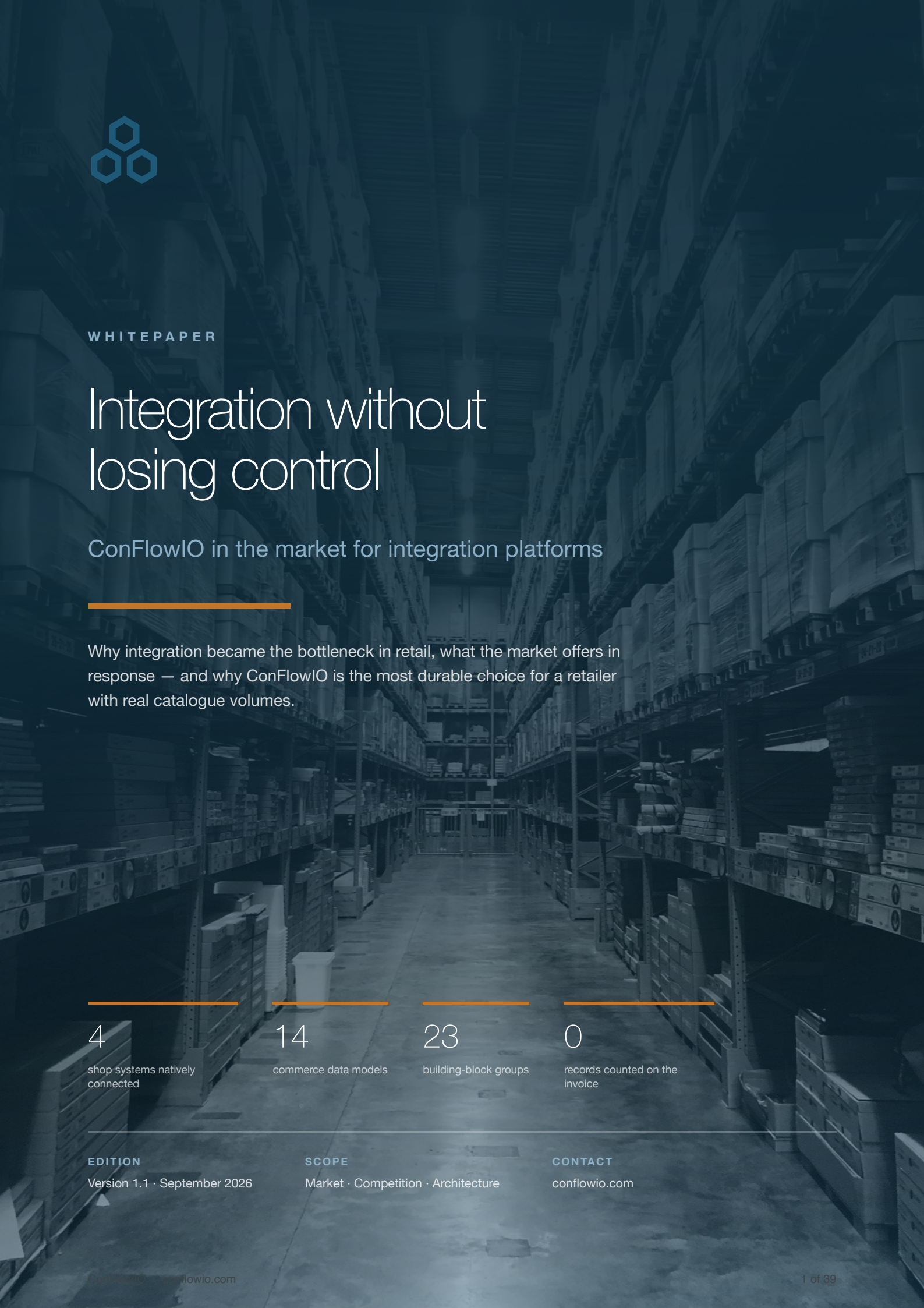


WHITEPAPER

Integration without losing control

ConFlowIO in the market for integration platforms

Why integration became the bottleneck in retail, what the market offers in response — and why ConFlowIO is the most durable choice for a retailer with real catalogue volumes.

4	14	23	0
shop systems natively connected	commerce data models	building-block groups	records counted on the invoice

EDITION
Version 1.1 · September 2026

SCOPE
Market · Competition · Architecture

CONTACT
conflowio.com

Contents

CHAPTER 1

Executive summary

What this paper says in five statements — and what you get out of it.

01

No retailer chose their system landscape; it accumulated. Shop platform, ERP, PIM, marketplace, carrier, accounting — plus supplier catalogues in double digits. The question has long stopped being *whether* these systems talk to each other. It is **who operates the connections, who understands them, and what happens when one of them breaks at three in the morning**.

The market for integration platforms is well populated — over 17 billion US dollars by 2028.[1] But it falls into categories that each carry a structural weakness for retail: enterprise platforms are powerful but consulting-heavy; automation tools are cheap but were never built for catalogue volumes; e-commerce specialists ship good connectors yet bill per operation and run only in their own cloud; EDI houses master data exchange but think in documents rather than product catalogues.

4	14	23	0
shop systems natively connected — Shopware 6 and 5, Shopify, Adobe Commerce, plus marketplaces over TB.One	shared commerce data models instead of one mapping per shop	bundled building-block groups, from files to message queues	records metered: you pay for capacity, not consumption

ConFlowIO addresses exactly these four points — and is the only platform in this comparison that answers all four at once. It is delivered managed, so you spend day one in the editor rather than on a server, and it runs on request inside your own infrastructure. A run interrupted by a restart or a dropped connection is continued rather than started over, skipping what was already done. A **unified data model for commerce data** makes a mapping built once valid for several shop systems. And because nothing is billed per record, the price of a full catalogue synchronisation is predictable.

What you are actually buying

Peace of mind

It runs. And when something goes wrong, it does not cost you an evening.

- Failed steps retry themselves
- A failed run is continued, not started over
- A second attempt creates nothing twice
- A new supplier is a field mapping, not a project

Security

Your credentials and your data stay where they belong.

- Stored encrypted, stripped from every log line
- Vendor and hosting in Germany, in your own house on request
- Workspaces separated fail-closed, roles and an audit log
- Bulk deletion in the shop with a safety limit and a dry run

Insight

You see what is happening — while it happens, not afterwards.

- Live logs per step while the process is working
- Every run individually traceable: which node, how long, what was written
- Processes sit visibly in the editor rather than in a script
- Knowledge about an interface stays inside the company

The five core statements

1. The operating model has to fit the company, not the other way round.

NIS2 is in force, and the location of a data centre does not substitute for the vendor's jurisdiction.[2] ConFlowIO answers both the same way: **the vendor and the hosting are both in Germany.** On top of that, the platform is delivered managed *and* runs on request in your own house — the same platform, the same feature set.

2. E-commerce integration is a data model, not a connector.

Shopware, Shopify and Adobe Commerce hold fundamentally different views of what a product is. Resolve that in a shared model, or build every mapping as many times as you have shops.

3. Durability decides your operating effort, not your technology stack.

A half-written catalogue costs more than one that was never written. Continuing a failed run instead of repeating it, automatic retries and traceable runs save precisely the rework that otherwise falls to people.

4. A per-operation meter penalises the very processes that create value.

If you pay per record, you think twice about an hourly stock synchronisation. ConFlowIO bills for the **capacity** provided: more volume needs more machine here too, but an additional synchronisation within the capacity you already have costs nothing extra.

5. Openness beats connector count.

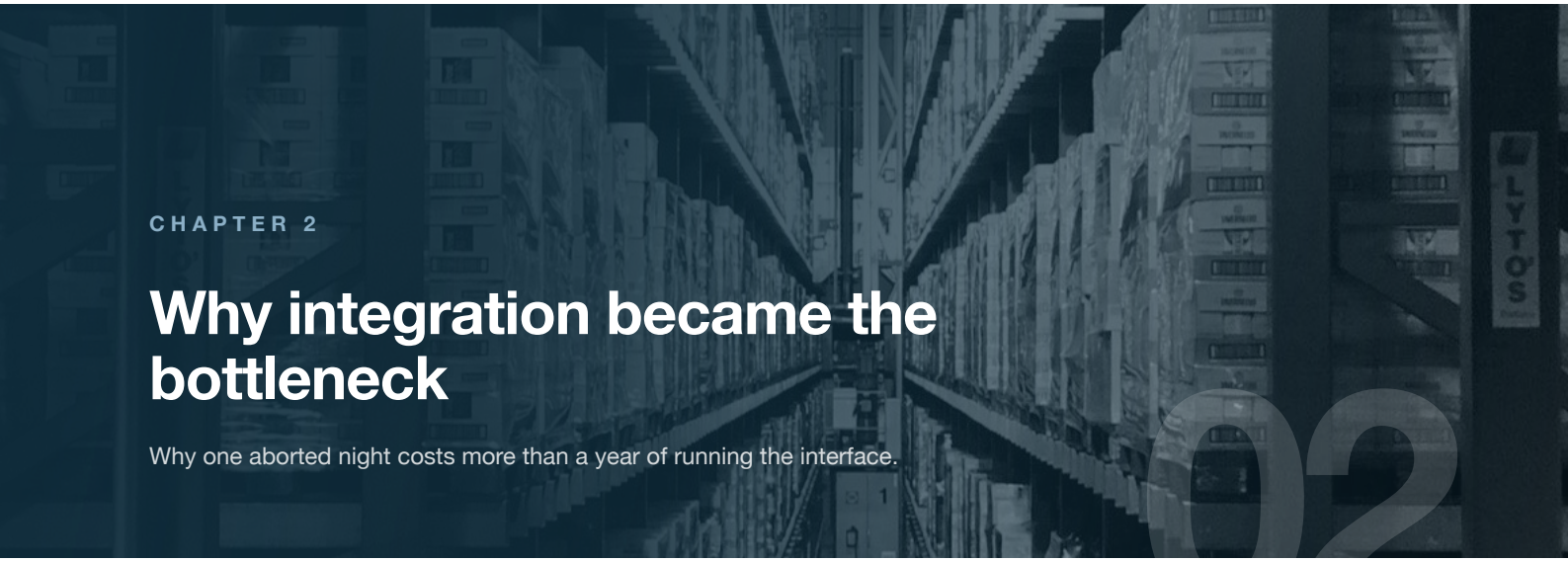
No vendor has the connector for your one odd supplier format. What matters is how quickly it can exist — and whether you are allowed to build it yourself.

THE FINDING IN ONE SENTENCE

Of the five vendor categories in Chapter 3, not one offers the ready-made commerce data model, the durable execution, cost measured by capacity rather than by record, and the choice of operating model together. **ConFlowIO does exactly that** — which is why, for a retailer with serious catalogue volumes, it is the most durable choice available.

WHO THIS PAPER IS FOR

IT leadership, e-commerce owners and digital agencies facing a selection decision. Chapter 3 maps the market, Chapter 5 covers e-commerce integration in detail, Chapter 9 lists twenty-five use cases to check off, and Chapter 10 is a checklist that applies to every vendor — ConFlowIO included.



CHAPTER 2

Why integration became the bottleneck

Why one aborted night costs more than a year of running the interface.

2.1 Connections grow faster than the IT department

Ten years ago: one shop, one inventory system, one interface. Today: a shop, one or two marketplaces, a PIM, an ERP, a shipping system, an accounting package — and suppliers in double digits. The number of possible connections grows quadratically with the number of systems. The number of people maintaining them does not grow at all.

What emerges is familiar to every grown retail IT estate: a script on a server nobody wants to touch; a cron job whose author left the company; an agency retainer that in practice only keeps one interface alive. That works until something changes — and in retail, something always changes.

HOW YOU KNOW THIS PAPER IS ABOUT YOU

- Somebody's first job in the morning is checking whether the night's import completed.
- A new supplier means "we'll manage that next quarter".
- Stock in the shop is from last night, and customer service handles the oversells.
- Nobody can say without looking which articles the last run actually wrote.
- Changing one pricing rule means carrying it into several imports by hand.
- The credentials for the shop and the ERP sit in a config file on a server.
- A second sales channel fails not on willingness but on the mapping having to be built again.

