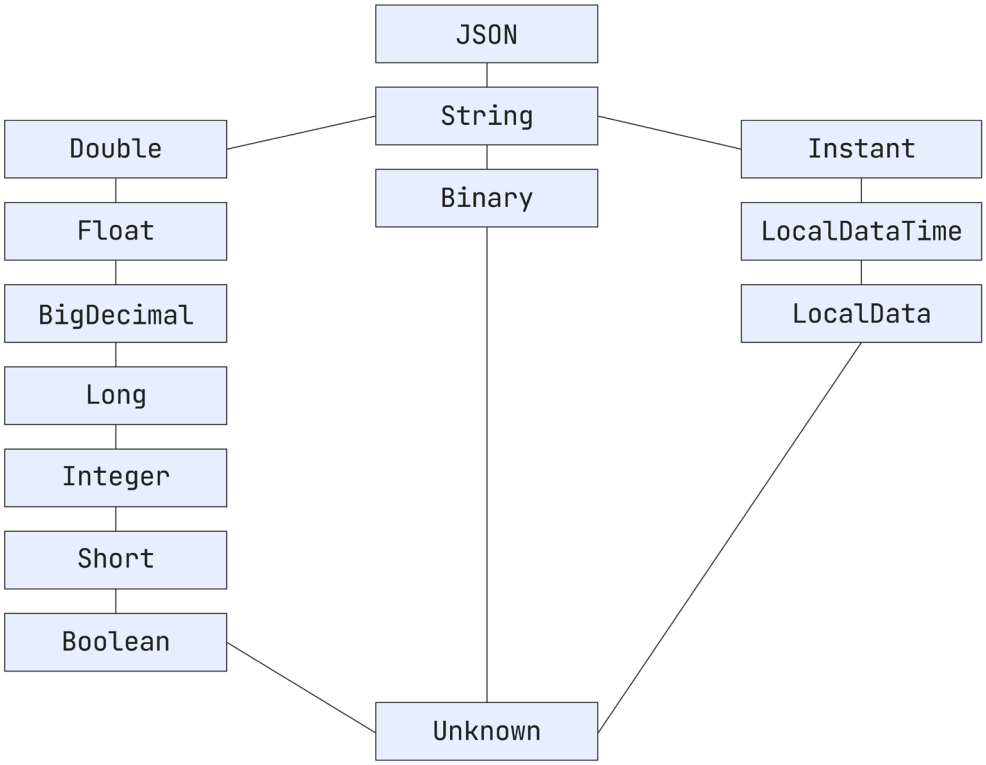
The Fivetran solution to retaining historical values is history mode. History mode retains the current and all previous versions of all rows in a table. A history table specifically contains "start" and "end" timestamps for every version of every value, in which the most recent version has a NULL (or equivalent) "end" timestamp. Deletions are signified when all rows with a particular primary key have a known end time stamp, meaning there is no current "live" version.

Note that keeping track of changes to the schema is a different matter, and schema changes must be represented separately from history tables.

History mode can be costly due to the sheer volume of data that is retained. A good practice is to allow users to selectively apply it to certain tables rather than entire schemas. It can readily coexist with either net-additive data movement or live updates, applied toward the same table.

THE HIERARCHY OF DATA TYPES

Besides a change in name, columns can also change by data type. The key to accommodating a data type change is to assign a new data type that is inclusive enough to handle both old and new values in that column; that is, assigning a supertype.



The chart above, from top to bottom, illustrates a hierarchy from the most inclusive data types (top) to the least (bottom).

Let's say a column consists of integers but fractional values/decimals are added to the column later on. At the destination, we will update the column's data type to "double" instead of "integer," because "double" can accommodate both types. Conversely, if a column consists of decimals that are converted to integers, its counterpart in the destination will remain a double because it's more inclusive.

Reliable data replication requires keeping a data pipeline running and ensuring faithful replication of source data in the midst of schema drift. It is more than a matter of simply copying and pasting data from one place to another, especially when large volumes of data are concerned and analysts require the ability to examine historical data.

Pipeline and network performance

The workflows associated with extracting, loading and transforming data feature a number of possible performance bottlenecks.

Fivetran uses three basic methods to overcome bottlenecks in a data pipeline:

1. **Algorithmic improvements and code optimization** don't involve major changes to the software's architecture.
2. **Architectural changes** can be used to parallelize simple, independent processes.
3. **Pipelining** involves architectural changes to separate the data movement process into distinct, sequential stages that can each simultaneously handle a workload, like an assembly line.

The full workflow of extracting, loading and transforming data can be a complex, path-dependent process. Architectural changes to software affect both how the code is organized and the hardware requirements of the system. This means parallelization and pipelining are potentially costly ways to improve performance and should be pursued only when algorithmic optimization has been exhausted.

ALGORITHMIC OPTIMIZATION

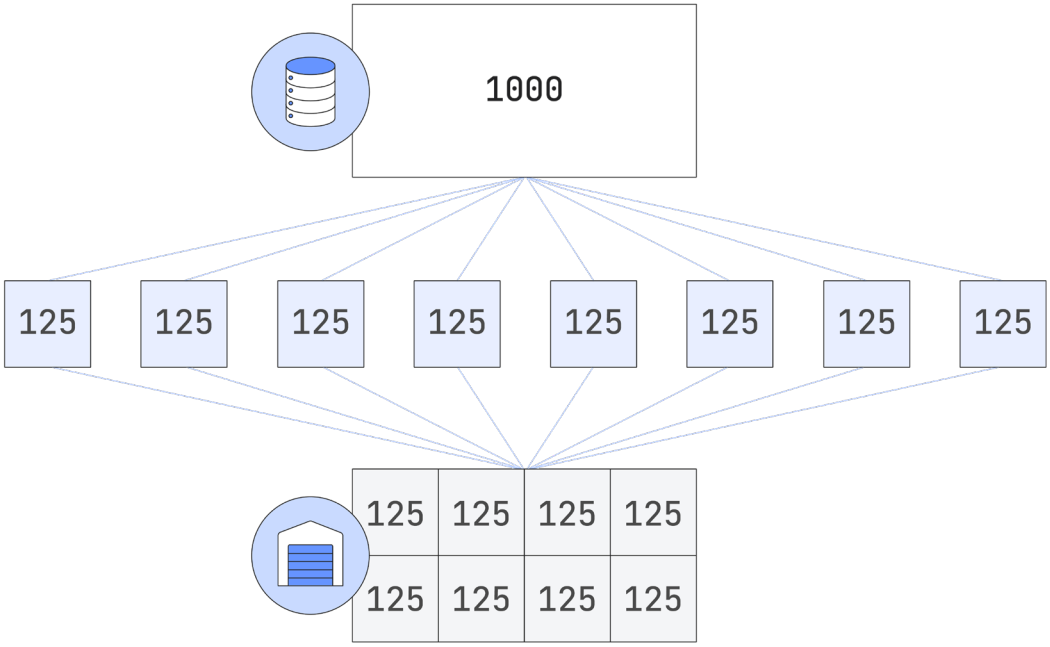
Algorithmic optimization is all about using the most efficient method for every computation. There are better and worse ways to perform every computation. It is also one of the few ways to directly squeeze monetary savings out of improved performance, as parallelization and pipelining require additional infrastructure.

> Suppose you want to test whether a figure is divisible by 2. The hard, slow way is to perform the actual computation, i.e., to divide the figure by 2. A better approach is to see whether the number is even or not by reading the trailing bit of the binary for that number, which should be 0.

In a data pipeline, suppose some data arrives as a JSON and you want to determine if a key has already been converted into a field. Rather than looping through a list of all of the keys, you can create an index out of your fields and then match the key against the index using search or lookup methods, like hash maps.

PARALLELIZATION

Parallelization is about splitting and distributing work into parcels to execute simultaneously rather than sequentially. Suppose you have 1,000 records and spawn eight processes. The original data can be split into eight queues of 125 records each that are executed in parallel rather than 1,000 in sequence.



