Technology Decision Log - Executive Summary

Project: Forecast Service Microservice
Date: January 18, 2026
Prepared for: Volue SmartPulse Technical Interview

Core Technology Stack

Component	Technology	Key Rationale
Framework	.NET 10 / ASP.NET Core	High performance, cross-platform, async-first, enterprise-ready with excellent Azure integration
Language	C# with modern features	Strong typing, compile-time safety, pragmatic use of `field` keyword for UTC normalization in BaseEntity
Database	PostgreSQL 16	Open-source, ACID compliance, excellent performance, Azure Database for PostgreSQL available
ORM	Entity Framework Core 10.0	Code-first migrations, LINQ support, async operations, excellent Npgsql provider
Message Broker	RabbitMQ 3.13	Reliable event publishing, management UI, AMQP standard (Azure Service Bus recommended for production)
Containerization	Docker Compose	Environment consistency, easy setup, orchestration-ready (Azure Container Apps for production)
API Documentation	Scalar (OpenAPI)	Modern UI, superior to Swagger, .NET 10 native support
Logging	Serilog	Structured logging, multiple sinks (console + file), 30-day retention

---

Architectural Decisions

Clean Architecture (4 Layers)

- **Domain Layer:** Core entities, no external dependencies
- **Application Layer:** Business logic, services, DTOs, Result Pattern
- **Infrastructure Layer:** EF Core, repositories, RabbitMQ publisher
- **API Layer:** Controllers, middleware, request/response handling

Benefits: Separation of concerns, testability, maintainability, dependency inversion

Repository Pattern

Abstracts data access with interfaces in Domain layer and implementations in Infrastructure layer. Enables easy testing with mocks and technology-independent business logic.

Result Pattern (Functional Error Handling)

Type-safe error handling without exceptions. `Result<T>` with explicit success/failure paths improves performance and provides clear error propagation through layers.

Input Validation Strategy

ASP.NET Core Data Annotations with nullable types (`Guid?`, `DateTime?`, `decimal?`) and `[Required]` attributes. Custom `InvalidModelStateResponseFactory` ensures consistent error format across all validation failures.

Evolution: Initially explored `field` keyword with `ArgumentException` for validation, but Data Annotations provide better separation of concerns and integrate seamlessly with ASP.NET Core model binding.

Azure Cloud Considerations

Database

Azure Database for PostgreSQL - Flexible Server offers fully managed service with automatic backups, HA, and cost-efficient Burstable/General Purpose tiers. Lower licensing costs compared to Azure SQL.

Message Broker

Azure Service Bus (Standard tier) recommended for production over self-hosted RabbitMQ. Fully managed, enterprise features (dead letter queues, duplicate detection), better Azure integration (Managed Identity, Key Vault).

Compute

Azure Container Apps for cost efficiency with auto-scaling and pay-per-use pricing. Scales to zero when idle. Migrate to **AKS** only if complex orchestration needed.

Event-Driven Architecture

PositionChanged Event emitted on forecast create/update to RabbitMQ exchange (`forecast.events`) with routing key (`position.changed`). Enables downstream systems (trading, reporting, alerts) to react asynchronously to position changes.

Performance & Scalability

- **Async/Await:** All I/O operations non-blocking
- **EF Core Optimizations:** `AsNoTracking()` for read queries, selective `Include()`, connection pooling
- **Stateless Design:** Horizontally scalable, scoped DbContext per request
- **Docker Ready:** Multi-stage builds, health checks, persistent volumes

Pragmatic Engineering Principles

1. **Favor Framework Conventions:** Data Annotations over custom validation logic
2. **Avoid Premature Optimization:** I/O-bound microservice - database/network latency dominates
3. **Readability over Cleverness:** Limited C# advanced features (no `Span<T>`, no Extension Members)
4. **Enterprise Patterns:** Result Pattern, Repository Pattern, structured logging
5. **Cloud-Ready:** Docker containers, Azure service recommendations, cost-efficient architecture

---

Summary

Technology choices balance **pragmatism** (efficient development) with **professionalism** (enterprise architecture). The stack demonstrates:

- Clean Architecture for maintainability and testability
- Event-driven design for scalability and decoupling
- Modern .NET performance with cross-platform containerization
- Azure cloud readiness with managed service options
- Industry best practices for energy trading platforms

Result: Enterprise-grade microservice exceeding interview requirements while maintaining code clarity and operational simplicity.