2.2 Four forces reshaping the market

Legacy platforms reaching end of life

Classic integration middleware is reaching the end of its lifecycle. Expiring support is forcing a whole generation of installations into a fresh selection — in a market whose pricing models have changed fundamentally since.[1]

Data sovereignty with legal consequences

In 2026 European regulation crossed a threshold at which the question of where data lives is no longer answered by IT alone. One distinction is routinely lost in tender documents: **the location of the data centre is not the same as the vendor's jurisdiction**. A vendor headquartered outside the EU can be compelled to disclose data even when its servers sit in Frankfurt.[2] For a company pushing supplier prices, margins and customer data through an integration platform, that is not an academic point — and it is the point at which vendors from third countries run out of answers. ConFlowIO answers both halves of the test: **the vendor is established in Germany and the platform is hosted in Germany**, so jurisdiction and data centre sit inside the same European framework. Where data must additionally stay in-house, the same platform runs on request in your own data centre.

Volume and cadence

A nightly stock synchronisation was the standard for a long time. It no longer is: marketplaces punish oversells, and a catalogue of 200,000 articles and variants is mid-market today, not enterprise. The requirement shifts from "it completes" to "it completes every hour without slowing the shop down".

Automation driven by AI agents

A schedule is no longer the only thing starting a process; increasingly, language models do too.[1] That raises the bar for traceability and access control rather than lowering it: every run has to stay explainable.

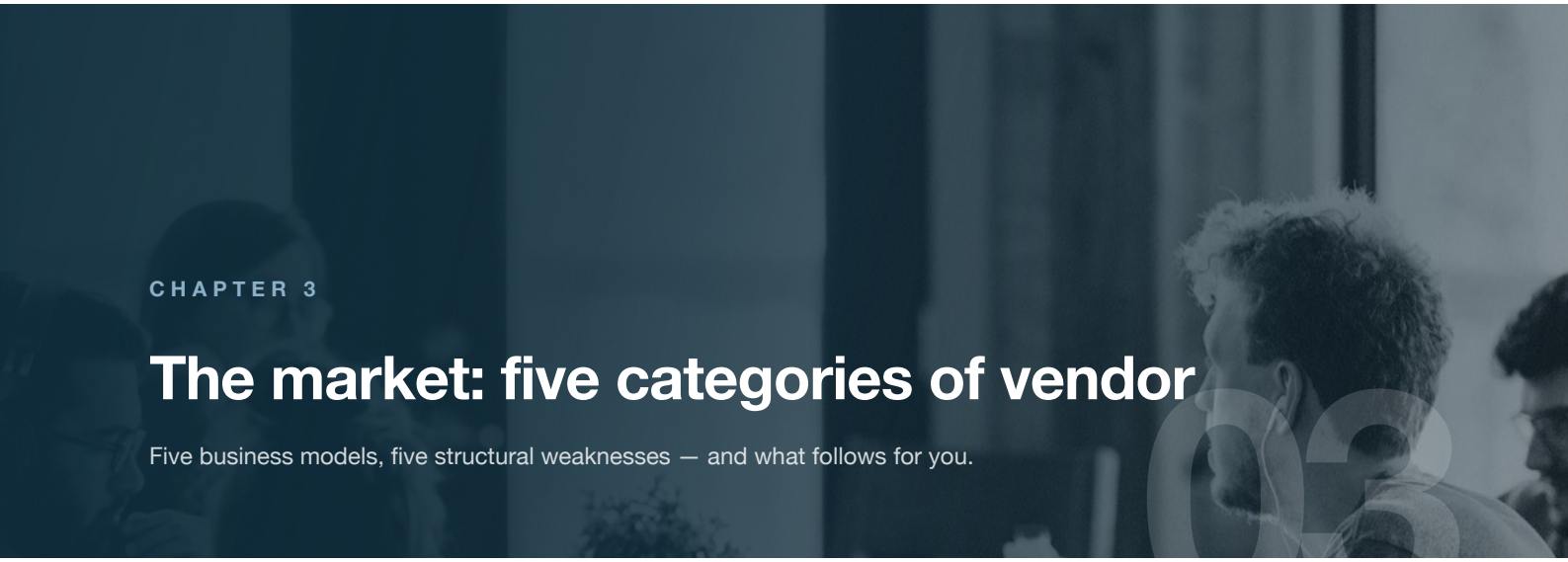
— 2.3 What an aborted run actually costs

The usual failure in a retail integration is not the total outage — that gets noticed. The expensive one is the **partial success**: 40,000 of 120,000 articles updated, then the connection drops. The catalogue is now inconsistent but not visibly broken. Prices are right for part of the range, stock levels for another.

The rework takes three steps, all of which need people: work out where the run stopped; decide whether a second one can safely run; and correct the damage the first one did. In total, an evening like that regularly costs more than a year of running the interface.

> A half-written catalogue costs more than one that was never written — and nobody notices until a customer orders something that is no longer there.

> **THE REAL REQUIREMENT**
>
> Not "the interface must never break" — that cannot be guaranteed, because the far end is not yours. Rather: **a break must not create rework.** The run retries failed steps by itself, can be continued rather than started over, and records traceably what was in fact written.



CHAPTER 3

The market: five categories of vendor

Five business models, five structural weaknesses — and what follows for you.

On the surface every vendor promises the same thing: "connect systems without programming". In fact they differ so much in origin, pricing logic and operating model that a feature comparison misleads. The categories below are therefore drawn along *business model* — because that decides what a solution costs in three years.

The middle column is not a criticism of the vendor but a description of what follows from its business model once catalogue volumes are involved.

— 3.1 The five categories

Category	The weakness in a retail context	What ConFlowIO does differently
Enterprise iPaaS Boomi · MuleSoft · Workato · SnapLogic · Frends — out of corporate IT, integration as a governance question	The entry point. Comparisons place Workato in the mid-market at roughly 200 to 1,500 US dollars per month[3], and an implementation project with external consultancy is normally on top. For a retailer who needs eight interfaces and has no integration department, the overhead exceeds the problem.	Tenant separation, roles and an audit log from day one — without the project this category makes them conditional on.
Automation tools Zapier · Make · n8n · Pipedream · Activepieces — out of office automation, from around 9 to 20 US dollars a month[3]	Volume. They are built around events — a new lead, a new order — not around a catalogue of 200,000 rows; operating models tip economically between 10,000 and one million operations per month[3]. n8n, the only one seriously self-hosted, is limited to "internal business purposes"[4] — a material boundary for agencies and system integrators.	Catalogue data runs page by page, change detection reduces a full catalogue to what actually changed — and there is no usage restriction for agencies.
E-commerce specialists Celigo · Patchworks[5] · Alumio[6] · Synesty[7] — specialised in precisely this use case	Operating model and price. All four run exclusively in the vendor's cloud. Patchworks bills by connector count and operation volume, Alumio by "integration capacity"; no prices are published, and every figure requires a sales conversation[8].	Comparable connector depth, but with no obligation to a vendor cloud and no per-operation meter. Neither connectors nor processes nor users are the price lever.

Category	The weakness in a retail context	What ConFlowIO does differently
EDI houses Lobster · SEEBURGER · ecosio · Procuros · Comarch — out of electronic document exchange	The mindset. Lobster, for instance, is a serious offer — over 2,000 customers, more than 90 connectors, ISO 27001, deployment on-premises, in the cloud or hybrid[9] — but built around the *document* rather than the *product catalogue*. For variants, properties, category trees and media that means: everything is possible, and everything is configuration work.	The same deployment flexibility, but thought through from the product catalogue. Fourteen ready-made commerce data models replace the configuration work that opens every project.
In-house build the script that runs at night — the most frequent choice and the one least often named in a tender	From the third interface onwards the arithmetic turns. Not because of the development, but because of everything else a platform brings: logging, retries, credential management, scheduling, visibility for non-developers. And the question of who takes over when the author leaves.	Exactly those supporting functions are the product — and the knowledge about an interface stays in the company when a person moves on.

Example vendors per category; the sourced figures carry their reference in Appendix B.

— 3.2 Comparison matrix

Category	Runs on your infrastructure	Pricing logic	Catalogue volumes	Shop data model	Typical entry
Enterprise iPaaS Boomi, MuleSoft, Workato	partly / hybrid	subscription + volume	good	generic	high, with a consulting project
Automation tools Zapier, Make, n8n	n8n only (licence limit)	per operation	weak	generic	very low
E-commerce specialists Celigo, Patchworks, Alumio, Synesty	no	connectors + operations, on request	good	present	medium, sales-led
EDI houses Lobster, SEEBURGER, ecosio	yes	subscription / licence	very good	document-oriented	medium to high
In-house build	yes	staff cost	varies	hand-built	low, then rising
ConFlowIO	**your choice: managed or in-house**	**capacity, not a per-operation meter**	**page-by-page processing**	**included**	**low**

Classification by operating model and pricing logic. Assessment based on publicly available vendor information and market comparisons (sources in Appendix B); snapshot as of September 2026.

Read this matrix by column: every category is strong in at least one and structurally weak in at least one. **ConFlowIO is the only row without a weak column** — and that is not a matter of feature count but a consequence of designing the operating model and the pricing logic for retail from the outset.

— 3.3 The three gaps no category closes

> **Gap 1: self-hosting *and* a commerce data model**
> Wanting to self-host lands you at n8n (with a licence boundary and no catalogue understanding) or an EDI house (thinking in documents). Wanting a ready-made shop data model lands you in a vendor's cloud. Almost nobody offers both.

Gap 2: predictable cost at high volume

Nearly every pricing model in the market scales with the number of records processed. The processes with the highest value — frequent stock synchronisation, daily catalogue imports — are therefore the most expensive.

Gap 3: durability as a product property

Plenty of vendors offer retries. A run that continues *at the same point* after a process restart, rather than from the beginning, is considerably rarer — even though in retail operations it saves the greater part of the rework.

These three gaps are not accidental; they follow from where each category came from. ConFlowIO was designed along exactly these three gaps — the next chapter says what follows from that.

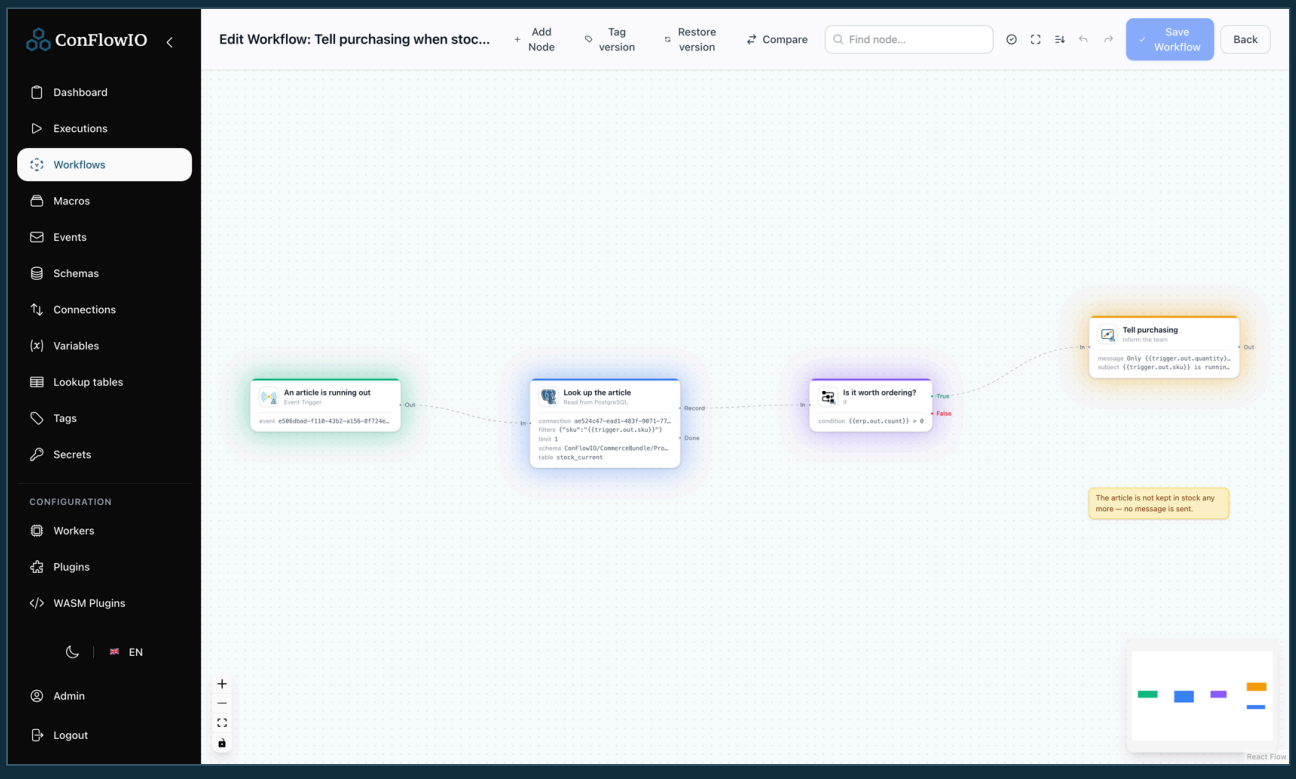
CHAPTER 4

Where ConFlowIO sits

Five principles you can hold us to.