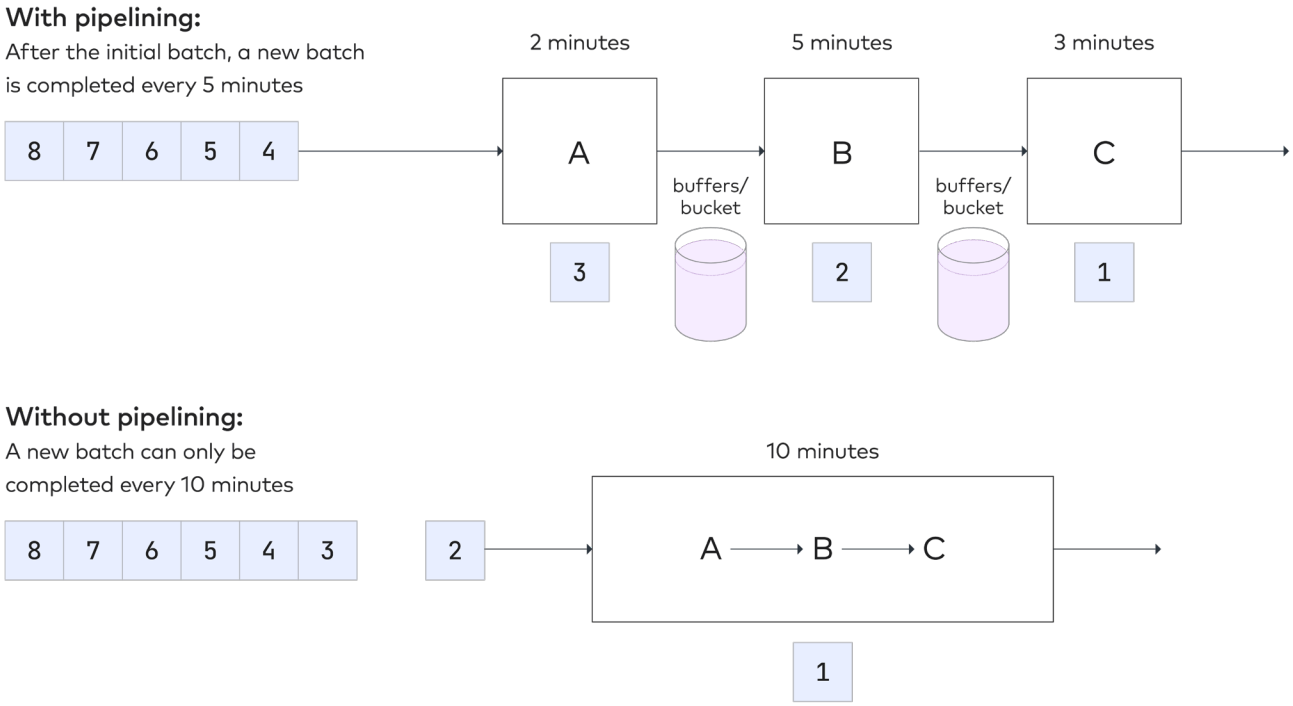
A concrete example is making API queries in parallel. Network request-response times impose a more-or-less fixed interval on each query. To prevent request-response times from adding up sequentially, it's better to send a pool of requests across the network at once.

PIPELINING

The full data movement workstream involves a number of intermediate steps, including deduplication and some forms of aggregation. Operations like deduplication affect entire data sets. They must be performed on monolithic blocks of data where the parts can't be separated in order to produce the correct outputs.

This means that, for some units of work, another approach is necessary for optimizing performance. Rather than splitting the work into pieces of parallel execution whose outputs are reassembled later, the key is to carefully separate the process into steps that can all be simultaneously populated, with buffers between them to ensure that files are fully written before they are handed off to the next stage.

Consider a queue with 8 separate batches that must each be processed en bloc through steps A, B and C:



NETWORK PERFORMANCE

Network performance is a subtopic of pipeline performance, concerning the movement of data between sources, nodes and destinations in a pipeline.

For traffic from most sources, particularly commodity SaaS applications, network performance generally matters very little because the overall volumes of data are low and data is quick to ingest regardless of network performance. At Fivetran, we commonly experience the intentional throttling of API bandwidth, especially in response to repeated queries.

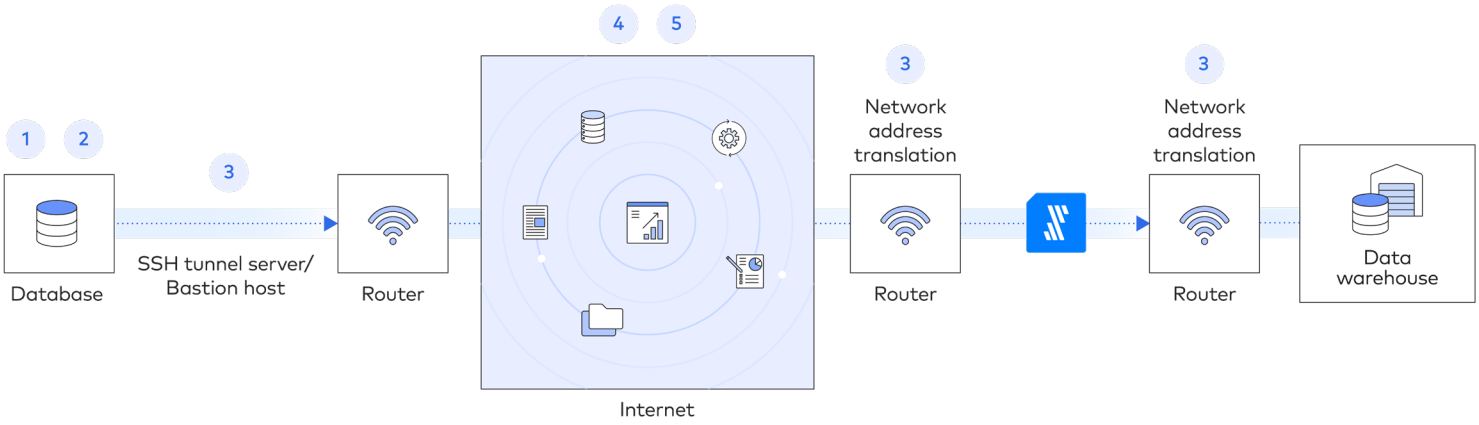
However, network performance is a far more serious matter for high-volume data sources, specifically operational databases. The huge volume of records stored in an operational database heavily impacts the speed and turnaround time of both historical and incremental updates.

Historical syncs, which ingest the full contents of a data source and are necessary both to initialize a data connector and to recover from serious errors, are especially impacted by network performance. Syncs that take many days or weeks can interfere with a company's operations and deter a company from modernizing its data movement infrastructure altogether. With very large datasets, even incremental syncs can bog down.

