ADR-1: Service Boundary Choice

1 CONTEXT

The DECODER API is designed as a minimal research-oriented slice to demonstrate software architecture thinking, basic analytics, and code quality. A key architectural decision is determining the service boundaries, whether to implement this as a monolithic service or as multiple microservices.

2 DECISION

I have chosen to implement the DECODER API as a monolithic Spring Boot application with clear internal boundaries (layered architecture) rather than multiple microservices.

3 RATIONALE

3.1 WHY MONOLITHIC?

3.1.1 Simplicity for Research Slice

- The goal is clarity and correctness, not complexity
- A monolithic service reduces operational overhead
- Easier to understand, test, and demonstrate architectural principles

3.1.2 Small Scope

- The requirements are well-defined and limited
- No need for independent scaling of components
- All components (ingest, analytics, forecasting) are tightly coupled

3.1.3 Single Technology Stack

- All components use Java/Spring Boot
- No need for polyglot programming at this stage
- Simpler deployment and development experience

3.1.4 Internal Boundaries

- Clear separation of concerns through layered architecture:
- Controller layer (API boundaries)
- Service layer (business logic)
- Repository layer (data access)
- This demonstrates architectural thinking without microservice complexity

4 CONSEQUENCES

4.1 POSITIVE

- Simpler Development: Single codebase, easier to navigate and understand
- Easier Testing: All components in one process, simpler integration testing
- Reduced Complexity: No service discovery, API gateways, or distributed tracing needed
- Faster Startup: Single application starts faster than multiple services
- Clear Boundaries: Layered architecture still demonstrates separation of concerns

4.2 NEGATIVE

- Scaling Limitations: Cannot independently scale forecast service vs ingest service
- Technology Lock-in: All components must use Java/Spring Boot
- Deployment Coupling: All components deploy together
- Single Point of Failure: If one component fails, entire service may be affected

5 ALTERNATIVES CONSIDERED

5.1 MICROSERVICES ARCHITECTURE

Rejected because:

- Too much complexity for a research slice
- Requires service discovery, API gateway, distributed tracing
- Operational overhead (container orchestration, monitoring)
- Not aligned with "clarity and correctness, not size or complexity" goal

5.2 MODULAR MONOLITH

Considered but not chosen:

- Similar benefits to monolith but with module boundaries
- Could be a future evolution if the service grows
- Current monolithic structure already demonstrates clear boundaries

5.3 FUTURE EVOLUTION

If the DECODER platform grows beyond a research slice:

- Short-term: Maintain monolithic structure with clear internal boundaries
- Medium-term: Consider module boundaries if codebase grows significantly
- Long-term: If independent scaling becomes necessary, consider extracting:
    - Forecasting service (if it needs more compute resources)
    - Analytics service (if queries become complex)
    - Authentication service (if RBAC becomes more complex)