04

ConFlowIO is an integration platform for companies that want their interfaces under control without having to program them. Processes are built in the browser as a graph of **nodes** — each node one step: read data, check it, transform it, write it. Execution is handled by a durable execution engine that records runs, retries failures and resumes interrupted runs.



What a process looks like: an event starts the run, one node reads the article from the ERP database, a branch decides, and the last node notifies purchasing. Every node shows its settings on the card itself — what the process does is in the picture, not in a script.

4.1 Five principles

Principle 1 — You choose the operating model, not the vendor

ConFlowIO is **delivered as a managed platform by default**: installation, updates, backups and monitoring are ours, and you spend day one in the editor rather than on a server. Where internal policy, group requirements or certification demand it, **the same platform runs on request inside your own infrastructure** — not a reduced spin-off, but the same product with the same feature set.

In this competitive field that is the exception. The e-commerce specialists in Chapter 3 offer only their own cloud; the EDI houses offer self-hosting without a commerce data model. Changing the operating model later is not a migration project here.

Principle 2 — A run survives a failure

Every run is a durable operation with its own state. If a worker dies, a server restarts or a connection drops, another worker takes over and continues from the last point reached. And a run that fails for good can be **continued rather than started over**: it skips what finished the first time and takes over the files already downloaded and parsed. User cancellation is cooperative: the step ends cleanly instead of being cut off mid-write.

Principle 3 — Commerce data has a model of its own

ConFlowIO ships a **Commerce Bundle**: fourteen coordinated data models for products, categories, manufacturers, properties, variants, prices, tax rates, sales channels, customer groups, media and downloads. Every shop integration speaks this model. Chapter 5 explains why that is the most important difference from a generic platform.

Principle 4 — You pay for the machine, not for permission to use it

What is billed is the **capacity** provided — the environment your volume needs. If your volume grows substantially, so does the price; that is no different here than anywhere else. The difference is *how* it grows: in a few predictable steps rather than with every individual record. **An additional run within the capacity you already have costs nothing extra.** Whether you synchronise daily or hourly is therefore a technical decision — under a per-operation model it is a cost decision, every single time.

Principle 5 — Extensible without asking the vendor

Building blocks are plugins. Beyond the ones shipped, you can add your own, including modules that run in an isolated environment and are loaded while the system is running. For the most common case — "our supplier sends a format nobody has ever seen" — there is also a code building block directly in the graph, with no plugin required.

4.2 A clear boundary

Two things this paper states plainly, because they matter in a selection decision:

- **Depth rather than a connector catalogue.** The bundled integrations cover the systems that actually appear in retail — at a depth a library of a thousand shallow entries does not reach. Everything else is reachable through files, HTTP, databases and message queues.

- **Not an ERP, a PIM or a shop system.** ConFlowIO replaces none of your systems; it connects them. That is precisely why it can run alongside every one of them without competing.

— 4.3 The head-to-head

Chapter 3 sorted the market; here is the result, one row per category, measured against the five criteria that proved decisive there.

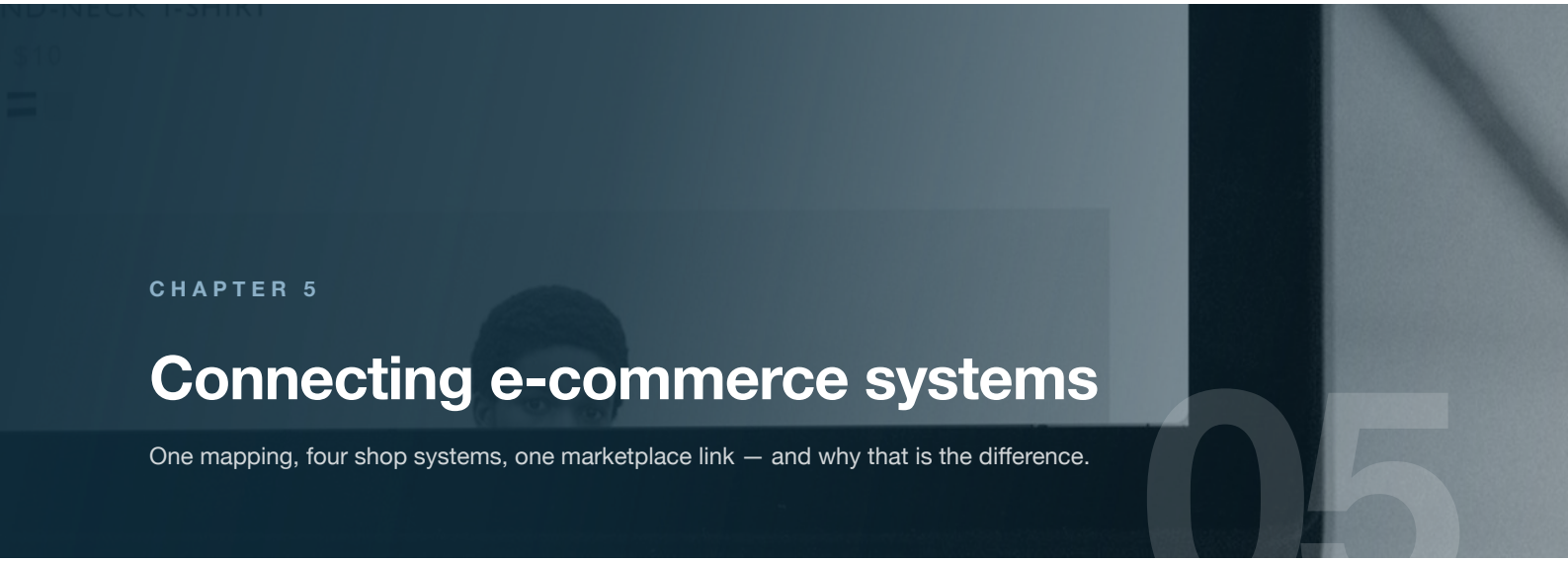
Vendor category	Choice of operating model	Commerce data model	Catalogue volumes	No per-operation meter	Durable runs
Enterprise iPaaS	◐	○	●	○	◐
Automation tools	◐	○	○	○	◐
E-commerce specialists	○	●	●	○	◐
EDI houses	●	○	●	◐	◐
In-house build	●	○	◐	◐	○
ConFlowIO	●	●	●	●	●

Our own assessment against the five criteria that Chapter 3 identified as decisive.

● met ◐ partly ○ not met

WHAT THE DECISION COMES DOWN TO

Every category in this table is strong in the field it came from. For a retailer with real catalogue volumes, several channels and limited IT capacity, however, the open points accumulate exactly where the daily business happens. **ConFlowIO is built for that case — and there it is the best available choice.**



CHAPTER 5

Connecting e-commerce systems

One mapping, four shop systems, one marketplace link — and why that is the difference.

This chapter is the core of the paper. Comparing platforms by connector list misses the actual work: *reaching* a shop system is not the hard part — each has a documented API. The hard part is that every shop system holds a different view of **what a product even is**.

NATIVELY CONNECTED

Shopware 6 · Shopware 5 · Shopify · Adobe Commerce · Tradebyte TB.One

— 5.1 The problem: four systems, four world views

One example that affects every migration and every multi-channel operation — the question of how a system recognises an article again when the same import runs tomorrow:

System	How an article is recognised	Where the price lives	Where stock lives
Shopware 6	a 32-character internal identifier; the article number must be mapped onto it	on the product, as a price list per currency	on the product
Shopware 5	directly by article number	on the article detail	on the article detail
Shopify	no business key — only an internal identifier and the URL handle	exclusively on the variant	per location, through a separate API
Adobe Commerce	directly by SKU	on the product, extra fields in a separate list	in the extension attributes

Identity, price and stock across four shop systems — the three fields every catalogue import hangs on.

Four systems, four fundamentally different answers to the same question. A generic integration platform hands that problem straight back to you: you build one mapping per shop, each with its own logic for identity, prices, variants, categories and media. Run two shops, or migrate from one to another, and you build

everything twice.

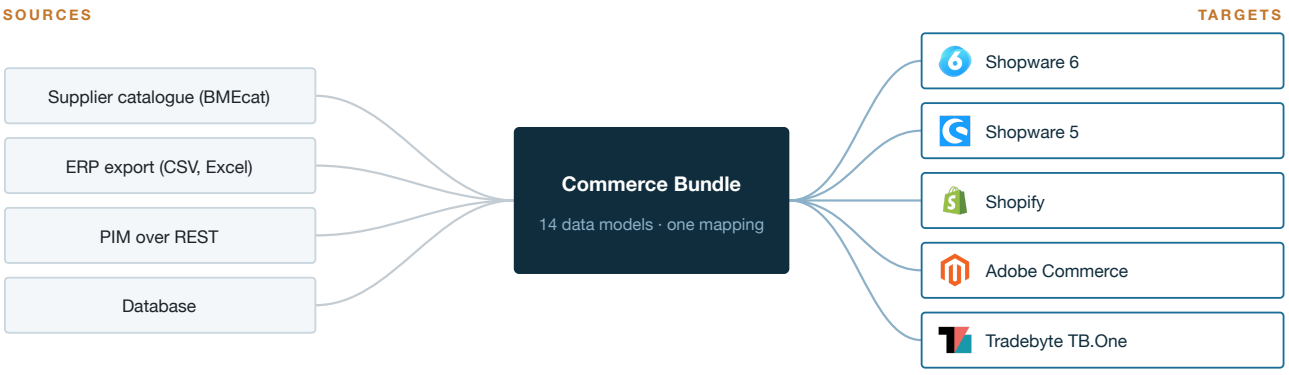
> Reaching a shop system is not the work. The work is that every shop system holds a different view of what a **product** is.