HOW NETWORK TRAFFIC FROM DATABASES GETS BOTTLENECKED

When the source is a database, there are several common ways syncs can be slowed down:

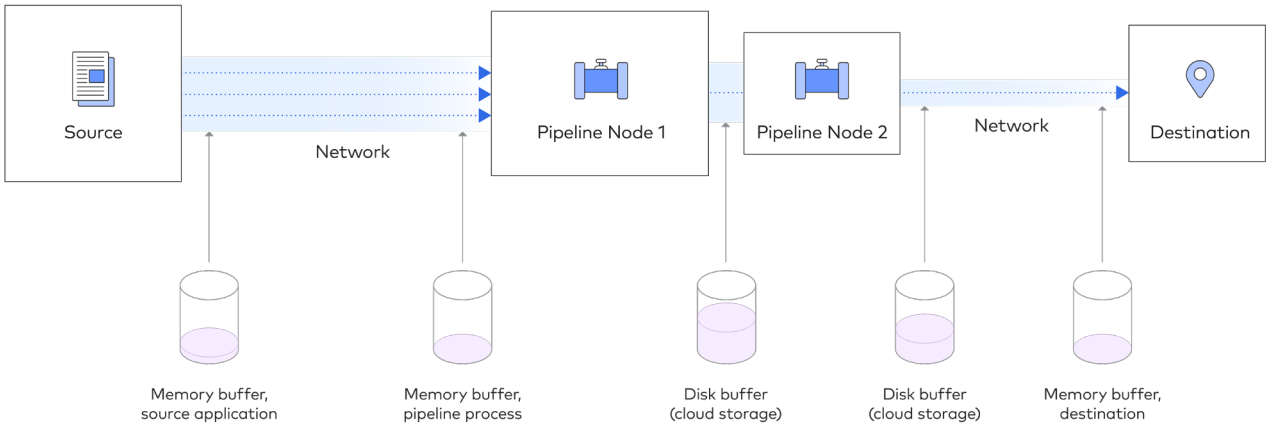
1. Database instances can be too small and lack the resources (i.e. processing power, RAM, disk space, local network bandwidth) to quickly transfer large amounts of data.
2. High transactional traffic can also leave databases too busy to service data extraction.
3. Databases can be gated behind other servers. Security protocols using SSH tunneling, bastion servers, port-forwarders and network address translation (NAT) often intentionally route traffic through narrow checkpoints in order to prevent sensitive data from being intercepted.
4. Inter-region transfer (e.g. from AWS East to AWS West) is slower than intra-region transfer, meaning that you can face slowdowns by moving data from one region to another.
5. Inter-cloud transfer (e.g. from AWS to GCP), or moving data from one network to another, can cause slowdowns.



Note that the example above is just one hypothetical. The relative severity of each chokepoint can vary, and it will take careful troubleshooting to identify which one impacts your data movement workflow the most.

ADDITIONAL CONSIDERATIONS

The measurement of bandwidth and throughput can be complicated by the presence of buffers.



Buffers are staging areas that temporarily contain data, either in memory or on disk, whenever data is set to be handed off between machines. The more bottlenecked a subsequent stage of a handoff in comparison to the previous stage, the more the buffer will fill.

The presence of buffers can radically alter the behavior of a network. Without buffers, the movement of data along every node of the network takes place at a constant rate and is limited by the slowest bottleneck. With buffers, the flow is less brittle as nodes can act as staging areas to store batches of data.

Observing how full buffers get (and how quickly they fill) can offer clues to where your network has bottlenecks.

Another important consideration is compression. Compressing data reduces memory usage and is nearly always worthwhile, especially when supported by the power of modern CPUs. However, compression can complicate how you measure transit through your system, requiring some judgment calls. If a 100 GB file reduces to 40 GB, did 100 GB move through the network or 40 GB?

HOW TO IMPROVE NETWORK PERFORMANCE

Every point of transition from one platform to another has the potential to throttle traffic. The ways Fivetran addresses network performance are similar to the ways Fivetran addresses pipeline performance. The first, and most obvious, is simply to scale up infrastructure, i.e. widen every part of the path. The second is to re-architect the data movement process so that there are fewer points of transition. Yet another way is to optimize algorithms and parallelize execution to speed up processing. Finally, it is important to properly leverage compression and decompression as data moves between platforms.

Transformations

ELT data pipelines extract and load raw data to a destination. Raw data often isn't especially usable to analysts and other users. **Data transformation** consists of all processes that alter raw data into structures called **data models**, collections of tables that directly support analytical and operational uses. These include:

Revising – values sometimes must be corrected or organized in a manner that supports their intended use.

Separating – data values are sometimes combined in the same field because of idiosyncrasies in data collection, but may need to be separated from others for more granular analysis.

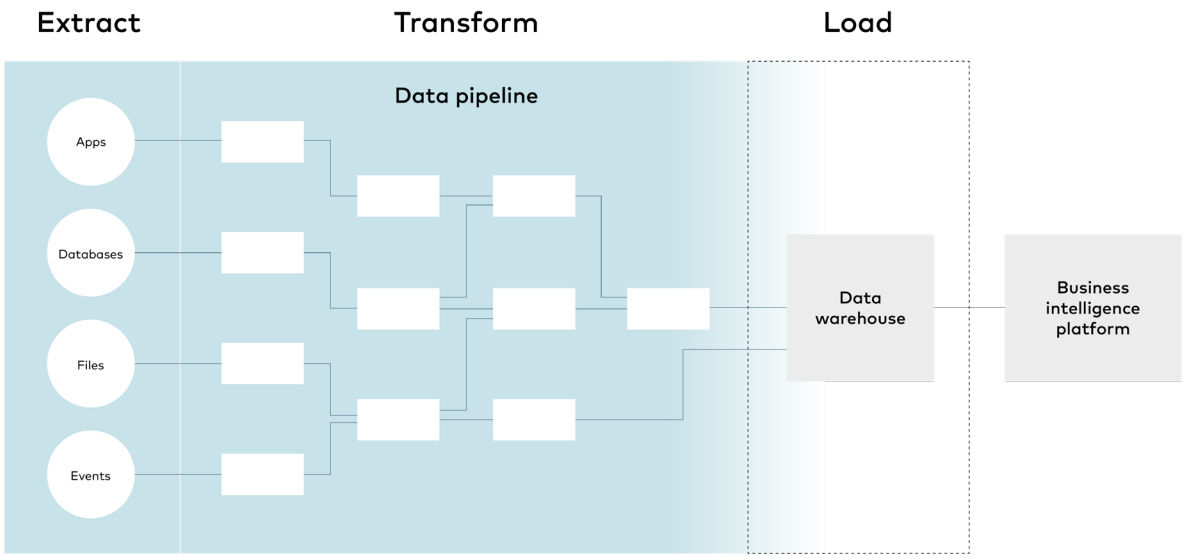
Computing – it is often necessary to turn raw numbers into rates, proportions, summary statistics and other important figures. For machine learning, unstructured data, such as media files, must be vectorized.

Joining – data must sometimes be linked across tables to build a full picture of some phenomenon.

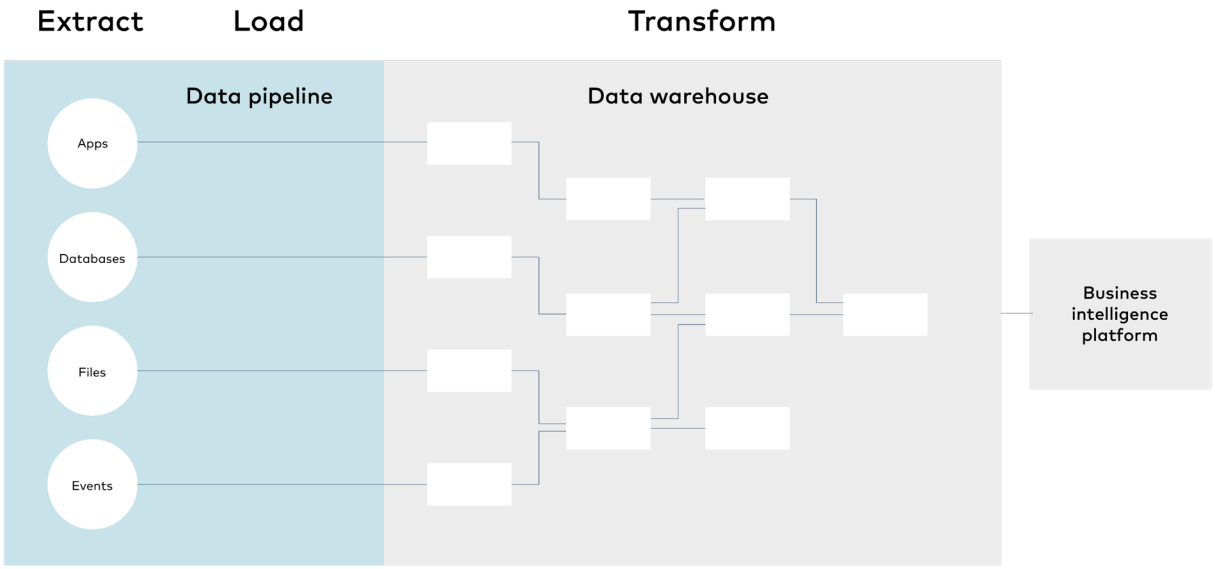
Analysts and other end users depend on the data models produced by transformation to produce visualizations, dashboards and reports. Transformation is also essential to supporting use cases such as embedded analytics, machine learning and data activation.

