5. Summary

Deployment & Structure

Aspect	Monolith	Layered Monolith	Modular Monolith	Microservices
Deployment Units	1	1	1	10-100+
Codebase	1 repo	1 repo	1 repo (or mono-repo)	Multiple repos
Build Time	5-30 min	5-30 min	5-30 min	1-5 min per service
Deployment Time	10-60 min	10-60 min	10-60 min	2-10 min per service
Deployment Complexity	Low	Low	Low	Very High
Deployment Independence	No	No	No	Yes

Development Experience

Aspect	Monolith	Layered Monolith	Modular Monolith	Microservices
Local Development	One command	One command	One command	Complex setup (Docker Compose)
Debugging	Easy (single process)	Easy	Easy	Hard (distributed tracing)
Testing	Simple	Simple	Simple	Complex (contract tests)
Learning Curve	Gentle	Gentle	Moderate	Steep
IDE Support	Excellent	Excellent	Excellent	Good but fragmented
Refactoring	Easy (IDE tools)	Easy	Easy	Difficult (cross-service)

Performance & Scalability

Aspect	Monolith	Layered Monolith	Modular Monolith	Microservices
Latency (internal calls)	< 1 ms (in-memory)	< 1 ms	< 1 ms	5-50 ms (network)
Throughput	High	High	High	Good
Scaling Strategy	Scale entire app	Scale entire app	Scale entire app	Scale services independently
Resource Efficiency	High	High	High	Lower (overhead)
Cold Start	Slow (large app)	Slow	Slow	Fast (small services)

Data Management

Aspect	Monolith	Layered Monolith	Modular Monolith	Microservices
Database Strategy	Shared DB	Shared DB	Shared DB (logical separation)	DB per service
Transactions	ACID	ACID	ACID	Eventual consistency
Data Consistency	Immediate	Immediate	Immediate	Eventual
Joins	SQL joins	SQL joins	SQL joins	Application-level
Data Duplication	None	None	Minimal	Common
Query Complexity	Simple	Simple	Simple	Complex

Team & Organization

Aspect	Monolith	Layered Monolith	Modular Monolith	Microservices
Optimal Team Size	1-20 devs	3-30 devs	5-50 devs	20-500+ devs
Team Structure	Single team	Single team or few teams	Multiple teams	Many independent teams
Coordination Needed	Low	Medium	Medium	Low (within team)
Ownership	Shared ownership	Feature ownership	Module ownership	Service ownership
Parallel Development	Limited	Possible	Good	Excellent
Onboarding Time	2-4 weeks	2-4 weeks	2-4 weeks	1-2 weeks (per service)

Operational Aspects

Aspect	Monolith	Layered Monolith	Modular Monolith	Microservices
Monitoring	Simple (1 app)	Simple	Simple	Complex (50+ services)
Logging	Single log file	Single log file	Single log file	Distributed (aggregation needed)
Alerting	Straightforward	Straightforward	Straightforward	Complex (many alerts)
Failure Isolation	None	None	None	Excellent
Disaster Recovery	Simple	Simple	Simple	Complex
Infrastructure Cost	Low	Low	Low	High
DevOps Expertise Required	Basic	Basic	Intermediate	Expert

Flexibility & Evolution

Aspect	Monolith	Layered Monolith	Modular Monolith	Microservices
Technology Diversity	Single stack	Single stack	Single stack	Multiple stacks
Framework Upgrade	Difficult (all at once)	Difficult	Difficult	Easy (per service)
Adding New Features	Fast	Fast	Fast	Slower (infra setup)
Experimentation	Hard	Hard	Moderate	Easy (isolated)
Technical Debt	Accumulates quickly	Accumulates	Manageable	Isolated to services
Migration to Microservices	Difficult	Difficult	Easier	N/A

Best For...

Monolith (Unstructured)

- Never recommended - leads to maintenance nightmare
- Only acceptable for throwaway prototypes

Layered Monolith

- Small to medium applications (< 100k lines of code)
- Teams of 5-20 developers
- Traditional CRUD applications
- Applications with clear technical layers
- When time-to-market is critical
- Limited DevOps resources

Modular Monolith

- Medium to large applications
- Teams of 10-50 developers
- Complex business domains
- When you want benefits of microservices without complexity
- Preparing for potential microservices migration
- Organizations valuing simplicity but needing structure
- Often the best choice for most applications

Microservices

- Very large scale (millions of users)
- Organizations with 20+ developers
- Multiple independent teams
- Need independent scaling of components
- Different parts need different technologies
- High availability requirements with failure isolation
- Mature DevOps culture and practices
- Only when complexity is justified by business needs

Anti-Patterns to Avoid

Microservices Too Early

"We're a 3-person startup building 15 microservices" Result: More time on infrastructure than product

Distributed Monolith

"We have microservices but can't deploy them independently" Result: Worst of both worlds - complexity + coupling

Big Ball of Mud Monolith

"Let's just write code without structure" Result: Unmaintainable mess that forces microservices migration

Nano-services

"Each function is a separate microservice" Result: Excessive network calls, deployment nightmare

Typical Evolution Path

```
Phase 1: Start Simple (Month 0–6)
   └→ Simple Layered Monolith

Phase 2: Add Structure (Month 6–18)
   └→ Modular Monolith

Phase 3: Extract When Needed (Month 18+)
   └→ Hybrid: Modular Monolith + 2–3 extracted services

Phase 4: Full Microservices (Only if necessary)
   └→ 10–20 microservices

Phase 5: Right-sizing (Ongoing)
   └→ Merge services that are too small
   └→ Split services that are too large
```

Key Insight

Architecture is not a destination, it's a journey.

The goal is not to achieve "perfect microservices" or "perfect monolith."

The goal is to build an architecture that:

- Matches your team's capabilities
- Solves your current problems
- Can evolve as requirements change
- Delivers value to users efficiently

Start simple. Add complexity only when you have concrete evidence you need it.