5.2 The Commerce Bundle — one model for every shop

ConFlowIO resolves this with an intermediate layer. The **Commerce Bundle** defines fourteen data models describing what commerce data is, independently of which shop it eventually lands in:

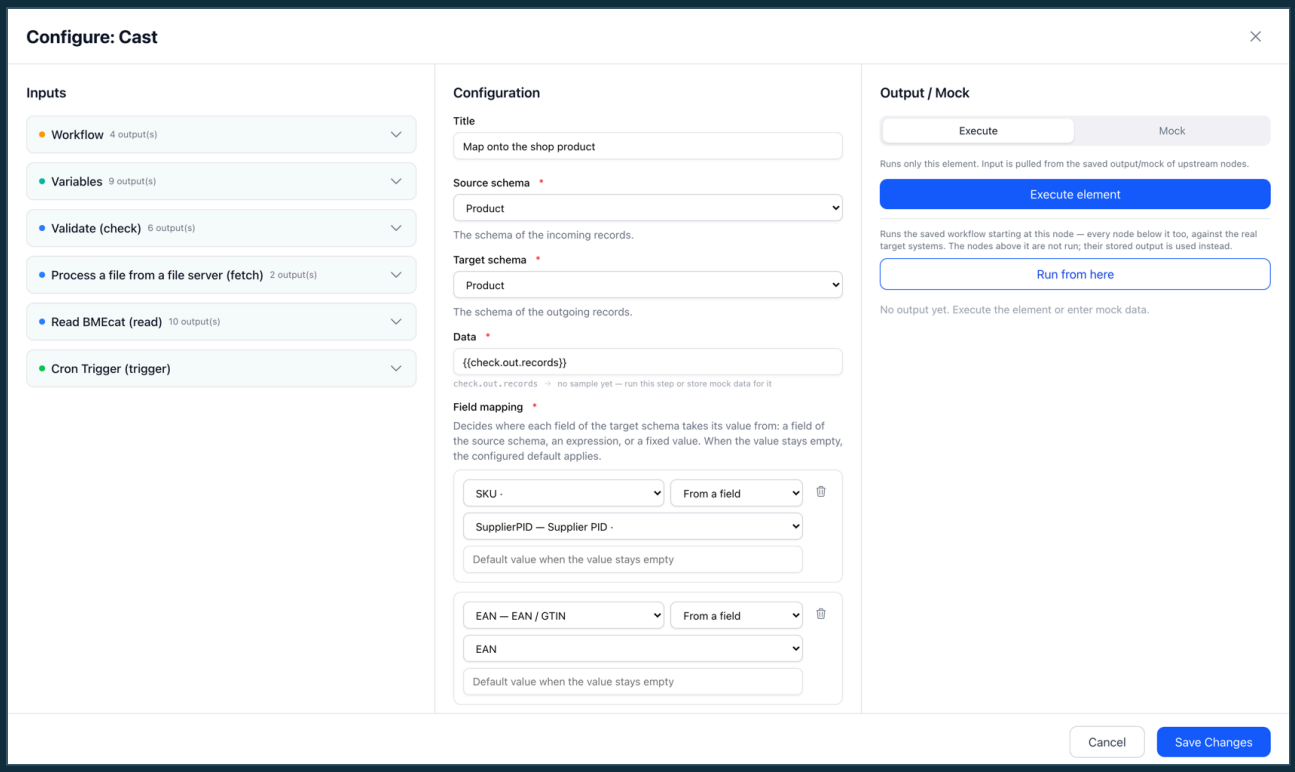
Model	Content (extract)
Product	SKU, product number, EAN/GTIN, manufacturer part number, name, description, short description, manufacturer, brand, categories, net and gross price, list price, purchase price, currency, tax rate, stock, delivery time, unit of measure, packaging and base-price unit, weight and dimensions
Category	category tree with parent-child relationships
Manufacturer	manufacturer master data
Property group / property	filter and variant characteristics
Variant	the variations of a product
Price	tiered and customer-group-dependent prices
Tax rate	tax rates and their assignment
Sales channel	the channels sold through
Customer group	price and visibility groups
Media	images and files with their assignment
Download	digital attachments to a product
Product reference	accessories, cross-selling, relationships
Product visibility	which product appears in which channel

The fourteen data models of the Commerce Bundle; property group and property share a row.



The mapping is built once against the Commerce Bundle. Which target system is written to is then decided by the last node in the graph.

> **THE PRACTICAL EFFECT**
>
> You map your source data onto this model **once** — the supplier catalogue, the ERP export, the price list. Which target system is written to is then a question of the last node in the graph. Moving from Shopware 5 to 6, adding a second channel on Shopify, or handing the same assortment to the TB.One marketplace link swaps that node and leaves the mapping untouched.



The field mapping of one node: on the left the outputs of the upstream steps, in the middle the source and target schema and the rule per field — from a field, from an expression or as a fixed value — and on the right a trial run for this step alone. This is the work a generic platform demands again for every shop system; here it is done once, against the Commerce Bundle.

5.3 Shopware 6

The most extensive integration, with seven building blocks: *write*, *read*, *media*, *file upload*, *delete*, *indexing*, and a *free API call* for anything the prebuilt blocks do not cover.

What matters in practice: Shopware 6 addresses entities through an internal 32-character identifier, while your catalogue knows article numbers. ConFlowIO maintains that mapping, so a second run hits the same products instead of creating them again. Writing goes in batches — several dozen products per operation rather than one at a time, the difference between an hour and an afternoon.

Two details that matter in daily operation: **indexing can be switched off for the duration of an import** and triggered deliberately afterwards — the usual way to stop a large import from slowing the shop down. And **deletion carries a safety limit**: a maximum permissible share of the stock, plus a dry run, prevent a faulty supplier export from removing half the range from the shop.

5.4 Shopware 5

Still in the market and relevant for migration projects. The integration takes advantage of the fact that Shopware 5 addresses articles directly by their article number — so no identifier bridging is needed, and a second run lands on the same articles without a lookup. Categories, manufacturers and tax rates are found by name. Writing happens per record rather than in batches: slower, but a failure affects one article instead of a whole batch.

Deliberately not covered: properties (Shopware 5 requires a filter-set assignment that a catalogue does not supply) and the variant configurator.

5.5 Shopify

Connected through the current GraphQL Admin API. Three peculiarities shape this integration:

- **No business key for products.** There is the internal identifier and the handle from the shop URL. ConFlowIO derives the handle reproducibly from the article number, so a follow-up import lands on the existing products; variants are found by their SKU.
- **A product is essentially its variant.** Price, barcode and weight live on the variant, not on the product. The integration models that so you do not have to rebuild it in your mapping.
- **Stock belongs to a location.** You name the location in the building block; leave it empty and no stock is written. A fallback to "the first location" would be worse — stock in the wrong warehouse is only noticed when the wrong warehouse is asked to ship.

Deliberately not covered: smart collections (defined by rule rather than membership) and category trees (Shopify collections are flat). Existing image galleries are not overwritten.

5.6 Adobe Commerce (Magento 2)

Connected through its REST API; Magento Open Source behaves identically. Products are addressed directly by SKU. What is special: a large part of a product does not sit in the product — anything beyond the core columns lives in a list of custom attributes, and stock in the extension attributes. ConFlowIO assembles all three parts from a single field table, none of which is visible in your mapping. Product images are expected as content rather than as an address, which is why downloading by the platform is off by default and size-capped per image.

5.7 Which building blocks exist per shop system

Building block	Shopware 6	Shopware 5	Shopify	Adobe Commerce
Write products, categories and master data	•	•	•	•
Read data	•	•	•	•
Write product images along with the product	•	•	•	•
Free API call	•	•	•	•
Media library as a building block of its own	•	–	–	–
Upload arbitrary files	•	–	–	–

Building block	Shopware 6	Shopware 5	Shopify	Adobe Commerce
Bulk deletion with a safety limit	•	–	–	–
Control indexing	•	–	–	–

A dot means a ready-made building block in the editor. A dash does *not* mean the shop cannot do it — the free API call reaches every endpoint of the respective shop, just without the groundwork and the safeguards of a ready-made block.

All four integrations write images — by three different routes

The *media library* row is easily misread. Product images end up in **every** one of the four shops, straight out of the write block and with no extra configuration. What differs is only how the shop accepts them: Shopware 6, Shopware 5 and Shopify are handed the *address* and fetch the image themselves — no image file travels through ConFlowIO. Adobe Commerce accepts images only as *content*, so there the platform fetches them itself, off by default and size-capped. Shopware 6's standalone media block covers something else: that shop's *media library*, with folders and files that exist independently of any product.

Deletion and indexing

A **bulk deletion with a safety limit** — at most a configurable share of the stock, with a dry run first — requires the shop to accept many records in one operation. Of the four, only Shopware 6 can. In the others, deleting is one call per record and possible through the free API call, just without the brake. **Indexing** is likewise a Shopware 6 peculiarity — the other three manage their search indexes themselves.

> **WHERE THE DIFFERENCE IN DEPTH COMES FROM**
>
> Shopware 6 is the most frequently requested integration and therefore the most built out — a matter of project experience, not of architecture. Each of these blocks is a plugin element and can be brought to any other shop without changing anything about the platform. If your project needs one, say so: a building block, not a rebuild.

5.8 Marketplaces: Tradebyte TB.One

Selling through marketplaces does not mean running another shop — it means running a **link**. Tradebyte's **TB.One** sits between merchant and marketplaces and knows exactly four exchanges on the merchant's side: catalogue out, stock out, orders in, status back. Nothing to look up, no identifier to bridge, nothing to search — which is why this integration looks different from a shop's.

In one respect it is identical, and that is the one that counts: the catalogue is built **from the Commerce Bundle**. The same mapping that prepares your supplier catalogue for Shopware or Shopify also feeds the marketplace link — in the graph, the difference is a different target node.

Every document is XML, and TB.One validates each one against a published schema: anything that does not fit is refused *whole*. ConFlowIO therefore does not assemble those documents as text but checks the records before anything is sent. A message naming the record and the field is the difference between an error someone can fix and the verdict “the document is invalid”.

The eight building blocks

Building block	What it does
Hand over catalogue	Hands products from the commerce model over as a catalogue — either the whole assortment or only the changes. Records sharing a product number become one product with several articles.
Hand over stock	Hands stock levels over in batches of bounded size: if a batch is refused, a batch is repeated rather than the entire assortment.
Read orders	Reads the open orders together with their items. The number per run can be capped; the rest stays open and arrives on the next run.
Close order	Reports that the named orders have been taken over. They are not handed out again afterwards.
Report order status	Reports shipment, cancellation, return or deductions back, per order item. An order counts as completed once every item has been reported.
Read order messages	Reads what happened after despatch: buyer cancellations, returns booked by the marketplace, and the merchant's own notices.
Fetch order document	Downloads the document the marketplace provides with an order — invoice, delivery note or shipping label — as a PDF.
Free API call	Calls any endpoint of the interface with the same credentials, for anything there is no dedicated block for.

> **WHY CLOSING AN ORDER IS A BLOCK OF ITS OWN**
>
> TB.One keeps handing an order out until the merchant confirms they have taken it. But “read” and “stored” are two different moments, and only your process knows when the second one happened. That is why closing is a node of its own, placed *after* the step that persists the order. Leave it out and every run reads the same orders again; place it too early and an order whose storage then failed is lost.

Because a marketplace order is not a piece of commerce master data, this integration brings four data models of its own — order, order item, order message and article stock. The catalogue side needs none: it uses the Commerce Bundle's models like every other target system.

Deliberately not covered: the channel side of the interface — the calls a *marketplace* makes to TB.One, which a merchant account is not entitled to use — and reading one's own catalogue back.

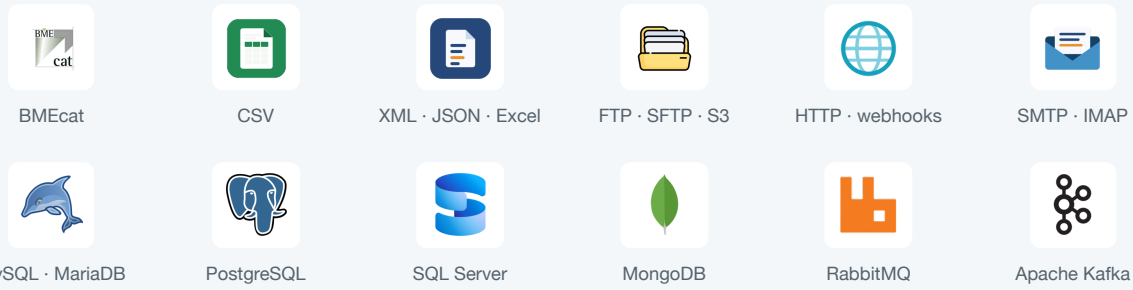
— 5.9 The systems around the shop

A shop is rarely the only counterpart. The following integrations are also part of the platform and cover what typically surrounds a shop in retail projects:

Area	What is connected
Supplier catalogues	BMEcat — the standard format for product catalogues in B2B trade, including the category tree
File formats	CSV, fixed-width records, XML, JSON, Excel (XLSX) — reading and writing
File transport	FTP and SFTP (download and upload), S3-compatible object storage (the same read blocks, a separate block for putting files there), local directories, creating and extracting archives
Images	image processing — size, format, preparation for the shop
APIs	HTTP/REST in both directions, including inbound webhooks with per-node authentication

Area	What is connected
Databases	MySQL/MariaDB, PostgreSQL, Microsoft SQL Server, MongoDB — reading and writing
Message queues	RabbitMQ and Apache Kafka, as trigger and as target
Email	sending over SMTP, mailbox monitoring over IMAP as a process trigger

AROUND THE SHOP



On top of that come eleven building blocks for data preparation itself, which in catalogue projects make up most of the graph: type conversion, splitting, filtering, validating, joining, sorting, deduplicating, aggregating, collecting — and a **change-detection block** that passes on only the records that actually changed within a full catalogue. In practice, this is the block that turns a nightly full synchronisation into an hourly one.

CHAPTER 6

Architecture and execution

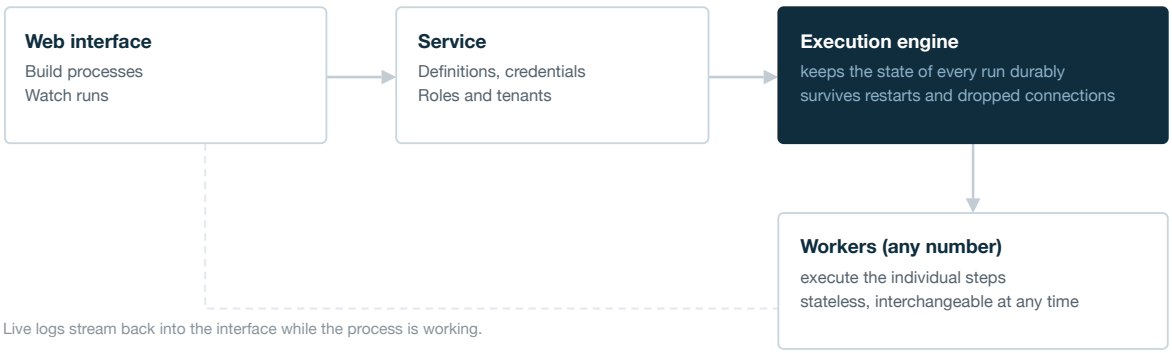
Where the reliability comes from, and where your insight comes from.

This chapter explains what the promises in Chapter 4 rest on technically. It can be skipped without loss to the rest of the paper.

— 6.1 Definition and execution are separate

In operation, ConFlowIO consists of three parts: a **web interface** in which processes are built and observed; a **service** that manages definitions, credentials and permissions; and any number of **workers** that do the actual work. Between them sits an execution engine that keeps the state of every run durably.

The benefit: the interface stays responsive while an import of a hundred thousand records runs in the background, and throughput grows with the number of workers — a configuration value, not an architectural change. Workers are stateless; any of them can take on any task.



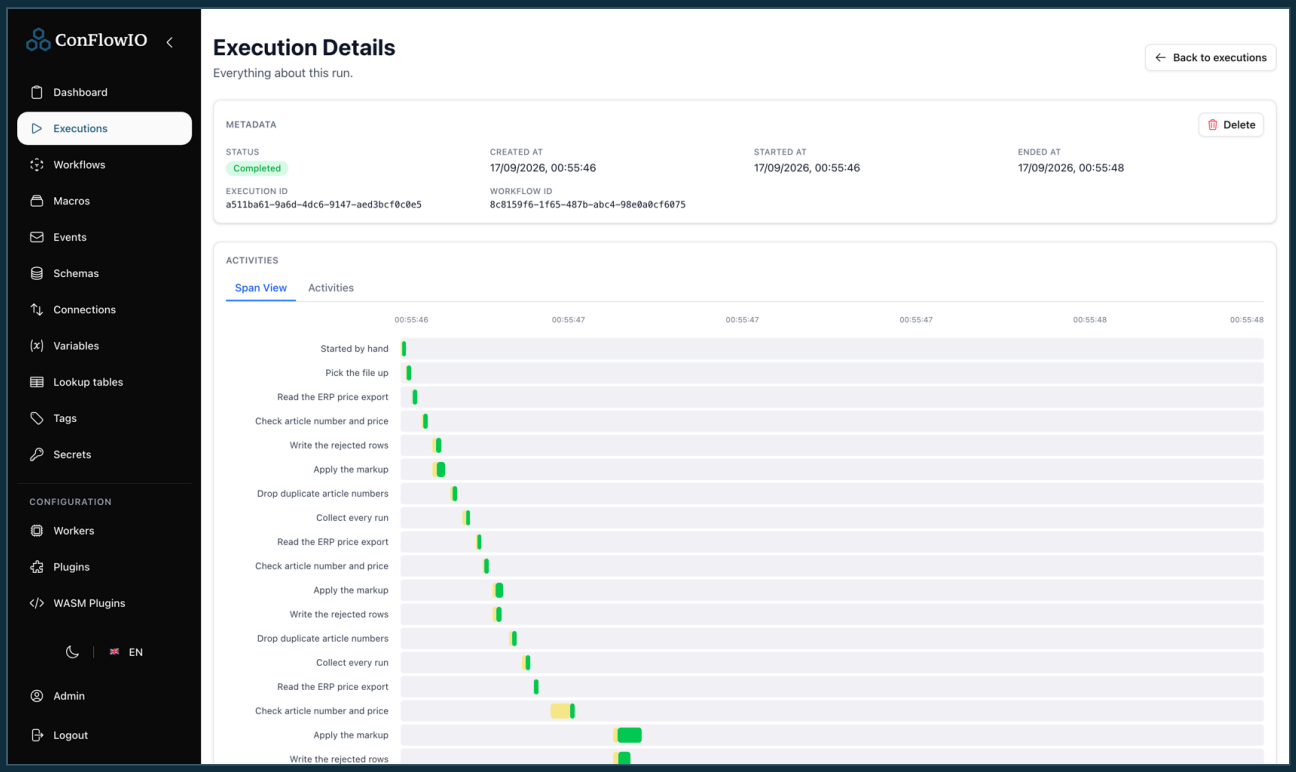
Definition and execution are separate: the interface stays responsive while a hundred thousand records are processed in the background.

— 6.2 How a run survives a failure

The state of a run does not live in one process's memory; it is kept durably. In practice that means:

- **Failed steps are retried automatically** — up to three times with growing intervals, adjustable per node. The briefly unreachable shop resolves itself.

- **Completed steps never run again.** A graph that fails at the fifteenth node repeats that node — not the fourteen before it.
- **Large reads run page by page** (see 6.3), and each page is a step of its own: fail at page 90 and page 90 is repeated.
- **A retry is safe because writes go through the business key.** A second attempt lands on the same products instead of creating them again — which is why Chapter 5 goes into recognition in such detail.
- **A server restart does not end a run** — another worker takes over. User cancellation is cooperative: the running step finishes in an orderly way instead of vanishing mid-write.



A single run: above, the timeline of every step with its duration; below, the same graph with the state of each node. A read node that delivers its data page by page appears once per page — five times here. This is what makes "where did the run stop?" an answerable question.

On demand: continuing a failed run

If a run fails for good, it can be **continued rather than started over**. The continued run skips every branch that finished the first time: a catalogue import that broke off at record 6,342 does not write the first 6,341 a second time. And it takes over the failed run's working files — whatever had been downloaded and parsed does not have to be fetched and read again. That is the difference between a continue that costs seconds and one that costs the original forty minutes all over again.

Two properties make that dependable. A finished record is recognised by its **content** rather than its position, so a record added to or removed from the source does not throw the matching off. And **every building block declares for itself** whether it may take part in a continued run; a single one that may not refuses the continue and is named when it does. Better a refused continue than one that silently leaves records out.

What a continued run deliberately does *not* skip is the path to the outstanding records: reading and parsing run again, because their result is what produces the records at all. And a run is continued once — to carry on after that, continue the continued run.

6.3 Large volumes: page by page, not all at once

The usual reason automation tools fail on catalogue data is a size limit on how much data a single step may hand on. A catalogue of 200,000 articles exceeds it.

ConFlowIO lets reading blocks deliver **page by page**: one page out, the downstream steps process it, then the next — until the source is exhausted. Any size limit then applies per page rather than to the whole data set. For processes that should act only at the end — "send a summary once every page is through" — there are connections that fire only after the last branch, and only if no branch failed.

6.4 Triggers: six ways to start a process

Trigger	Use
Manual	start from the interface, optionally from a specific node — for testing and troubleshooting
Schedule	recurring by time rule, or once at a chosen moment
Webhook	its own address per node; authentication per node via key, bearer token, basic auth or signature check — unknown schemes are rejected, not let through
Event	one process triggers another, including across system boundaries
Message queue	an incoming message from RabbitMQ or Kafka
Mailbox	an incoming email over IMAP, e.g. for supplier files in an attachment

When a trigger fires while work is still running

With hourly synchronisations the case arises regularly that the next scheduled time comes round before the running synchronisation has finished. What happens then is decided by the process, not by the trigger — a choice between *skip* (the default; the skipped start is still recorded as such, because a trigger that silently does nothing is indistinguishable from a broken one), *queue*, *cancel the running one* and *allow in parallel*. That is what keeps two runs from writing to the same shop at the same time.

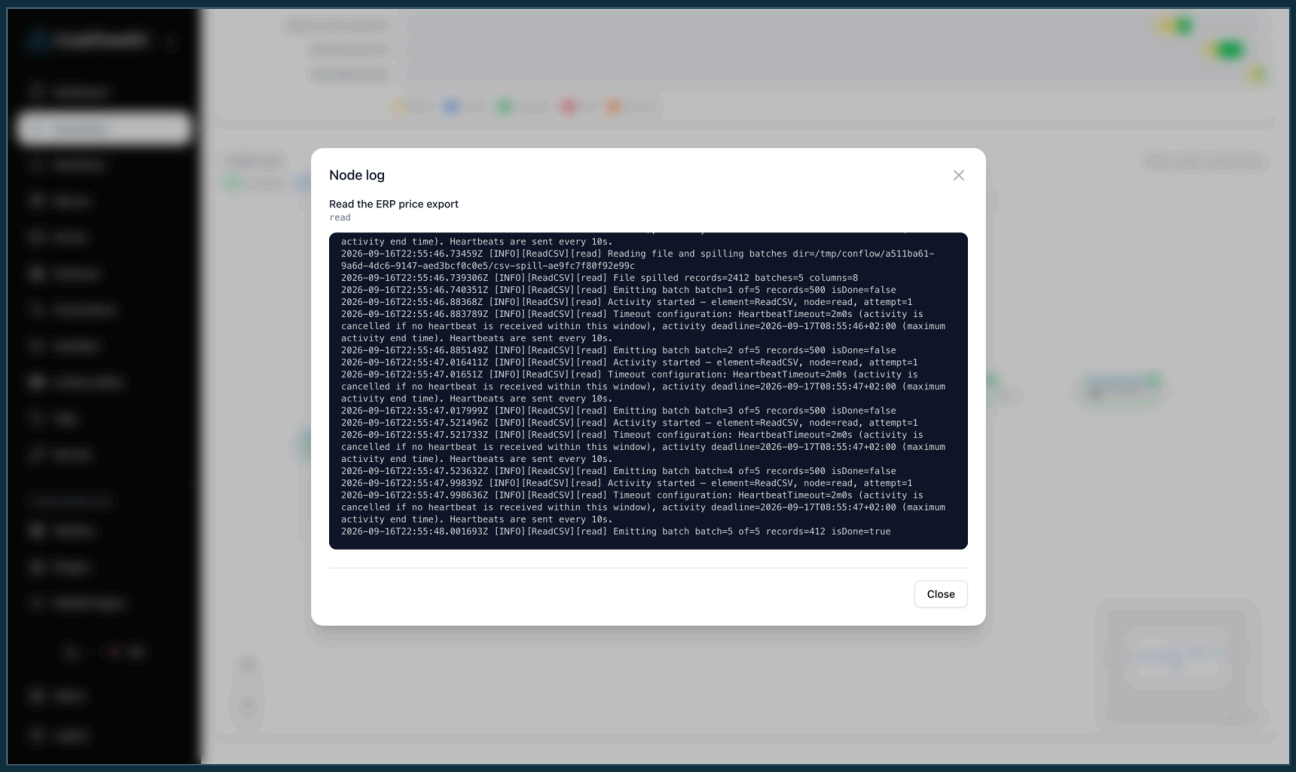
6.5 Reuse: macros

Anyone running five supplier imports has the same price calculation and the same check five times over. That becomes a **macro**: a sub-process built in the same editor, with a defined input and output, appearing in any number of processes as a single node. A change to the macro takes effect everywhere; macros may be nested.

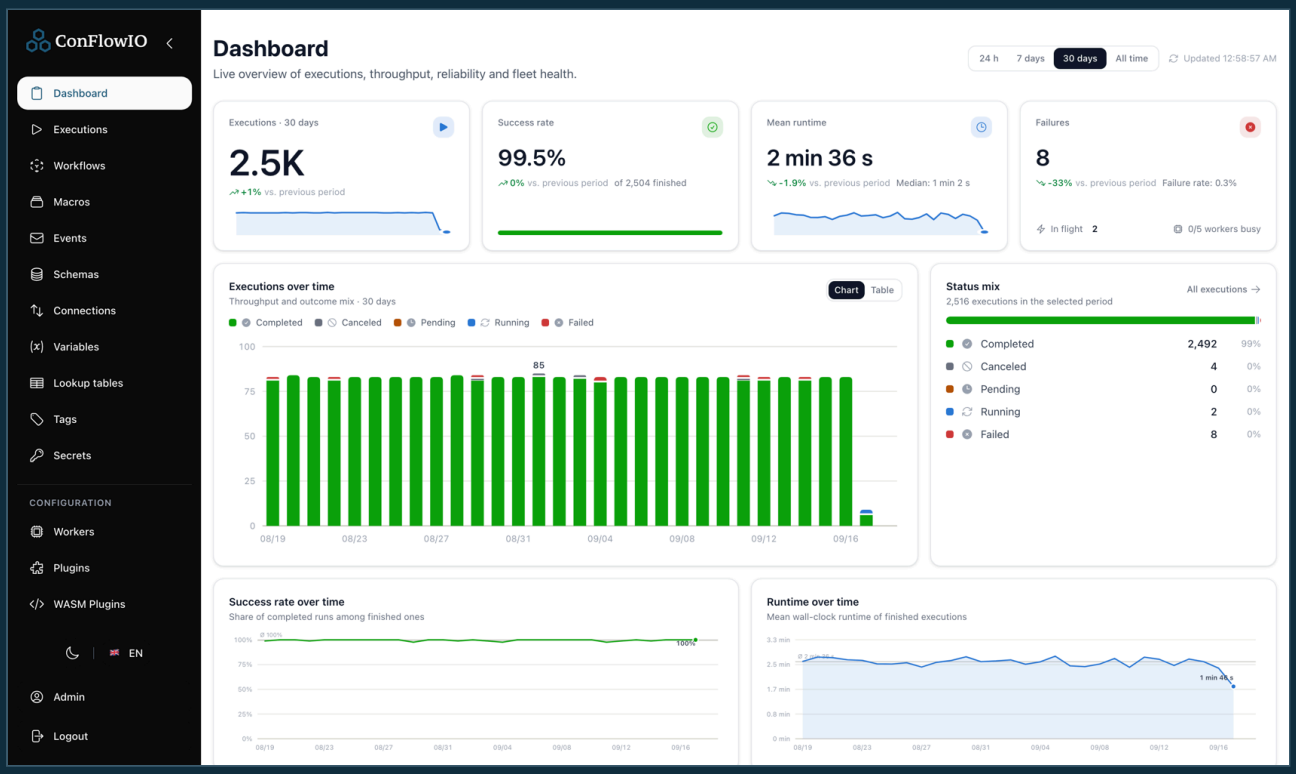
6.6 Visibility

Every run is individually traceable: which nodes ran, how long they took, what each step logged. Logs stream **live** into the interface while the process is working — not only at the end. That is the difference between "the import has been running for two hours, no idea where it is" and a dependable statement.

Credentials are stripped from every log line and error message, including those returned by a remote system.



The log of a single step, opened straight from the node: the read reports 2,412 records across five passes and states per pass how many records were handed on. Lines like these are the difference between "the import is running" and a dependable statement about what it has done.



The overview of an installation: how many runs, how many of them succeeded, how long they take and how busy the workers are — each for the selected period. Every bar leads into the individual run and its log. Below sit the busiest processes, the slowest ones and the most recent failures. The interface comes in light and dark.

6.7 Extensibility

Should a building block be missing, there are three routes in ascending effort: the **code block** directly in the graph for one-off special logic; a **custom plugin** loaded at runtime into an isolated environment; and extending the bundled blocks when an integration should become a permanent part of the platform. In addition, **custom data models** can be defined — including ones derived from the bundled models with individual fields overridden.

CHAPTER 7

Security, tenants and data sovereignty

Where your credentials live and who can see them.

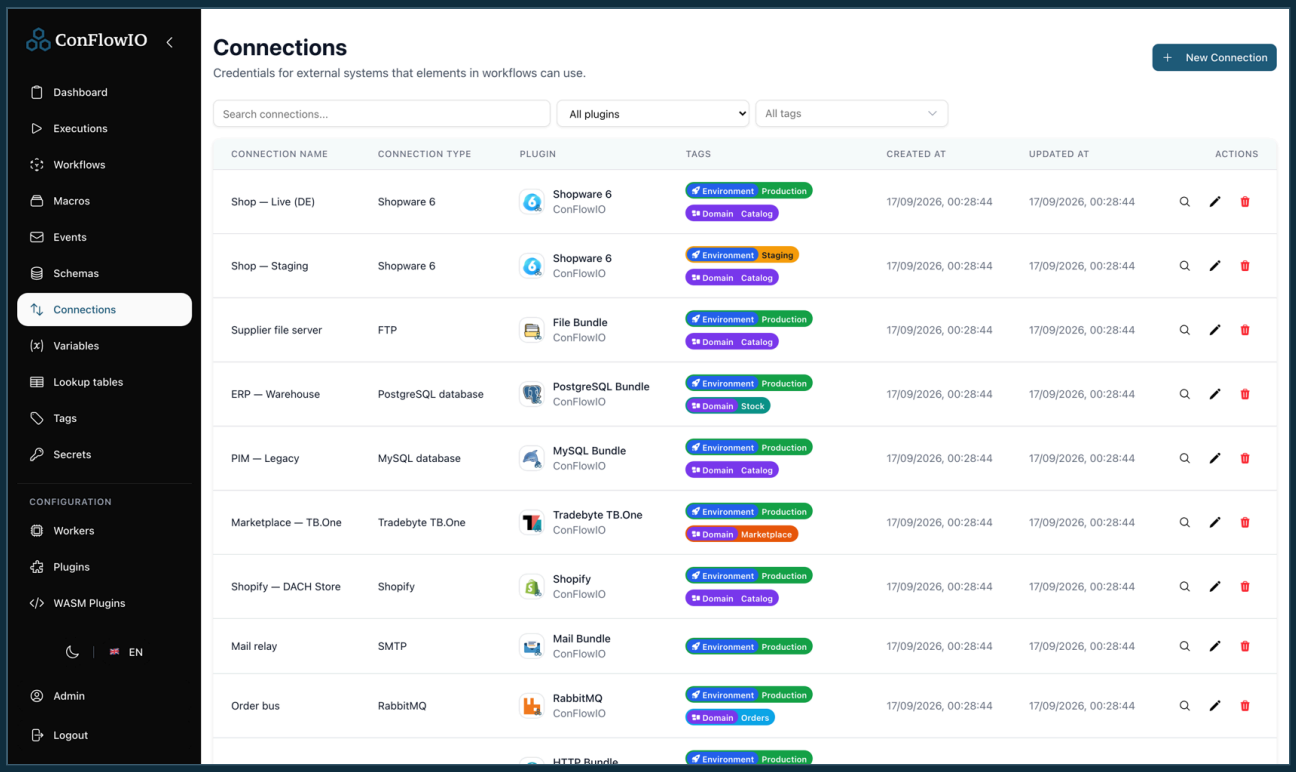
07

An integration platform is where the credentials to every other system live. That makes it the most interesting system in the house from a security point of view — and it is routinely the one examined most superficially during selection.

— 7.1 Credentials

Credentials live in exactly one place and are **stored encrypted** there. Processes and connections never store the password itself, only a reference to it; the plaintext is substituted only when a run starts and is removed from every log output.

Reading a stored credential back is possible through **exactly one route**, and that route demands the account password again — a valid session alone is not enough. A stolen session token therefore does not suffice to read out the credentials to your shops.



Connections to the attached systems in one place, labelled by environment and domain. The password itself appears in none of these rows: a connection stores only the reference to the stored credential.

— 7.2 Tenant separation

One installation can hold several completely separate workspaces — relevant for agencies, system integrators and groups with several brands. Every process, every connection, every credential and every run belongs to exactly one workspace and is invisible from all others.

Important for evaluation: the separation **fails closed**. A query naming no workspace is refused rather than answered more broadly — a forgotten filter produces an error, not somebody else's data. For installations with a single tenant the mechanism is invisible.

— 7.3 Access and traceability

- **Roles** at workspace level, enforced at the API — not merely by hiding menu entries.
- **Second factor** via time-based one-time codes or passkeys.
- **Lockout after failed sign-in attempts**, per account.
- **Audit log** covering administrative actions.
- **Access token in memory**, not in browser storage; the refresh token exclusively in a cookie inaccessible to scripts.

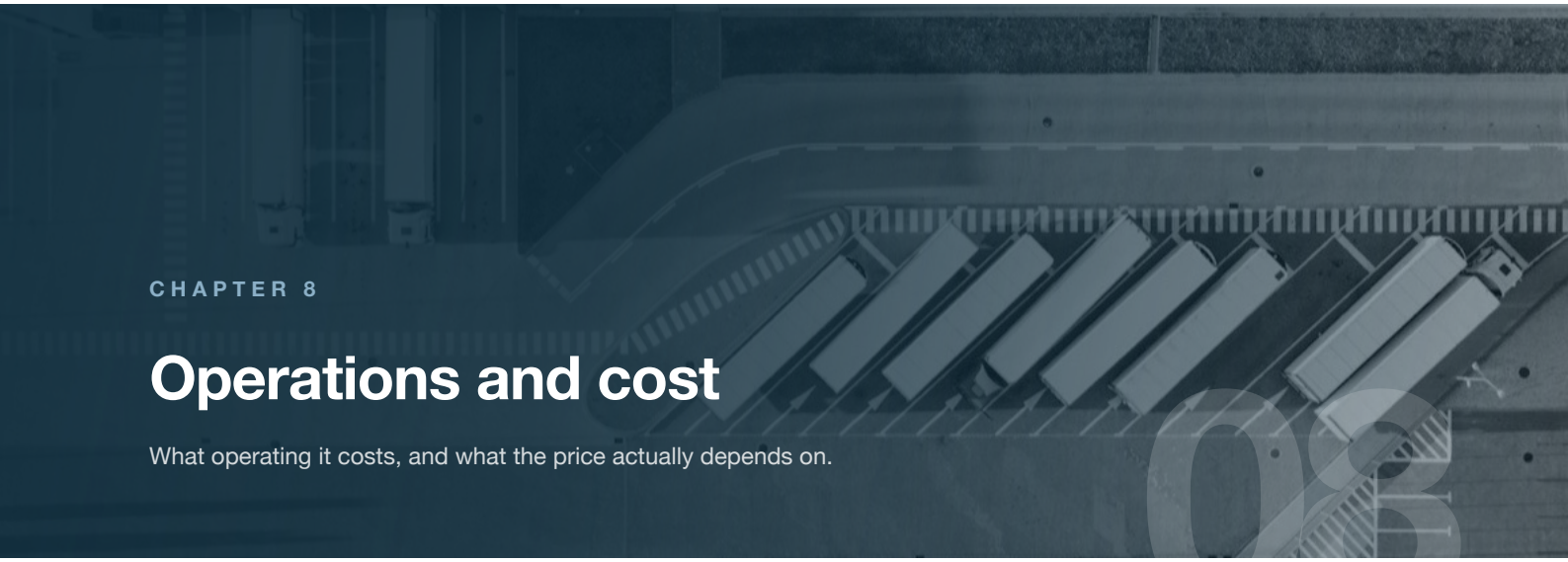
7.4 Operating model and data sovereignty

Where does the data live, who has administrative access, which jurisdiction does the vendor fall under, what happens on an official request? For most companies the managed platform is the right answer — it takes operational responsibility off your hands, and everything in 7.1 to 7.3 applies unchanged.