Traditional ETL features two main pitfalls. The first is that it makes transformation an engineering-centric process. The system requires specialized engineering skills to build and maintain, and imposes a technical barrier between analysts and the functionality they need to perform their role.

The second is that closely coupling transformation with the extraction and loading of data means that the moment upstream data sources or downstream data requirements change, the pipeline will need to be adjusted, resulting in downtime and consuming additional engineering hours.



By contrast, the Fivetran approach to ELT makes transformation accessible to analysts by bringing it into the data warehouse. Transformations inside the warehouse are written using SQL, a more accessible language than Python or Java and the prevailing standard in database operations.



DATA MODELS

Fivetran applies light transformations in the data pipeline to clean and normalize data. This keeps data close to its raw form while eliminating redundancy and enhancing interpretability. With Fivetran data models, we push the data further along to a usable state. This enables analysts to self-serve with less effort and turnaround time.

Fivetran data models are the product of our integration with **dbt™** to power **transformations**. The SQL-based dbt™ data transformation tool enables analysts to build data models iteratively and automate data transformation. Users model their data using SQL SELECT statements, create relationships and dependencies between models and then materialize those models as tables and views in a data warehouse. From there, the analytics-ready tables can be used to produce reports and dashboards.

A dbt project is a directory of SQL and YAML files hosted on a user's GitHub repository. The YAML file contains project configuration information, while each SQL file either contains SELECT statements that transform data in some way or macros that enable code reuse.

With dbt, users can:

- Leverage an extensive open source **library of pre-built data models**
- Practice continuous integration/continuous development (CI/CD), collaboration and version control
- Customize or produce data models by writing, testing and performing SQL-based transformations
- Create and share documentation

The Fivetran approach to data modeling strives to lower the barrier to entry by reducing the **use of the command line** and configuration in several environments. Fivetran Quickstart Data Models allow you to utilize our pre-built data models directly through the platform. This further lowers the technical barrier to entry, bringing yet more modularity to data movement. As with creating new connectors, the process is performed entirely through a graphical user interface.

ORCHESTRATION, INTEGRATED SCHEDULING AND DATA LINEAGE

Data orchestration is the process of coordinating the execution and monitoring of data movement workflows. It is an essential feature of a fully automated data pipeline. In the context of transformations, the main need is to ensure that data is transformed in the correct sequence (and that the process is triggered at the correct time) in order to minimize latency and maintain freshness.

Orchestrations can involve one data source or many. Fivetran automates orchestration for off-the-shelf data models from our library as well as data models custom-built by users. This completely replaces processes that were traditionally conducted using cron jobs, traditional ETL tools like Informatica, scripting and orchestrating with Python and Airflow, etc.

A related consideration to data orchestration is to automatically trigger sequences of transformations based on the status of a connector. This is called integrated scheduling.

In **fully integrated** scheduling, Fivetran automatically runs all transformations downstream of a connector as soon as we finish loading data in your destination.

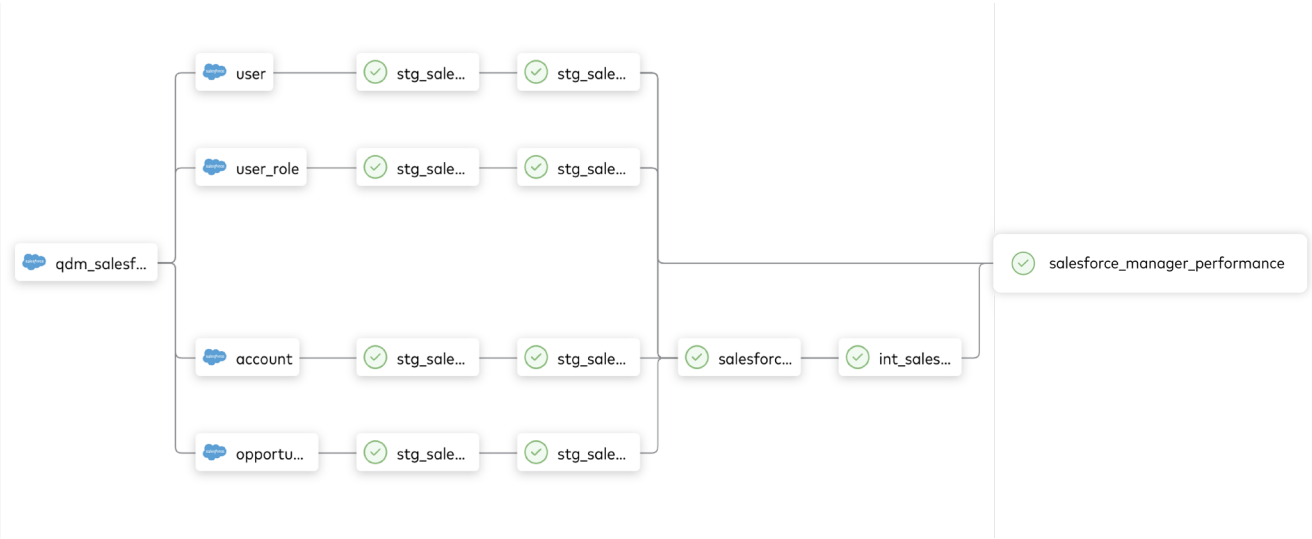
In **partially integrated** scheduling, Fivetran will run transformations according to a schedule you set, except when its schedule overlaps with the associated connector's schedule. In that case, we wait to run the transformation until the connector finishes running.

In **independent** scheduling, Fivetran will run transformations on a fixed schedule determined by the user without consideration for the status of upstream connectors.

Ensuring that models run in the correct sequence and schedule prevents unnecessary syncs, saving compute costs and fostering data integrity by preventing discrepancies between related tables.

It is important to be able to visualize the process of orchestration. Once your data pipeline is set up and automated, data lineage graphs illustrate your data pipeline end-to-end, showing the dependencies between dbt models so that you can track the flow of data through intermediate models. Data lineage graphs also display the run status for each connector and model in the pipeline, enabling you to troubleshoot failed runs.

