Simplicity-First Design Checklist

Use this before finalizing any architecture to prevent over-engineering

---

PART 1: COMPLEXITY JUSTIFICATION TEST

For each complex architectural decision, complete this sentence:

[ ] "I'm using [pattern/technology] specifically because [stated requirement] demands it"

[ ] "A simpler approach of [alternative] wouldn't work because [specific limitation]"

[ ] If you cannot complete both sentences, reconsider the complexity

PART 2: COMMON OVER-ENGINEERING PATTERNS

Check if you're falling into these traps:

[ ] Using microservices without clear service boundaries or scaling justification

[ ] Implementing database sharding for <1M records or <1K QPS

[ ] Adding distributed cache for <100 RPS

[ ] Using event sourcing without audit/replay requirement

[ ] Implementing CQRS without read/write scaling mismatch

[ ] Adding message queues for synchronous operations

[ ] Using multiple data centers without global distribution requirement

PART 3: SIMPLICITY-FIRST QUESTIONS

Ask yourself:

[ ] Could a monolithic application meet these requirements?

[ ] Could a single database with replicas handle this scale?

[ ] Is this complexity solving a stated problem or a hypothetical future problem?

[ ] Can I explain this architecture clearly in 3 minutes?

[ ] Would this design survive Occam's Razor?

PART 4: TIER VALIDATION

Based on your calculations:

[ ] <100 RPS + <1TB data → Start with Tier 1 (Simple)

[ ] 1K-10K RPS or multi-region → Tier 2 (Moderate)

[ ] >100K RPS or explicit complexity requirements → Tier 3 (High)

REMEMBER: You can always add complexity when requirements demand it. Starting complex and simplifying later is much harder in interviews (and in production).