The vendor and the hosting are both in Germany. Your supplier prices, margins, customer data and the credentials to your systems therefore sit in German infrastructure and do not leave the European legal area. The question that trips up the third-country vendors in section 2.2 simply does not arise for ConFlowIO — and it does not arise in the standard managed setup, without you having to run anything yourself.

Where the requirement is stricter — group policy, certification, particularly sensitive data — ConFlowIO is one of the few platforms in this market category that can respond at all: the same software runs on request entirely inside your own infrastructure. You do not have to make that decision at selection time, and you do not have to regret it later.

> **ASK EVERY VENDOR THIS**
>
> "Can you also run the platform in our data centre if we need you to?" is the question on which this market divides.[2] The pure cloud vendors have to answer no; ConFlowIO answers yes.



CHAPTER 8

Operations and cost

What operating it costs, and what the price actually depends on.

8.1 Two operating models, one platform

The managed platform is the default. Installation, updates, backups and monitoring are ours; you sign in and build the first process. There is no server to set up, no database to back up and no key to manage — the fastest route from decision to a productive import.

On request, the same platform runs on your side. Two routes are documented for that, both intended for production:

- **Docker Compose** — for operation on one or a few servers. The usual route for mid-market installations. Designed to run behind an existing reverse proxy with TLS termination, with resource limits, restart policies and log rotation.
- **Kubernetes** — as Kustomize manifests with a base and a production overlay, suitable for GitOps operation. Containers there run without elevated privileges.

PostgreSQL or SQLite may then serve as the database. Both the database *and* the encryption key for credentials must be backed up together, because without the key stored credentials cannot be recovered; a dedicated procedure re-encrypts them when the key is rotated. In the managed model all of that falls away.

8.2 Scaling

Execution throughput scales with the number of workers, which distribute the pending work among themselves. In the managed model that grows with your volume without any action on your part; when self-hosted it is a single configuration value. Either way there is no limit on the number of workers, processes or users.

8.3 Pricing and licensing

What is billed is the **capacity** provided. If your volume outgrows what the agreed environment carries, the environment grows with it — and so does the price. That is honestly true of ConFlowIO as well: more transactions need more compute, and compute costs money. What the model expressly does *not* do:

- It counts no records, runs or operations — within a given capacity it makes no difference how often you use it.
- It does not cap the number of connectors, processes or users.
- It unlocks no features that cost extra — the feature set is the same for everyone.

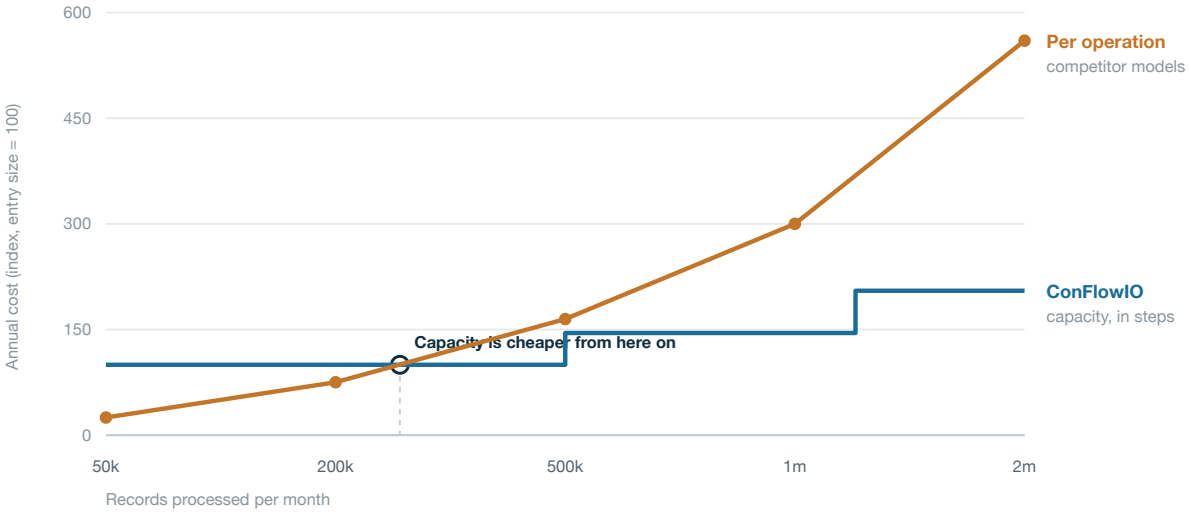
That defuses the most expensive question in Chapter 3: how often a process runs is a technical decision for as long as the capacity carries it — and when it no longer does, you see it coming instead of discovering it on the invoice. Nothing is switched off without warning: when a term expires, a **grace period of 14 days** follows in which everything continues unchanged, and the interface points it out thirty days beforehand.

— 8.4 Total cost of ownership

A dependable price comparison is difficult, because the e-commerce specialists do not publish prices.[8] What can be compared is the *structure* of the cost:

Model	What drives the price	Behaviour as you grow	Predictability
Per operation (automation tools)	number of records processed	rises proportionally	low
Connectors + volume (e-commerce iPaaS)	connected systems and operations	steps up in tiers	medium
Enterprise licence (enterprise iPaaS)	contract negotiation	rises at renewal	medium
In-house build	person-days	rises with each interface	low
ConFlowIO	**the capacity provided**	**rises in a few steps**	**high**

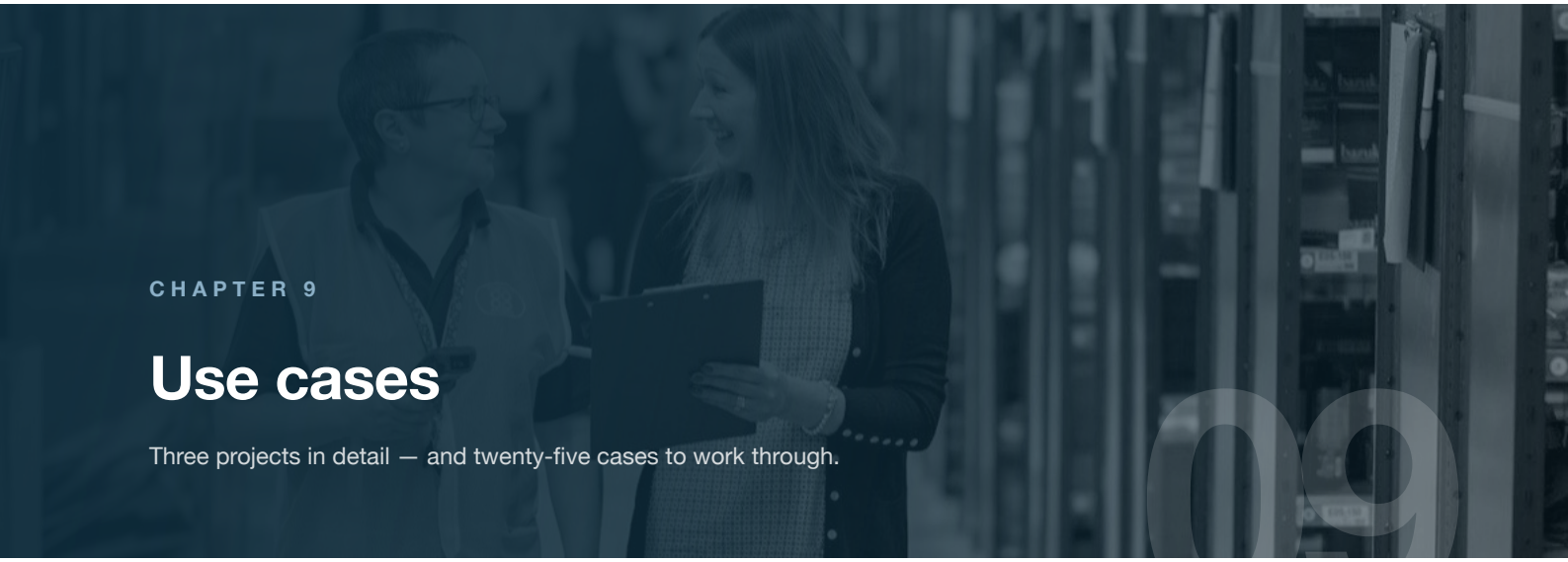
Cost structure by model — not price level, but behaviour as volume grows.



Schematic, not a price statement. ConFlowIO's price rises with volume too — but in steps, because it follows the capacity provided rather than the individual record. Within a step, an additional synchronisation costs nothing.

THE QUESTION THAT SEPARATES VENDORS

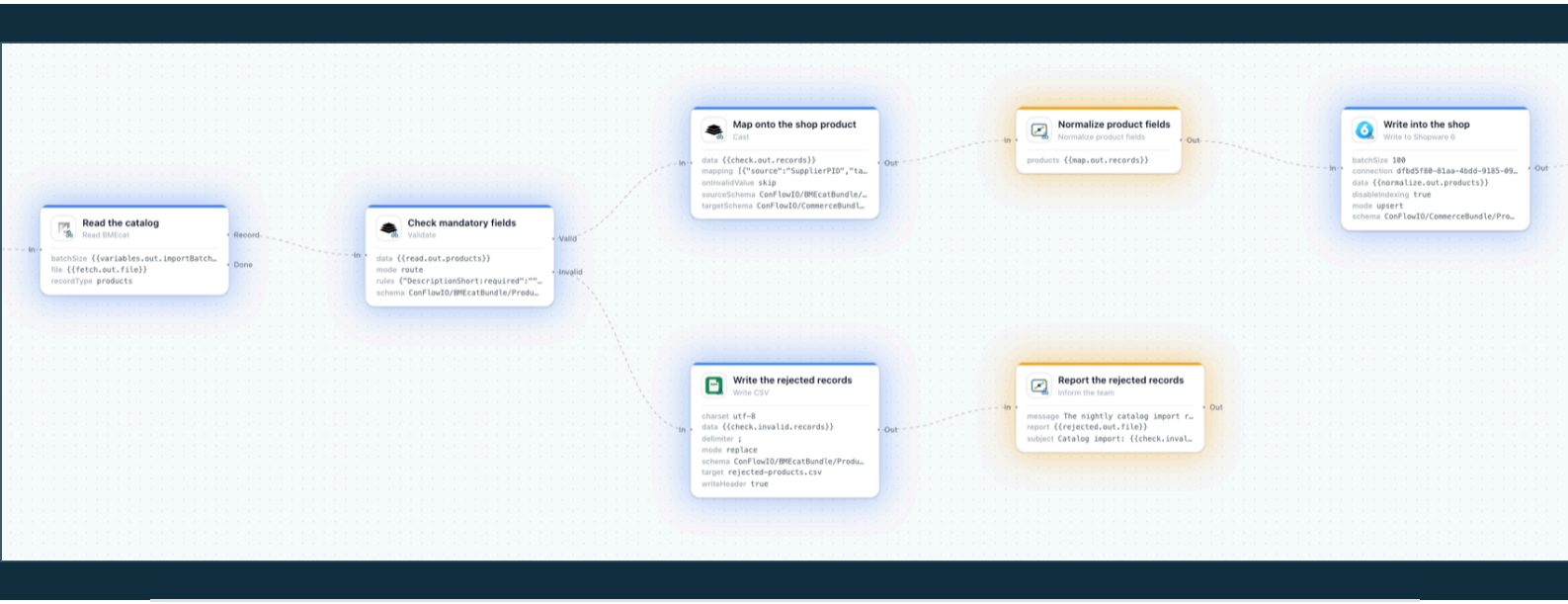
Work out your most expensive planned process — usually the hourly stock and price synchronisation across the whole article master — and ask every vendor what *that one process* costs per year. In our experience the answers differ by orders of magnitude, independently of list price.