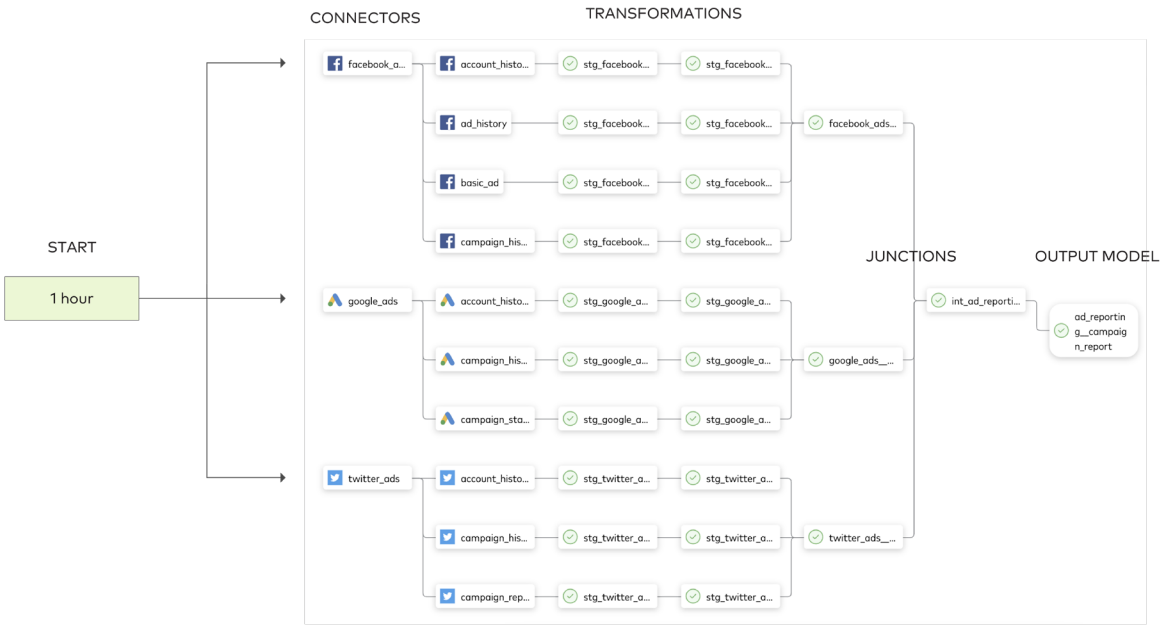
The example below illustrates the data lineage of our Salesforce data model for manager performance:



The graph consists of:

- **Source tables (with the Salesforce logo) –** The starting tables in the sequence
- **Intermediate models (everything else except the output model) –** All models in between the source and output models
- **Output models (at the end, labeled "salesforce_manager_performance") –** The final model analysts use for analytics

A more complex example might involve multiple data sources. Suppose you want to combine data from your various marketing channels to get a view of your return on advertising spend (ROAS). The lineage might look like so:



This graph consists of the following elements:

- The **start** is the interval that initiates the pipeline.
- A **junction** waits for multiple connectors to finish syncing before it triggers a dbt transformation.
- A **connector** updates source tables in the destination.
- An **output** model generates an analytics-ready table. It is typically a leaf node on your data lineage graph.
- A **transformation** is a model or a collection of models that updates downstream tables in the destination.

These visual representations, combined with alerts and notifications, expose dependencies between data models and are essential to tracking the flow of data from your connectors to your destination and helping pinpoint failure points and causes of stoppages.

CHAPTER 2

III. Security

Every enterprise is security conscious for the following reasons:

- Regulatory compliance
- Managing brand risk
- System availability risk
- Basic ethics
- Protecting customers from identity theft and breaches of privacy
- Protecting internal operations, sensitive data and trade secrets

From a technological standpoint, Fivetran addresses security through:

1. Data security, regarding the protection of customer data processed by our services
2. Platform security, regarding tools

Data security

Data security is fundamentally about protecting sensitive data such as personally identifiable information (PII). In the context of data movement, this means applying data security in the pipeline before data is loaded to the customer's destination. Fivetran's primary method of ensuring appropriate access is through column masking. Column masking takes two main forms:

1. **Blocking** data by excluding it from entering the destination altogether. Note that primary keys, due to their importance for idempotence and deduplication, cannot be blocked. Blocked data may still pass through our systems and be stored temporarily but will not be accessible through either the user interface or the destination.
2. **Hashing** data by anonymizing and obscuring it while preserving its analytical value. You will still be able to use hashed columns as keys, joining records across data sets but will not be able to read the original value. Unlike encryption, hashing is one-way and not intended to be reversible. Moreover, Fivetran uses a unique salt for every destination so that general knowledge of the hashing algorithm isn't enough to decode hashed values.

As schemas inevitably change, users can also avail themselves of three sets of schema change handling rules:

Allow all

New schemas, tables and columns will all be synced.

Allow columns

New schemas and tables are excluded but new columns for existing tables are synced.

Block all

New schemas, tables and columns are all excluded by default. The user can choose to be alerted whenever new objects are detected at the source.

Although blocking and hashing can be performed within a destination as well, Fivetran chooses to implement these features as far upstream as possible in order to minimize the risk of exposure.

Platform security

In addition to security at the level of individual data fields, Fivetran also supports many security features built into the architecture of the platform.

A short list of these features includes:

- SAML and SCIM
- Logging and log forwarding
- Access auditing

We will discuss other features in greater detail shortly.

COMPLIANCE CERTIFICATIONS

Fivetran is compliant with a number of common security standards across different industries and jurisdictions:

SOC 1 Type 2

Fivetran undergoes an annual, independent SOC 1 Type 2 audit. This standard allows customers to process data in our platform that is material for financial reporting.

SOC 2 Type 2

Fivetran undergoes an annual, independent SOC 2 Type 2 audit. This standard demonstrates common security, availability and confidentiality controls are in place within our platform.

ISO 27001

These standards require a vendor to:

- Systematically account for information security risks
- Design and implement information security controls and contingencies
- Maintain plans to ensure continued and ongoing compliance

PCI DSS Level 1

This is the most stringent of the Payment Card Industry Data Security Standards, and mandatory for merchants that process at least 6 million credit card/financial data transactions a year, such as retailers.

HIPAA BAA

Although Fivetran is not a healthcare provider or other HIPAA-covered entity, Fivetran complies with the stipulated standards for protected health information (PHI) and will sign a business associate agreement (BAA) with customers who handle healthcare data.

CCPA

Similar to but more expansive than GDPR, CCPA is a California standard that encompasses household as well as personal data.

GDPR

Fivetran compliance with GDPR is mainly enabled by column masking. GDPR is an EU-wide privacy rule positing that end users have the following basic rights regarding personal data:

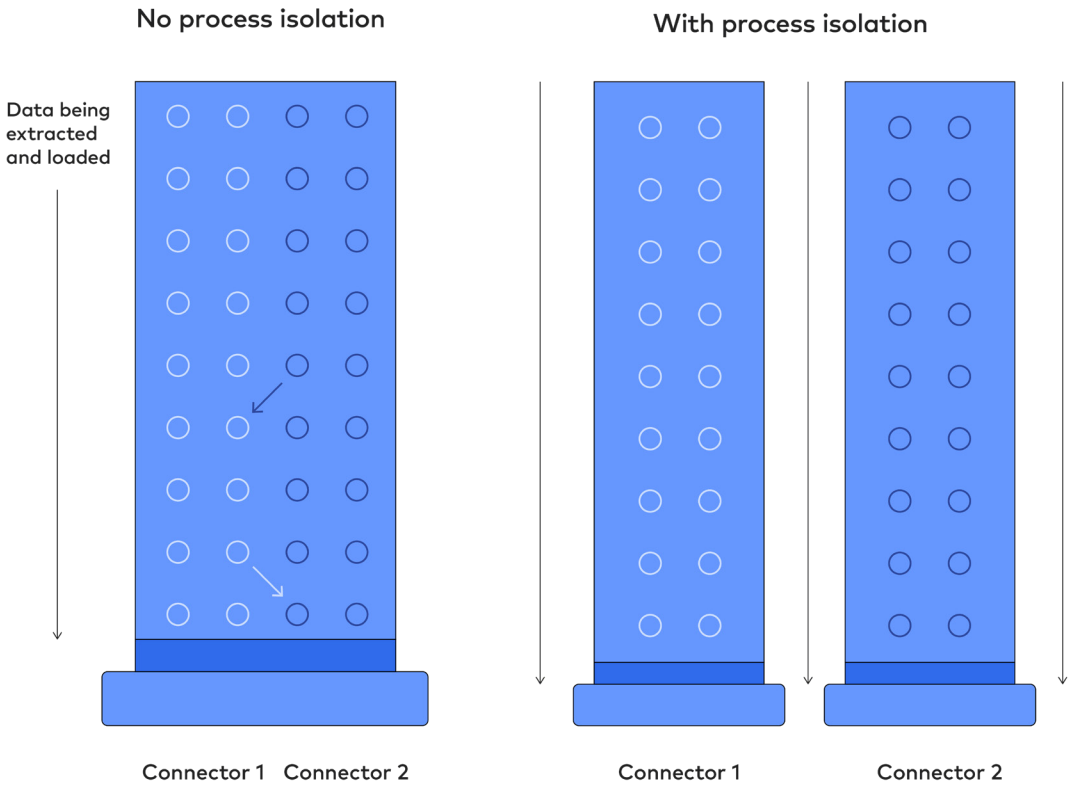
- The right to access
- The right to be informed
- The right to data portability
- The right to be forgotten
- The right to object
- The right to restrict processing
- The right to be notified
- The right to rectification

At least annually, Fivetran engages qualified third parties to perform penetration testing.

SINGLE TENANCY AND PROCESS ISOLATION

Fivetran is architected as an isolated, single-tenant temporary process that expires the moment the sync ends. This prevents persistent storage and cross-contamination of sensitive data.

Single tenancy is related to the concept of process isolation, which we also practice. Single tenancy means that instances of the Fivetran application are never shared between customers. Process isolation means they aren't even shared between different connectors belonging to the same customer, ensuring that there is no accidental cross-contamination of data.



END-TO-END ENCRYPTION

Fivetran encrypts data in transit and credentials using the following methods:

- Credentials are encrypted through a key management service and optionally dual-encrypted using a customer master key, which the customer can withdraw access to at any time.
- All communication between the customer's cloud and the Fivetran cloud is conducted through PrivateLink, VPN or SSH.

- Data in transit is encrypted and decrypted using ephemeral keys, meaning that a unique key is generated for each execution.

The entire architecture depends on the principle of "least privilege" – that is, only the minimum necessary permission is granted for any task. All customer data is purged from Fivetran after it is synced to the destination. The only exceptions are the following, which are required to maintain the continued functioning of connectors:

Customer access keys

Fivetran must access destinations in order to extract and load data to them

Customer metadata

Fivetran stores configuration and other account details in order to display them to customers

Data from email and event stream connectors

Since these sources don't persistently store data, Fivetran does so in case future re-syncs are ever needed

As you may have noticed in the data movement architectural diagram, there are also layers of strict separation between anything the user directly interacts with and the pipeline itself, meaning that, by design, data is never erroneously exposed through the user interface.

DATA ACCESS

Normally, customers enter credentials to connect their Fivetran instance with a database. We retain these credentials in order to continuously extract data and troubleshoot customer issues. These credentials are securely retained in a key management system. Customers control whether and when Fivetran can access their data.

We can optionally further encrypt credentials using customer master keys, which Fivetran does not retain but must consult to use. At any time, customers may revoke Fivetran access to the customer master key, effectively making it an immediate kill switch.

Besides ensuring that third parties cannot access sensitive data, there is the matter of ensuring appropriate permissions and access within your organization. To this end, Fivetran offers role-based access control (RBAC). This enables fine-grained control over:

- Onboarding
- Access
- Auditing
- Monitoring traffic

You can create roles with specific, granular kinds of authority and access and assign users accordingly. These roles can also be combined with SAML/SSO through tools such as Okta, Azure Single Sign-on, etc. We will discuss this topic further in the data governance section.

DEPLOYMENT METHODS

The Fivetran data pipeline can run on a number of different clouds, including AWS, Azure and GCP, with a choice of over 20 major cloud regions worldwide, across North America, Europe, Asia and the Pacific. Customers may have a number of reasons related to specific industries, jurisdictions and use cases for choosing specific regions for residency. We can support geographically bounded US-only access, as well.

Fivetran also offers private networking through services such as PrivateLink.

For those who have highly sensitive and confidential data, Fivetran can also be privately deployed on-premises for database replication.

Learn more about **Fivetran database replication** options.

CHAPTER 3

IV.

Governance

Data governance fundamentally consists of three needs, each of which grows in difficulty and complexity as an organization grows its headcount and the volume, variety and velocity of its data:

1. **Knowing data**

   Keeping a full inventory of all relevant data assets

2. **Accessing data**

   Ensuring that the appropriate parties within an organization have access to data

3. **Protecting data**

   Minimizing misuse or unwanted exposure of data, including by parties within an organization

Without highly capable data governance tools, many organizations are forced to choose between access and compliance.

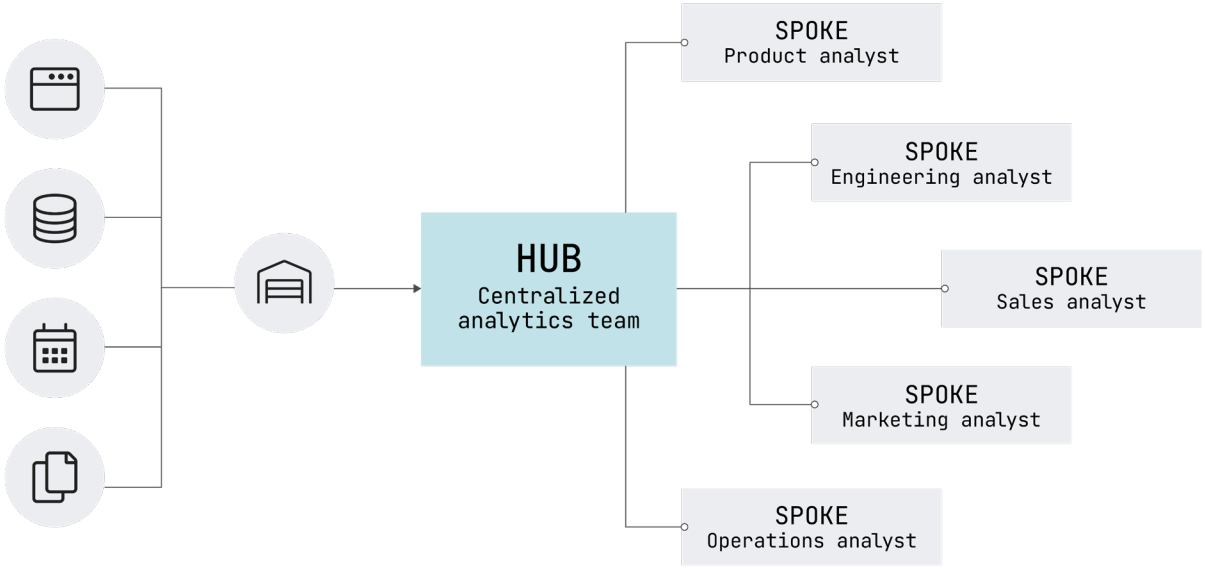
Access vs. compliance tradeoff

At small scales of both data and personnel, organizations have a high degree of trust, familiarity, cohesion, openness and predictability. Many members of the organization are generalists, and different job functions may not be heavily differentiated or siloed. There is little danger posed by granting everyone nearly unlimited access to all data and data products.

By contrast, at larger scales of data and people, emergent complexity arises from the interactions between an organization's members and data assets. Both the tools used by an organization and the job functions performed by its members become more specialized and diverse. Left unchecked, this becomes a recipe for two serious problems.

1. From a security perspective, there is considerably more potential for improper exposure and misuse of data
2. There is more potential for spurious or duplicative data products – models, dashboards, reports and more – creating multiple and conflicting "sources of truth" with unclear provenance

In order to deal with these challenges, many organizations impose limitations on access, creating necessary safeguards to control authorized data access. They may centralize authority and control over access to data, usually in central data teams which are sometimes supported by analysts or data engineers embedded in different business units or departments, i.e. a **hub-and-spoke model**. This model intentionally creates information silos to prevent leaks of sensitive data. The tradeoff is that parties who need the data may face longer project turnaround times and stale data.



