# **Building a Real-Time Financial Analytics Dashboard: A Comprehensive Roadmap (v2)**

## **I. Introduction**

**Purpose:** This report provides a comprehensive roadmap, technical guidance, and curated resources for constructing a sophisticated real-time financial analytics dashboard. The objective is to offer a step-by-step plan suitable for a developer undertaking this complex full-stack project.

**Project Vision:** The goal is to create a web application inspired by the functionality and user interface of platforms like TradingView. This dashboard will feature live, interactive stock charts utilizing the Lightweight Charts library, a dynamic list displaying Nifty 50 stocks with real-time price updates, and a dedicated section providing **comprehensive real-time analysis** for selected stocks, including:

1. **Sentiment Result:** (Very Bearish, Bearish, Neutral, Bullish, Very Bullish) based on news/social media.
2. **Short-term Analysis:** (Bearish, Bullish, Neutral) based on recent price trends.
3. **Volatility:** (Very Low, Low, Neutral, High, Very High) based on price fluctuations.
4. **Risk:** (Very Low, Low, Neutral, High, Very High) based on volatility/market correlation.
5. **Liquidity Crunch:** Indication of potential liquidity issues based on market depth.
6. **Overall Stock Mood Index:** (Very Low, Low, Neutral, High, Very High) a composite indicator.

These insights will be derived from processing data streams originating from financial news, social media, and **real-time market data feeds**.

**Technology Stack Overview:**

* **Frontend:** React.js (potentially with jQuery UI integration), SCSS for styling, Lightweight Charts library.
* **Backend:** Python with FastAPI, WebSockets for bidirectional communication.
* **Data Pipeline:** Apache Kafka, Apache Flink (Python API), TimescaleDB (PostgreSQL extension).
* **Data Sources:**
  + **Dhan API:** **(Assumed Paid Data Feed Subscription - within budget)** for live market data (Ticker, Quote, Depth) and historical data. Trading API remains free.
  + **Finnhub API (Free Tier):** For general financial news headlines.
  + **Reddit API (Free Tier):** For social media posts/sentiment gathering via PRAW.
  + **(Twitter/X API Removed):** Due to prohibitive costs for real-time data access.

This revised stack acknowledges the necessity of a paid broker data feed (Dhan) to enable the requested quantitative analysis (volatility, risk, liquidity, short-term trend). The budget constraint (up to Rs 2000/month) accommodates this, but likely precludes a paid real-time news feed.

**Roadmap Structure:** Phased approach:

Initiation -> Backend (Core + Real-time Market Data) -> Data Pipeline (Ingestion + Enhanced Processing) -> Frontend -> Integration, Testing, Deployment.

## **II. Phase 0: Project Initiation and Environment Setup (Estimated Time: 1-2 Weeks)**

*(Minor Updates)* Focuses on environment setup (Docker Compose) and API investigation.

**A. Development Environment Configuration:**

* **Docker Compose:** Setup remains crucial for Kafka, Zookeeper, Flink, TimescaleDB, Backend.
* **Frontend:** React (CRA), SCSS integration. Re-evaluate jQuery UI necessity vs. React alternatives.
* **Backend:** FastAPI setup with virtual environment.
* **Version Control:** Git and .gitignore.

**B. API Access and Investigation:**

* **Dhan API:**
  + **Crucial Change:** Confirm ability to subscribe to the **paid Data API feed** (~Rs 500/month + tax, potential waiver). This is **required** for the new analysis outcomes. Understand access to **Ticker, Quote, and Market Depth** WebSocket streams. Note the binary format. Verify rate limits. Obtain Access Token and Client ID.
* **Finnhub API:** Obtain a **free API key**. Understand limitations (general news, rate limits).
* **Reddit API:** Register a script app, obtain Client ID/Secret. Understand free tier rate limits (100 QPM w/ OAuth) and User-Agent requirements.
* **(Removed):** Twitter/X API investigation (deemed too costly for real-time reads).
* **Updated API Comparison:** Reflect the assumption of using the paid Dhan feed and free tiers for Finnhub/Reddit.

## **III. Phase 1: Backend Development - Core Setup & Stock List (Estimated Time: 1-2 Weeks)**

*(Largely Unchanged)* Builds foundational backend components.

**A. FastAPI Server Setup:**

* Project structure (routes, models, services, core, tests).
* Configuration management (.env file).
* Initial API routes (/health, /api/v1/stocks/nifty50-list).
* Pydantic models for validation.

**B. Dhan API Integration (Static Data):**

* Implement connectivity (using Access Token/Client ID). Consider DhanHQ-py library.
* Service function to fetch the static Nifty 50 list (symbols, names, security IDs) using appropriate Dhan REST endpoints.
* FastAPI endpoint (GET /api/v1/stocks/nifty50-list) to serve this list.

**C. Initial WebSocket Setup (Foundation):**

* Define backend WebSocket endpoint (/ws/stocks).
* Implement basic connection handshake (await websocket.accept()).
* Create ConnectionManager class for tracking active frontend connections.

## **IV. Phase 2: Backend Development - Real-time Market Data Streaming (Estimated Time: 3-4 Weeks)**

*(Significant Enhancement)* Focus shifts to handling **real-time market data** from Dhan.

**A. Dhan WebSocket Integration (Real-time Market Data):**

* **Connect:** Establish WebSocket connection to Dhan's Market Feed endpoint.
* **Authorize:** Implement binary authorization packet logic.
* **Subscribe:** Subscribe to **Ticker, Quote, and Market Depth** streams for Nifty 50 constituents. Manage subscription chunking (100 instruments/message).
* **Parse Binary Data:** **Crucial Step.** Implement robust parsing for different packet types (Ticker, Quote, Depth) based on Dhan v2 documentation. Extract LTP, timestamp, OHLCV, volume, total buy/sell quantity, bid/ask prices, and quantities from depth packets. Use Python's struct module.
* **Keep-Alive & Reconnection:** Handle Ping/Pong and implement robust reconnection logic.

