



PLAYBOOK

Integration Architecture

Who owns what, and what happens at the seams

Ownership first, patterns per workflow, and a failure plan written before the first incident rather than after it.

CommerceStudio.ai

2026 Edition

An outdoor recreation vehicle manufacturer I worked with ran several brands under one roof. Each brand had its own ERP, its own logistics, its own operations team, and its own idea of what a customer record was. Nobody had done anything wrong. Each brand had grown up that way, and each one guarded its data for reasons that held up under questioning.

The ask was to build common infrastructure across all of them without flattening any of it. Shared where sharing paid for itself, separate where separation was the whole point, and no brand able to see another brand's numbers.

That's the real work in integration architecture. Deciding who owns what, who's allowed to see it, and what happens at the seams. Moving data between systems is the easy part, and it's the part everybody starts with.

This playbook starts somewhere else.

The question nobody writes down

Two systems disagree about a customer's credit limit. Which one is right?

If you can't answer that in one sentence, you don't have an integration architecture. You have a set of connections that happen to work today because nobody has stress-tested them yet.

Most integration projects are scoped as a list of connections. ERP to commerce. PIM to commerce. Commerce to OMS. Each one gets built, each one works in isolation, and the architecture quietly becomes whatever those connections add up to. Then someone changes a field in the ERP and three things break in a way that takes two days to trace.

Ownership is the thing you decide first, and it's the cheapest decision you will ever make. Retrofitting it costs more than any other correction in this document.

Three roles, and every system in your stack gets exactly one of them per data domain:

Systems of record hold the truth. Your ERP owns pricing and inventory. Your PIM owns product attributes. Your CRM owns the account relationship. One owner per domain, written down, and the write direction is explicit.

Systems of engagement are where people touch the data. The storefront, the buyer portal, the sales app. They read constantly and they should almost never own anything.

Systems of intelligence consume data to decide things. Analytics, personalization, forecasting, whatever AI you're running. They own nothing and they need everything, which makes them the first place bad data quality becomes visible.

Back to the multi-brand problem: that framing is what made it solvable. Each brand kept its own ERP as system of record. The shared layer was engagement and intelligence, built once, with brand isolation enforced at the data layer rather than by policy.

Common infrastructure, independent operations, and no brand could query another brand's data because the architecture didn't permit it, not because a document asked them not to.

Pick the pattern per workflow

There's no house style here. Different flows want different patterns, and using one pattern for everything is how architectures get either brittle or overbuilt.

Pattern	Best for	Strengths	Tradeoffs	Watch for
Point-to-point	Low-volume, stable links between two systems	Simple, fast, cheap	Tight coupling, nothing reusable, painful at scale	Proliferation past five or ten connections
Orchestrated (iPaaS)	Multi-system flows needing transformation and sequencing	Central control, strong transforms, audit trail	Single point of failure, licence cost, needs skilled staff	Overbuilding for simple flows
Event-driven	Real-time and near-real-time where coupling must stay loose	Scales, resilient, decoupled, async by nature	Higher complexity, eventual consistency is a real problem	Event storms, no ordering guarantee
Batch	Bulk work that isn't time-sensitive	Low complexity, predictable load, well understood	Latency, stale data, large blast radius on failure	Jobs that fail silently
Hybrid	Almost every real commerce architecture	Right-sized per workflow, pragmatic	Needs governance or the patterns sprawl	Pattern choices that live only in someone's head

Most of you will land on hybrid, which is correct. The discipline is writing down why each workflow got the pattern it got. That document is worth more than the architecture diagram, because in two years the diagram will be wrong and the reasoning will still be useful.

Every integration fails. Plan the failure.

This is the section teams skip and then rebuild the architecture because of.

Integration designs assume the happy path. The API responds, the message arrives, the record saves. Then a vendor has an outage at 2am, or a rate limit changes without notice, or somebody's certificate expires, and you find out what your architecture does under failure, which in most cases is nothing useful.

Five things worth having before you go live, not after the first incident:

Idempotency. Can the same message process twice without creating two orders? If you have to think about it, the answer is no.

Retries with backoff. Immediate retries against a struggling system are an attack on your own infrastructure. Back off, and cap it.

Dead-letter queues. Failed messages need somewhere to go and a person who looks there. A dead-letter queue nobody monitors is a silent data loss mechanism with extra steps.

Circuit breakers. One failing integration should not take the rest down with it. Isolate the blast radius before you need to.

Alerting that someone acts on. If your first signal of an integration failure is a customer complaint, you don't have monitoring. And if your team has learned to ignore the alerts, you have something worse than no monitoring, because you think you're covered.

Where you probably are

Lean team scaling fast. Small crew, limited integration experience, volume climbing. Use iPaaS and pre-built connectors, batch for what tolerates it, near-real-time for orders and inventory. Your risk is outgrowing the tooling with no migration plan and a pile of undocumented point-to-point connections nobody remembers building.

Complex B2B manufacturer or distributor. ERP is the centre of gravity for pricing, inventory, and orders. Customer-specific pricing, contract terms, tax exemptions. Use orchestrated middleware with real transformation capability, events for inventory and order status, batch for catalog and price lists. Your risk is duplicating ERP business logic in the integration layer, which means every pricing rule now lives in two places and they will drift.

Enterprise modernizing a legacy stack. Decades of accumulated integration logic and high technical debt. Strangler fig, meaning you replace the legacy integrations piece by piece while the old ones keep running, behind an API gateway that hides the legacy complexity from anything new. Your risk is rebuilding the legacy complexity inside the shiny new middleware and calling it modernization.

New channel alongside existing operations. A DTC site, a marketplace, or a B2B portal launching into an established ecosystem. Keep the integration layer light and focused on the flows the launch actually needs. Your risk is that MVP shortcuts become permanent architecture, because nothing creates less urgency than something that works.

The AI part, which is not a phase two

Product data that an answer engine can't read is invisible to a growing share of buyers. Not ranked badly. Invisible.

This changes what "good enough" means for integration architecture. Data that was fine when the only consumer was a storefront template is not fine when the consumer is a model deciding whether to mention you. Structured attributes, clean normalization, machine-readable metadata beyond the product page, and APIs that expose it rather than lock it inside a system that was never meant to publish.

The teams that treated this as a phase two are the ones retrofitting it now, at a worse price, under more pressure.

How we score it

We weight the seven dimensions against your situation. These are a starting point, and they move. An organization with a gnarly ERP landscape should push ownership higher. One with live AI initiatives should push data quality higher.

Dimension	Weight
Business process alignment	20%
System roles and data ownership	20%
Integration patterns and flow design	15%
Data quality and AI readiness	15%
Operational resilience	10%
Execution and team reality	10%
Growth horizon and scale	10%

Stop and re-plan if

- More than ten point-to-point connections with no middleware and no governance
- No named system of record for a domain two systems both write to
- Integration monitoring that depends on end users noticing
- ERP integration deferred to phase two while sitting in the middle of a core workflow
- Data cleanup assumed rather than scoped and staffed
- No defined failure behaviour for any integration
- An architecture designed for a team you don't have and aren't hiring

Where we come in

We assess your integration landscape against this framework, map system-of-record ownership across every data domain, and hand you a target-state architecture that's right-sized for the team you actually have. Then we help build it, starting with the flows that carry revenue.

Platform-agnostic, and opinionated about patterns. We've untangled enough integration layers to know which shortcuts cost the most later.

If your integration layer is the reason things are slow, that's a conversation worth having.

Ready to start?

info@commercestudio.ai

No pitch. Just a conversation about where you are and where you're trying to get to.