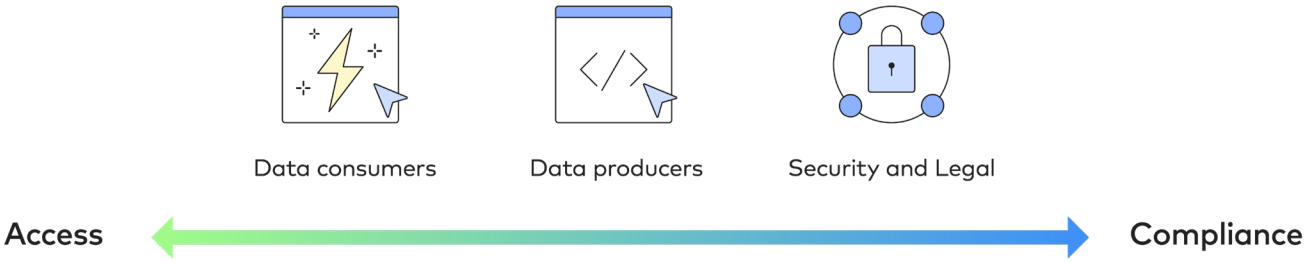
In effect, large companies with complex data needs must contend with three separate interest groups with different incentives and constraints:

Security and legal teams prioritize regulatory compliance, especially in the face of continued regulatory changes. Their main goal is to minimize data misuse. The worst case scenario for security and legal teams is to be the subject of an external audit with inadequate answers.

Data consumers such as analysts want unrestricted access to data to facilitate their projects. They are frustrated by slow turnaround and stale data, and often end up finding ways to circumvent existing data service processes, accessing and producing data products in ways that may not be sanctioned.

Data producers, typically teams of data engineers, are responsible for managing a growing queue of data movement projects. They are forced to balance the competing interests of the security and legal teams, on the one hand, and analysts on the other. For security and legal teams, they must have answers for audits. For analysts, they must be able to field requests for new data assets.

Without the appropriate solution, access and compliance end up as competing, directly opposed values.



Overcoming the access vs. compliance tradeoff

The solution to ensuring both access and compliance fundamentally comes down to ensuring both visibility and control. Visibility and control require automating or programmatically managing all access and compliance concerns at scale. Simple, repeatable and robust processes managed through software can bring an organization to its ideal end state, wherein:

- The concerns of security and legal teams are fully addressed, with full visibility in case of audits.
- Data consumers have full access to any data they require and can self-serve with minimal turnaround times.
- Data producers gain visibility into how data is being loaded, are able to enact appropriate policies for compliance and have the information to properly moderate access.

Let's review the capabilities and features required to knowing, accessing and protecting data.

KNOWING DATA

All three interest groups – security and legal teams, analysts and data teams – have an interest in knowing data. Security and legal teams need to audit incoming data and monitor access. Data teams need to ensure that they are meeting their obligations to analysts and other stakeholders, particularly impact analysis of how upstream data pipeline changes might affect data models downstream. Analysts need to understand the provenance of their data and what questions can and can't be answered.

Fivetran offers several features to bring full visibility into the data pipeline for data audits and use monitoring:

Column-level data lineage exposed through graphs, as previously discussed in the context of transformations

Real-time metadata capture and logging of
- Keys
- Tables
- Columns
- Data types

End-to-end audit trails logging all access, behaviors and changes to the pipeline

A metadata API enabling programmatic management of data movement

Although Fivetran does not feature a data catalog, we can expose all the relevant metadata so it can be imported to your dedicated governance tool.

ACCESSING DATA

Analysts depend entirely on access in order to perform their roles. Data teams are the main gatekeepers to analysts, managing approval workflows for interested parties.

The Fivetran solution consists of automated user provisioning, namely integration with SCIM providers such as Okta and Azure AD to quickly onboard new users. This obviates the need to manually create accounts and configure permissions, which can otherwise be prone to human error and delay.

PROTECTING DATA

Security and legal teams are mainly concerned with protecting data, and there is considerable overlap between data governance and security. Similarly, data teams directly manage access control and handle sensitive information.

To this end, Fivetran features the following, many of which are outlined in "Chapter 2 – Security":

Compliance with laws and regulations across many jurisdictions
- SOC 1 Type 2
- SOC 2 Type 2
- ISO 27001
- PCI DSS Level 1
- HIPAA BAA
- GDPR
- CCPA

Role-based access controls (RBAC) to determine who can make the following actions:
- Move data
- Transform data
- Control where data is loaded

Blocking and hashing

Automated, centralized user provisioning with granular permissions based on roles

Automatic tagging and categorizing of data, including PII

End-to-end encryption

CHAPTER 4

V. Extensibility

We can approach extensibility in two ways:

1

Interoperability is essential for all tools and technologies that manipulate data. As an organization's data needs grow in scale and complexity over time, it will need tools to:

- Perform administrative tasks at scale
- Integrate with other tools and technologies in the data operations ecosystem
- Build a sustainable foundation for future expansion of analytics and data-driven products

2

Data operations are foundational to other tools and technologies, including many that are as-yet undetermined.

At Fivetran, we often liken data to electricity – an enabling technology with unlimited potential for innovative uses.

Programmatic control and automation are central to addressing both understandings of extensibility.

Tools

Fivetran offers programmatic control and automation in the guise of a REST API, Connect Cards and data models.

REST API

The Fivetran REST API enables programmatic control over all aspects of the Fivetran application. Key tools include:

Certificate management

Approve Transport Layer Security (TLS) certificates.

Connector management

Create, edit, remove, run and list connectors and their schema configuration files. This is also the tool for generating Connect Card URIs – more on that later!

dbt transformation management

Create, edit, remove and list transformations within a specified group.

Destination management

Create and edit destinations within the group.

Team management

List, edit and delete your teams, manage team memberships and permissions. Teams can be connected with users, connectors and groups.

Group management

Monitor groups, which are collections of destinations and the users and connectors associated with them. Groups are created first, followed by destinations and connectors.

User management

List, invite, edit and delete users.

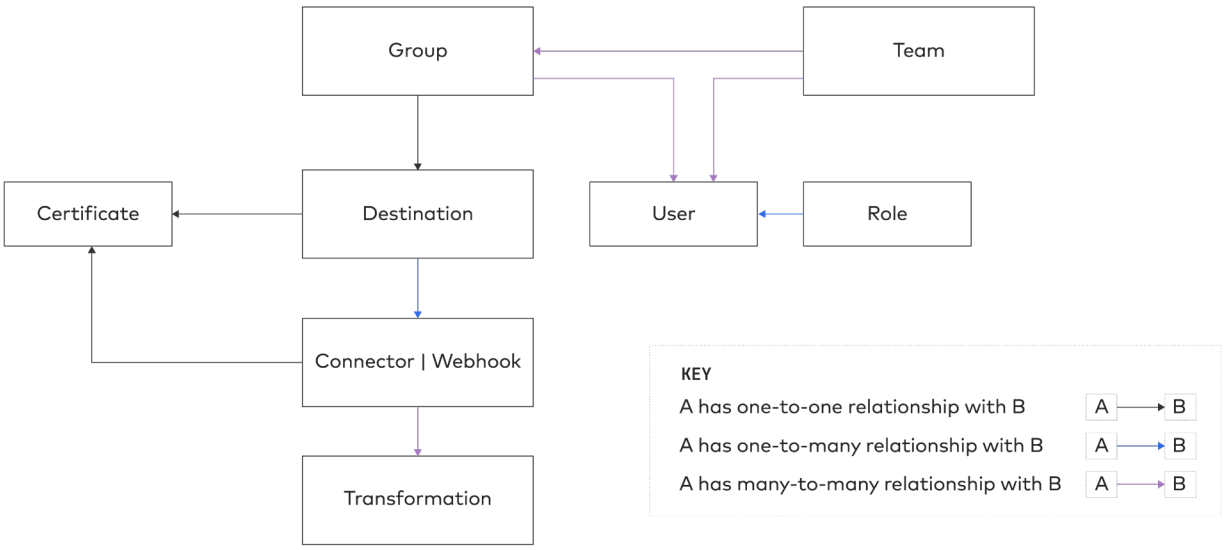
Role management

List all roles, whether pre-defined or custom. Roles are categories of users with specific kinds of permissions and access, which can be highly customized.

