You Asked

summarize https://medium.com/@codingplainenglish/why-teams-are-moving-back-from-microservices-to-modular-monoliths-in-2026-76a3eb7162b8?source=email-27d41029350d-1771117562666-digest.reader--76a3eb7162b8----2-98----------333a9158_526c_450f_abe9_e3da87b8fd3d-1

Gemini Fast

The article, "**Why Teams Are Moving Back From Microservices to Modular Monoliths in 2026,**" argues that the industry-wide trend of adopting microservices by default has reached a tipping point, leading many engineering teams to return to a more "pragmatic" architectural style: the **Modular Monolith**.

Here is a summary of the key themes and reasons for this shift:

1. The "Microservices Tax"

The article highlights that microservices don't eliminate complexity; they simply move it. In a monolith, complexity is in the **code** (easy to read and refactor). In microservices, complexity escapes into the **infrastructure**, resulting in:

- **Network Overhead:** Replacing simple method calls with network calls introduces latency and partial failure patterns.
- **Operational Burden:** Teams often spend more time maintaining Kubernetes, service meshes, and distributed tracing than shipping features.
- **Higher Costs:** Infrastructure costs for microservices can be 3x to 6x higher than monoliths due to the need for API gateways, load balancers, and dedicated platform engineers.

2. The Death of Developer Velocity

While microservices promise "independent deployments," many teams found the opposite to be true in practice:

- **Distributed Debugging:** Solving a single bug often requires chasing logs across dozens of services.
- **Coordinated Changes:** A simple feature change might require updating three different services, managing API versions, and deploying them in a specific order, which kills agility.

- **Local Development Pain:** Developers often can’t run the entire system on a single laptop anymore, making the "inner loop" of coding much slower.

3. What is a Modular Monolith?

The article defines the Modular Monolith as the "best of both worlds." It is **one deployable unit** (operationally simple like a monolith) but with **strict internal boundaries** (organized like microservices).

- Modules communicate via internal function calls rather than the network.
- It allows for atomic changes and easier refactoring.
- It scales better than its reputation suggests thanks to modern containerization.

4. High-Profile Precedents

The shift isn't just for startups; major tech companies are cited as examples of this "course correction":

- **Amazon Prime Video:** Switched a video quality monitoring service from microservices to a monolith, reducing infrastructure costs by 90%.
- **Shopify & Segment:** Both have publicly documented their moves toward modular monoliths to regain development speed and reduce "firefighting."

5. The Verdict for 2026

The article concludes that microservices are not "bad," but they should no longer be the **default**. They are a specialized tool for:

- Extremely large-scale organizations (hundreds of teams).
- Services with vastly different scaling requirements (e.g., a CPU-intensive ML service vs. a simple CRUD API).

For most mid-sized teams in 2026, the goal has shifted from "architectural purity" to **cognitive clarity**—building systems that are easy to understand, cheap to run, and fast to change.