**B. Backend Processing & Forwarding to Kafka:**

* **Data Transformation:** As binary market data arrives, parse it into structured Python objects/dictionaries.
* **Kafka Producer:** Integrate kafka-python within the backend service (or a dedicated producer microservice).
* **Publish to Kafka:**
  + Publish parsed **tick/quote data** (timestamp, symbol, ltp, volume, ohlc) to a new Kafka topic (e.g., stock\_market\_data).
  + Publish parsed **market depth data** (timestamp, symbol, bids list, asks list) to another new Kafka topic (e.g., market\_depth\_data).
* **Error Handling:** Implement error handling for Kafka production.

**C. Backend WebSocket Broadcasting (Basic Tickers):**

* **Connection Manager:** Use the existing manager.
* **Ticker Forwarding:** For simple UI updates, the backend can *also* directly broadcast basic **Ticker** updates (LTP, change) received from Dhan to connected frontend clients via /ws/stocks for the Nifty 50 list display. This avoids waiting for Flink processing for simple price updates.
* **Filtering:** Broadcast Nifty 50 ticker updates to all clients. Detailed quote/depth data is now primarily processed by Flink.

**D. API Endpoints for Enhanced Analysis:**

* **Define New Endpoint:** Create a new endpoint like @app.get("/api/analysis/{stock\_symbol}").
* **Placeholder:** Initially, return mock data for the 5 new outcomes or a "processing not ready" status. This endpoint will later query TimescaleDB (Phase 6).

## **V. Phase 3: Data Engineering Pipeline - Setup & Ingestion (Estimated Time: 2-3 Weeks)**

*(Updated Topics and Producers)* Sets up Kafka, TimescaleDB, and producers for *external* data (News, Reddit).

**A. Kafka Cluster Setup & Topic Creation:**

* Verify Kafka/Zookeeper from docker-compose.yml.
* Create Kafka topics using CLI tools within the Kafka container:
  + raw\_news (Input for Flink)
  + raw\_reddit\_posts (Input for Flink)
  + stock\_market\_data (Input for Flink, Output from Backend Phase 2)
  + market\_depth\_data (Input for Flink, Output from Backend Phase 2)
  + processed\_analysis (Output from Flink, potentially for backend consumption)

**B. Data Source Producers (News, Reddit):**

* **News Producer (Finnhub Free Tier):** Python script (news\_producer.py) fetches general news via Finnhub API, formats as JSON, publishes to raw\_news topic. Handles API key, rate limits.
* **Reddit Producer (PRAW Free Tier):** Python script (reddit\_producer.py) fetches posts/comments from specified subreddits via Reddit API, formats as JSON, publishes to raw\_reddit\_posts topic. Handles OAuth, User-Agent, rate limits.
* **(Removed):** Twitter producer.

**C. TimescaleDB/PostgreSQL Setup:**

* Verify TimescaleDB container.
* **Define Enhanced Schema:** Create/update schema.sql.
  + Keep stock\_sentiment table (populated by Flink).

Create a new table market\_analysis\_results to store the final 5 outcomes:  
 SQL  
CREATE TABLE market\_analysis\_results (

time TIMESTAMPTZ NOT NULL,

stock\_symbol VARCHAR(20) NOT NULL,

sentiment\_label VARCHAR(15), -- Very Bearish..Very Bullish

short\_term\_label VARCHAR(10), -- Bearish, Bullish, Neutral

volatility\_label VARCHAR(15), -- Very Low..Very High

risk\_label VARCHAR(15), -- Very Low..Very High

liquidity\_crunch BOOLEAN, -- True/False

mood\_index\_label VARCHAR(15), -- Very Low..Very High

-- Optional: Add underlying scores if needed

sentiment\_score DOUBLE PRECISION,

volatility\_score DOUBLE PRECISION,

risk\_score DOUBLE PRECISION,

mood\_index\_score DOUBLE PRECISION,

PRIMARY KEY (time, stock\_symbol)

);

-- Convert to Hypertable

SELECT create\_hypertable('market\_analysis\_results', 'time');

-- Add Index

CREATE INDEX idx\_analysis\_symbol\_time ON market\_analysis\_results (stock\_symbol, time DESC);

* + *(Optional):* Tables for raw stock\_market\_data and market\_depth\_data if persistence before Flink processing is desired.
* Apply schema and create hypertables.

## **VI. Phase 4: Data Engineering Pipeline - Real-time Enhanced Processing (Flink) (Estimated Time: 5-7 Weeks)**

*(Major Overhaul)* Build Flink job(s) to consume multiple Kafka topics, perform complex calculations, and sink results.

**A. Flink Job Setup:**

* **API:** Python DataStream/Table API recommended.
* **Connectors:** Configure Kafka Sources (stock\_market\_data, market\_depth\_data, raw\_news, raw\_reddit\_posts) and JDBC Sink (TimescaleDB). Ensure connector JARs are in Flink's /lib.
* **Checkpointing:** Enable robust checkpointing for fault tolerance.
* **Dependencies:** Ensure Flink environment has access to nltk, pandas, numpy, potentially ta-lib.

**B. Flink Processing Logic (Potentially Multiple Operators or Jobs):**

* **Input Streams:** Read data from the four input Kafka topics.
* **Keying:** Use keyBy(stock\_symbol) extensively to process data per stock.
* **Windowing:** Use time windows (tumbling, sliding) for aggregations and calculations.
* **Sentiment Calculation:**
  + