

If you're in retail or enterprise e-commerce, especially on the technical or marketing side, you've probably felt the frustration of vendor sprawl. Your agency recommends adding "just one more" vendor for a supposedly seamless composable architecture — and suddenly you're juggling half a dozen providers, plus integration handoffs that look more like drop-offs.

As someone who has led multiple replatforms and headless rollouts, I get it. The promise of composability and modularity is alluring, but the reality? Unless you keep strict discipline, too many vendors can spiral costs and complicate ownership. Let's unpack why agencies push for these expanding stacks and how you can keep smart control over cost, scope, and long-term success.

What Agencies Mean by “Composable” and Why More Vendors?

The marketing gloss is simple: decouple your digital commerce experience into best-in-class modules — headless storefronts, API-driven integrations, flexible checkout, personalization engines, and more. This approach lets you swap or upgrade individual components without rebuilding the entire system.

Companies like **Netguru**, **DEPT**, and **Codal** are well-known proponents of composable commerce, often emphasizing API-first architecture and integrations that aim to reduce vendor lock-in.

But here's the rub: because each module does one thing well, you often end up needing more vendors, which the agency frames as flexibility. Each team or product owner can optimize independently. Sounds awesome, right?

Now imagine the integration effort. You're not just buying software, you're buying glue to connect them. Those API-driven integrations become complex contracts, coordination points, and potential points of failure.

Why More Vendors Can Hide Costs

Agencies tend to pitch the stack modularity as a way to control costs and fingerlakes1.com reduce risk. And yes, modular scope control can prevent the dreaded “big bang” rebuilds that blow timelines and budgets.

But the catch is how disciplined you really are about scope, ownership, and integration. Here's where most teams go sideways:



- **Hidden integration costs:** Mapping APIs, maintaining middleware, monitoring data flows, and troubleshooting failures—it all adds operational overhead.
- **“Too many cooks” mentality:** Vendor sprawl creates complex handoffs. One vendor’s “done” is another’s “start”. Without crystal-clear ownership, accountability scatters.
- **Long-term maintenance:** Initial delivery might be smooth, but by year two or three you’re juggling upgrades across multiple vendors, each with different roadmaps and support.

I keep a running list of these hidden costs, and it’s always longer than the initial budget suggests. So I ask every vendor meeting: *“Who owns this in year two?”* If the answer isn’t clear, red flags go up.

Clear System Boundaries and Replaceability Are Key

A frequent mistake is letting features or responsibilities bleed across vendors without clear boundaries. When scope isn’t modular at the system level, the cost of replacing one vendor with another skyrockets.



Good System BoundaryBad System Boundary Headless storefront strictly handles presentation and queries product data via APIs Storefront vendor also maintaining inventory sync and checkout workflows internally Checkout logic managed by a single, well-defined payment vendor with documented API contracts Distributed checkout across multiple vendors with interdependent APIs

By forcing clear boundaries like this, you reduce complexity. The goal is that if Vendor A suddenly increases prices or their roadmap shifts, you can replace them without project-wide rewrites.

The Myth of “We Can Do Anything” and Integration Handoffs

One of my biggest pet peeves is vague agency promises like “we can do anything” or “we integrate it all seamlessly.” Without clear operational process, these comments often mask the reality of multiple integration handoffs that become project bottlenecks.

When the agency suggests adding vendors for specialized components, ask these blunt questions:

1. What exact APIs or protocols does Vendor X expose for integration?
2. Who owns the coordination of those handoffs?
3. Have you built reusable middleware or connector patterns to avoid recreating effort?

Avoid vendor stacks that seem like patchworks of point solutions without unified architecture guidance. Companies like **Netguru** and **Codal** emphasize controlled evolution — using an API-first architecture that sets clear contract expectations for integrations and future change.

Modular Scope Discipline: Controlling Costs Long Term

The single biggest lever to resist vendor sprawl and integration complexity? Modular scope discipline.

Every additional vendor adds cost not just in license fees but in:

- Project management time coordinating deliverables
- Designing and maintaining API contracts
- Testing data consistency across systems

- User support for cross-system issues

That means rigorous prioritization in your rollout plan. You don't need the fanciest personalization engine day one, nor the newest headless CMS if your storefront vendor covers that well enough. Resisting scope creep protects budget and timeline.

Think in phases:

1. Identify core modules that deliver business impact — checkout, product catalog, user authentication
2. Pin down precise API contracts & ownership for each module
3. Roll out integrations with one or two key vendors, then stabilize
4. Evaluate gaps and add new vendors only when justified by ROI and maturity of your internal ownership model

DEPT's approach often includes emphasizing long-term ownership over one-off delivery — ensuring your internal team or vendor alliance fully supports operations after launch.

API-First Architecture Enables Controlled Evolution

Composable commerce only works if your APIs are high quality, well documented, and version-controlled. This keeps integrations stable and reduces reactive firefighting as vendors update features.

A solid API-first posture means:

- Clear data models and endpoints for each service
- Strict versioning allowing gradual migration off old contracts
- Monitoring and observability built into the integration layer
- Security baked into each interaction point

When your agency pitches another vendor, make sure the new module truly fits the API-first blueprint and isn't just another "black box" to bolt on.

What To Do If Your Agency Pushing Too Many Vendors

If you recognize "too many vendors" and feel buried under integration handoffs, here's a blunt checklist:

1. **Demand a vendor inventory:** List every vendor currently proposed or live, their responsibilities, and overlapping domains.
2. **Map integration handoffs:** Visualize where APIs and data move between vendors — highlight complexity hotspots.
3. **Clarify ownership:** Insist each integration point have a clear owner, and confirm the long-term support model.
4. **Push back on scope creep:** Say no to adding vendors without documented business case and modular boundary enforcement.
5. **Request a stack simplification plan:** Challenge the agency to identify where your stack can consolidate or use fewer vendors without losing benefits.

There's a balance between flexibility and chaos. Composable architectures led poorly become nightmares of vendor sprawl, with ballooning costs and fractured responsibility.

Conclusion: Own Your Stack, Limit Vendor Sprawl

Agencies like **Netguru**, **DEPT**, and **Codal** deliver great composable commerce experiences, but the real magic happens when you don't just accept every new vendor as gospel. Push for:

- Modular scope discipline — don't build more than you need
- Clear system boundaries — define replaceability and ownership up front
- API-first architecture — enforce contracts and versioning
- Long-term operational responsibility — who owns year two and beyond?

Too many vendors and integration handoffs won't magically make your platform better — they create hidden costs and operational complexity. Be the person who asks the hard questions, sets boundaries, and insists on simplicity where it counts. Your future self (and your finance team) will thank you.