CHAPTER 9

Use cases

Three projects in detail — and twenty-five cases to work through.



A catalogue import as it actually looks: the catalogue is read and checked; what passes the check is mapped onto the shop product, normalised by a macro and written to Shopware 6. What fails leaves the run through an output of its own, is written to a file and reported to purchasing — instead of stopping the import.

— 9.1 Supplier catalogue into the shop

Starting point: 40 suppliers, each with its own format and its own rhythm.

TRIGGER	READ	MACRO	WRITE
Schedule, or an incoming file over SFTP	BMEcat, CSV or Excel — per supplier	Price logic and checks, identical for all	Shopware 6, in batches

The effort is not in the reading but in the shared part that follows: markups, rounding rules, category assignment, mandatory-field checks. That part is a macro shared by all 40 processes.

> **OUTCOME**
>
> A new supplier is one read node and a field mapping — a morning rather than a sprint. And a change to the price logic applies to all forty imports at once.

9.2 Stock and prices on an hourly cycle

Starting point: the ERP holds stock, three sales channels must follow.

TRIGGER
Schedule, hourly

→

READ
ERP database, page by page

→

CHANGES
only records that actually changed

→

CHANNELS
Shopware 6, Shopify, marketplaces over TB.One

OUTCOME

Of 200,000 articles, a few hundred change each hour — only those are written. The shop shows stock levels that are correct, and the hourly cadence stays a technical decision rather than a cost one.

9.3 Orders back into the ERP

Starting point: orders should reach the ERP without delay and without polling.

TRIGGER
Shop webhook, authenticated per node

→

VERIFY
Signature, structure, plausibility

→

TRANSFORM
Addresses, line items, payment methods

→

WRITE
ERP over REST or database

OUTCOME

No order is lost and none is re-entered by hand. If the ERP is briefly unreachable the step retries itself; if the run fails for good, the overview says which order it concerns and what went wrong.

9.4 Twenty-five use cases

The cases below are not a list of ideas but what retail projects actually build. Each one is made of bundled building blocks; the right-hand column names the chain they form in the editor. Whichever of them sounds familiar is the best place to start a conversation.

Catalogue and product data

Use case	What the process looks like
1 · Supplier catalogue into the shop The supplier drops a BMEcat overnight; by morning the range is in the shop.	Schedule → fetch file over SFTP → read BMEcat → check → map onto the commerce model → Shopware 6
2 · Price or article list as a file The same route for suppliers who send CSV or Excel instead of a catalogue format.	Schedule → read CSV or XLSX → check → commerce model → shop
3 · Reduce a full catalogue to what changed The supplier always sends everything; only the differences are written.	read → change detection against the last state → write only changed records
4 · Migration from Shopware 5 to Shopware 6 The old shop is the source, the new one the target — with no export in between.	read Shopware 5 → prepare → write Shopware 6
5 · A second channel without a second mapping The same range additionally in another shop system.	existing graph → second target node (Shopify or Adobe Commerce)

Use case	What the process looks like
6 · Build the category tree from the catalogue The supplier's catalogue groups become the navigation in the shop.	read catalogue groups → map onto categories → write the parent-child relation
7 · Normalise manufacturer names and units Three suppliers, three spellings of the same brand — maintainable without opening the process.	look the value up in a lookup table during mapping → write it normalised
8 · Prepare and hand over product images Images arrive in whatever size they arrive in, and should look the same in the shop.	read the image list → fetch the files → resize and strip metadata → media in the shop
9 · Set aside articles missing mandatory fields The import still completes; the gaps go to purchasing rather than into the shop.	check → write the valid records → rejected ones as CSV → email to purchasing

Stock, prices and marketplaces

Use case	What the process looks like
10 · Hourly stock synchronisation The shop shows stock that is correct — not last night's.	Schedule, hourly → read the ERP database page by page → change detection → write to the shop
11 · Prices with markup and rounding rules Purchase price in, sales price out — the same rule for every supplier.	read → price logic in a macro → write to shop and marketplace
12 · Customer groups and what they see The B2B case: who sees which range in which channel.	write customer groups and sales channels → set product visibility per channel
13 · Stock to the marketplace An oversell on a marketplace costs more than one in your own shop.	read stock → change detection → hand stock over to TB.One
14 · Catalogue to the marketplace The same range that goes into the shop also goes onto the marketplace.	commerce model → hand the catalogue over to TB.One
15 · Remove discontinued articles With a safety limit, so a faulty export cannot delete half the range.	read the target state → build the difference → dry run → delete with an upper bound

Orders and everything after them

Use case	What the process looks like
16 · Shop order into the ERP The moment it is placed — no polling loop every five minutes.	shop webhook → verify the signature → transform → ERP over REST or database
17 · Collect marketplace orders Closed only once the order has actually been stored.	Schedule → read orders → write into the ERP → close the order
18 · Report shipment status back The buyer sees the tracking number without anyone typing it in.	read despatch data → report the status per order item to TB.One; into the shop through the free API call
19 · Read returns and cancellations What happens after despatch lands in the ERP rather than in a mailbox.	Schedule → read order messages → post them in the ERP

Use case	What the process looks like
20 · File order documents Invoice, delivery note and shipping label where logistics looks for them.	read the order → fetch the document → place it on SFTP or in object storage

Operations, data and special cases

Use case	What the process looks like
21 · Supplier file from a mailbox For the supplier who sends the price list as an attachment.	watch the mailbox over IMAP → take the attachment as a file → as with any other import
22 · Nightly export for accounting or field sales A file somebody actually opens, instead of an account nobody uses.	Schedule → read the data → write it as Excel → place on SFTP and send by email
23 · Database to database Reporting on a database of your own, without anyone querying the ERP.	read the ERP database page by page → transform → write the target database
24 · An unknown supplier format The format no vendor has a connector for.	read the file → code block in the graph → from here on like any other import
25 · Several tenants in one installation For agencies and corporate groups: separate workspaces, one platform.	one workspace per customer, with its own processes, connections and credentials

> **WHAT THIS LIST SAYS ABOUT THE PLATFORM**
>
> Twenty-five cases, and not one of them is a detour: always the same building blocks in a different order, always the same commerce data model in the middle, always the same execution underneath. That is why the second process is built considerably faster than the first — and the twenty-fifth is no more of a project than the second.

CHAPTER 10

Selection criteria for your project

Twelve questions for every vendor, ConFlowIO included.

10

The twelve questions below apply to any vendor — ConFlowIO included. Take them into your tender unchanged if you like.

— Operating model and law

- ☐ Does the vendor offer *both* operating models — managed and in our own house — or does it lock us into one permanently?
- ☐ In which country is the vendor established, and which jurisdiction does it fall under — independently of where the servers are?
- ☐ What happens to our running interfaces if we end the contract, or if the vendor discontinues the product?

— Cost

- ☐ What does our highest-volume process cost per year — calculated concretely, not as a list price?
- ☐ Which quantity drives the price: records, connectors, users, processes?
- ☐ What happens to the price if we move a process from daily to hourly?

— E-commerce capability

- ☐ Is there a bundled data model for commerce data, or do we rebuild the mapping for every shop system?
- ☐ How does a second run recognise the products it already created — and what happens if that recognition fails?
- ☐ How are variants, category trees, media and channel-dependent visibility represented?

— Operational safety

- ☐ What happens to a run interrupted by a server restart at record 80,000 of 120,000?

- ☐ Are logs visible during the run or only afterwards — and are credentials reliably removed from them?
- ☐ How is an accidental mass deletion in the target system prevented?

> **RECOMMENDATION FOR THE SELECTION PROCESS**
>
> Do not accept a demo flow with five test records. Ask for your real catalogue extract — at least 10,000 articles, with variants and images. The difference between vendors only shows up there, and it shows up immediately.

Conclusion

The market for integration platforms is well populated, but organised along business models rather than along the requirements of mid-market retail. That is why something is always left open: the commerce data model, the catalogue volume, the cost logic, or the question of where the platform is allowed to run.

ConFlowIO closes those points together — as a managed platform that is ready on day one, and on request inside your own data centre; with fourteen ready-made commerce data models and four natively connected shop systems; with an execution model that survives a failure instead of creating rework; and with a price that follows capacity rather than the individual record — predictable in steps instead of growing with every synchronisation.

For a retailer with real catalogue volumes, several channels and limited IT capacity, our reading of this market makes that the best available choice. The twenty-five cases in Chapter 9 show where to start; the twelve questions in Chapter 10 verify it against any vendor, ConFlowIO included.

Next step

We are happy to show ConFlowIO against your own data. Bring a real catalogue extract — the questions in Chapter 10 answer themselves fastest that way.

conflowio.com · info@conflowio.com

— Appendix A — Connected systems and building blocks

Group	Building blocks
Shop systems	Shopware 6 · Shopware 5 · Shopify · Adobe Commerce (Magento 2)
Marketplaces	Tradebyte TB.One — handing over catalogue and stock, retrieving orders, status reports, order documents
Commerce data model	Commerce Bundle with 14 data models

Group	Building blocks
Catalogues	BMEcat incl. category tree
File formats	CSV · fixed-width · XML · JSON · Excel (XLSX)
File transport	FTP · SFTP · S3-compatible storage · local directories · archives · image processing
APIs	outbound HTTP/REST · inbound webhooks with per-node authentication
Databases	MySQL/MariaDB · PostgreSQL · Microsoft SQL Server · MongoDB
Messaging	RabbitMQ · Apache Kafka — as trigger and as target
Email	SMTP sending · IMAP mailbox monitoring as a trigger
Data preparation	type conversion · splitting · filtering · validating · joining · change detection · sorting · deduplicating · aggregating · collecting · code
Process control	manual start · schedule · one-off execution · webhook · event · macros
Extension	custom plugins in an isolated runtime · custom data models with inheritance

Appendix B — Sources and method

The competitive assessment in Chapter 3 is based on publicly available vendor information and market comparisons, retrieved in September 2026. Price figures are indicative values from third-party comparisons; binding prices come from the vendors themselves. Statements about ConFlowIO refer to the product as of September 2026. Brand and product names belong to their respective owners. Photography: Unsplash (Unsplash License, commercial use permitted without attribution).

1. Frends: *11 best iPaaS platforms in 2026, compared by features, pricing and governance.* frends.com/insights/best-ipaas-platforms-2026-compared
2. Elest.io: *Digital Sovereignty in 2026 — How EU Data Residency Laws Are Driving the Self-Hosting Boom;* Frends: *The best integration platforms (iPaaS) for European businesses.* blog.elest.io · frends.com/insights
3. Peliqan: *12 Best Workflow Automation Tools Compared (2026);* Celigo: *n8n alternatives — Enterprise-grade platforms for integration needs.* peliqan.io/blog/workflow-automation-tools · celigo.com/blog/n8n-alternatives
4. n8n: *Sustainable Use License.* docs.n8n.io/sustainable-use-license
5. Patchworks — product profile and classification. rfp.wiki/retail-ecommerce/e-commerce-integration-software/patchworks
6. Alumio: *3 Great Shopware 6 Integrations;* commerce product page. alumio.com/blog/3-great-alumio-ipaas-shopware-6-integrations
7. Synesty: *The central e-commerce interface between shop and inventory management.* synesty.com/de/explore/usecase/e-commerce-connector
8. G2 / Capterra / RFP.wiki: pricing overviews for Patchworks, Celigo and Alumio, 2026. g2.com/products/patchworks/pricing · capterra.com/p/231811/Alumio/pricing
9. Lobster: *Lobster Data Platform — platform overview;* OMR Reviews: Lobster-data. lobstersoftware.com/en/platform-overview · omr.com/en/reviews/product/lobster-data