Webhook management

List, create, edit, remove and test webhooks, a specific kind of data source that requires a slightly different architecture (updates are pushed rather than pulled).

The relationships between all the elements of the Fivetran application are illustrated below:



CONNECT CARDS AND POWERED BY FIVETRAN

The connector management API is used to enable Connect Cards, which are embeddable pop-up windows that enable outside parties to set up Fivetran connectors. Connect Cards are essential for companies who operate connectors and destinations on behalf of customers.

Connect Cards and the REST API are the central elements of Powered by Fivetran (PBF), an implementation of Fivetran designed for embedding into web applications and analytics portals. The general principle is similar to third-party personal finance applications like Mint or Paypal that require connections to bank or credit card accounts. Connect cards are used for "embedded" onboarding to PBF.

The alternative is to build a "headless" onboarding experience using a custom-built authentication UI. PBF Headless requires more upfront development work but can potentially offer users a more seamless experience, enabling an organization to directly embed Fivetran into a web application.

DATA MODELS

As previously discussed in Chapter 1 – Data movement, Fivetran data models enable rapid bootstrapping of dashboards, reports and other analytical data assets. They enable quick turnaround, use and customization of data models for all uses, up to and including business process automation and predictive modeling.

EXTERNAL LOGGING

Fivetran records and reports a number of behaviors from connectors, data models and accounts. These include errors, failures and delays of all kinds, as well as new users and monthly spend. Although Fivetran features a log connector, Fivetran is not a logging service and only retains log data for a week. Instead, we offer integration with a number of leading logging services, namely AWS CloudWatch, Azure Log Analytics, Datadog, Splunk and Google Cloud Logging.

The simplest use for external logging is to monitor immediate problems. Savvier and more advanced use cases include mining the data for predictive and proactive monitoring, which can then inform automated responses. For the sake of legal and regulatory compliance, logs are also critical for auditing and observability.

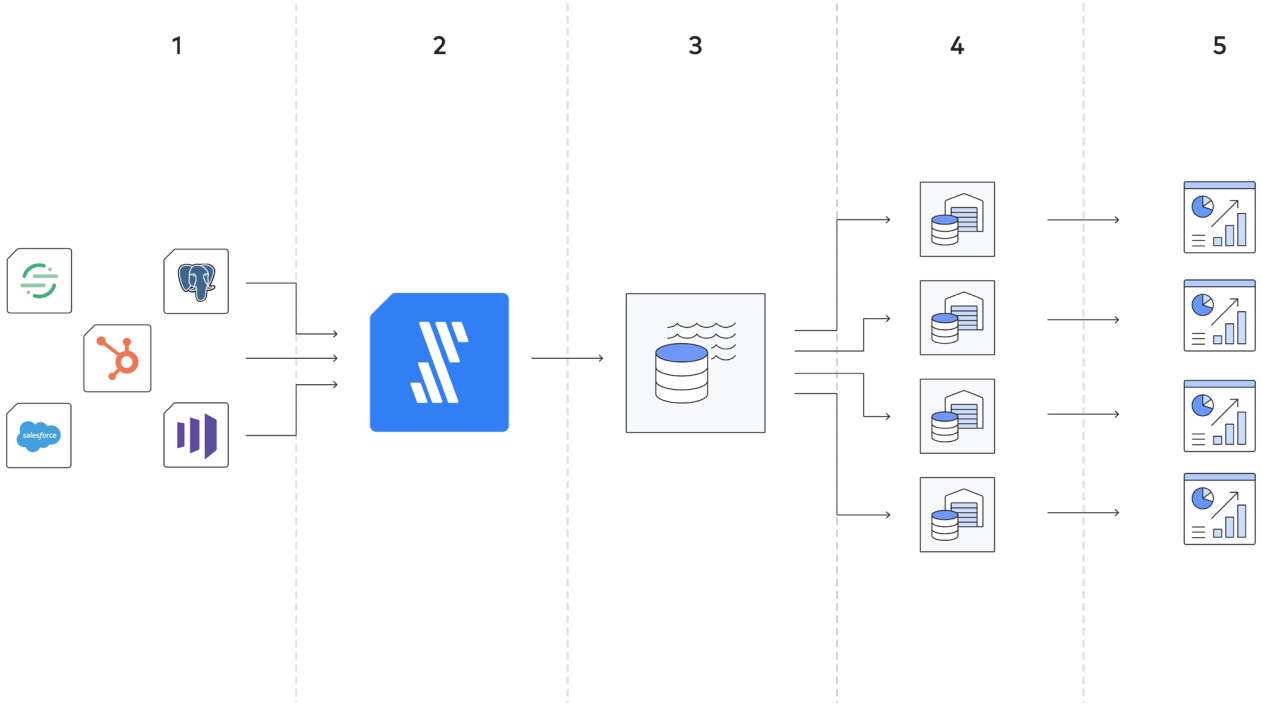
Use cases

Data movement by Fivetran is just one element of a broader ecosystem of tools and technologies for data operations, not to mention business operations and products.

ECOSYSTEM INTEGRATION

The Fivetran REST API can be used to ensure that Fivetran communicates properly with other elements of the data stack. Using **orchestration tools** such as Airflow, Dagster and Prefect, data models that are passed between platforms, such as from a data pipeline (Fivetran) to a transformation tool (dbt) to a data activation tool (Hightouch) can be grouped together into functional units. Developers can use these blocks of code to exercise fine-grained control over scheduling and dependencies. There are many potential reasons a data operations team might be concerned with scheduling. One practical reason is to manage usage-based costs on cloud platforms by minimizing active time/maximizing idle time.

Another important use case for the Fivetran REST API is to help create self-service **data marts**. Programmatic management of data marts can be useful as an organization becomes increasingly complex, or if a company manages multiple destinations on behalf of customers.



PRACTICAL EXAMPLES OF EXTENSIBILITY

Programmatic control over data movement offers a multitude of possibilities for novel products and business models. Some products our customers have built using Fivetran include:

- Business valuation enabled by aggregating data across a huge number of ecommerce businesses
- On-the-fly decision support for ecommerce retailers based on inventory, customer, product and marketing data
- Consolidated business intelligence platform featuring data from every major advertising platform
- Ad recommendation engine for stock photography customers
- Aggregating WiFi usage data to help small businesses optimize service and boost margins
- White glove data integration services, encompassing infrastructure to analytics
- Revenue forecasting built on a model that includes industry-wide data
- Aggregating market-wide performance metrics to help companies evaluate their performance against industry norms
- Personalized learning to help onboard new employees

These products demonstrate the power of data aggregation, as well as the understandable reluctance of many businesses to directly engage in data integration and analytics.

Tools like Fivetran solve basic capabilities for data movement. As more organizations build a firm foundation of data operations and infrastructure, the future will likely see the continued growth of decision support, business process automation, predictive modeling, autonomous agents and more. The most impactful uses for data have yet to be invented or brought to market.

Fivetran

Fivetran automates data movement out of, into and across cloud data platforms. We automate the most time-consuming parts of the ELT process from extracts to schema drift handling to transformations, so data engineers can focus on higher-impact projects with total pipeline peace of mind. With 99.9% uptime and self-healing pipelines, Fivetran enables hundreds of leading brands across the globe, including Autodesk, Conagra Brands, JetBlue, Lionsgate, Morgan Stanley, and Ziff Davis, to accelerate data-driven decisions and drive business growth. Fivetran is headquartered in Oakland, California, with offices around the world.

For more info, visit Fivetran.com.

Start your free trial