# First Principles Architecture: Deephaven Data Platform

## 1. Problem Statement

We aim to rebuild our data analytics platform to support:  
- Enable Quants and Strats to use Deephaven directly for model development and strategy testing  
- Real-time, time-series data ingestion and processing  
- Structured, queryable, low-latency data for internal users and systems  
- Enhancing data with high-valued analytics  
- Common dataset sharing across teams and applications  
- Operational simplicity, observability, and maintainability

## 2. First Principles

|  |  |
| --- | --- |
| Principle | Rationale |
| Real-time, continuous data flow is required | We handle market data, logs, and metrics that arrive live |
| Deephaven is in-process and table-native | Processing should happen within Deephaven where possible |
| Stream-based ingestion is essential | Batch adds latency and fragility |
| Simplicity and transparency are key | Overengineering leads to tech debt and team confusion |
| Access must be fast and programmable | BI tools, internal apps, and engineers require direct access |

## 3. Simplest Viable Stack

|  |  |
| --- | --- |
| Layer | Tool/Tech |
| Ingestion | Kafka / Redpanda using Deephaven plugins |
| Transformation | Python + Deephaven table operations (joins, filters, logic) |
| Storage | In-memory or Parquet-backed tables |
| Serving | Deephaven UI, Python client, REST/WebSocket API |
| Monitoring | Deephaven logs + Prometheus/JMX as needed |

## 4. Core Questions to Drive Architecture

- What is the most direct data path from raw to insight?

- Are we duplicating transformation logic outside Deephaven?

- What parts of the pipeline need durability vs. can be ephemeral?

- Can each transformation be modeled as a table operation?

- Are we introducing tools or components that are not necessary?

## 5. Reference Architecture

Kafka → Deephaven Engine → In-Memory/Parquet → APIs/UI  
Transformation, joins, enrichment in-process

## 6. Action Plan for the Team

1. Define one real-time use case (e.g., market data table)

2. Implement ingestion → transformation → serving in Deephaven only

3. Measure latency, reliability, and complexity

4. Identify pain points and adjust from principles

5. Only add layers/tools if a fundamental constraint demands it

6. Define high-valued analytics and common enrichments to be applied across datasets

## 7. High-Valued and Common Analytics

To maximize the platform’s value and minimize duplicated work, we will establish shared analytics logic that can be applied across datasets. These serve as building blocks for both quantitative research and real-time strategy deployment.

### Example Analytics Enrichments

#### Security Master Join

- Normalize identifiers (e.g., symbol, CUSIP, ISIN)

- Attach metadata (e.g., sector, exchange, instrument type)

#### Corporate Actions Adjustment

- Price and volume adjustment for splits and dividends

- Maintain clean, backtestable time series

#### Rolling Time Series Features

- Moving average, volatility, z-score, exponential smoothing

- Applicable to prices, volumes, spreads, etc.

#### Market Microstructure Metrics

- Bid-ask spread: Difference between best bid and ask prices

- Order imbalance: Net buying or selling pressure based on volume at bid/ask

- Mid-price movement: Changes in the midpoint between bid and ask over time

- Trade aggressiveness: Classify trades as buyer/seller-initiated

- Quote-to-trade ratio: Frequency of quote updates relative to executed trades

- Book pressure: Depth imbalance at top N levels of the order book

- Volatility clustering: Micro-volatility measures over short intervals

- Tick-to-trade latency: Measure delay between quote and trade events

- Quote fade/quote stuffing detection: Identify anomalous liquidity patterns

- Useful for alpha generation, transaction cost analysis, and real-time execution strategy design

#### Event Annotations

- Tag earnings releases, macroeconomic announcements, news events

- Align temporal events to time-series data for downstream modeling

### Implementation Principles

- Implement within Deephaven as table-native transforms

- Ensure outputs are composable and reusable across teams

- Maintain documentation and example notebooks for usage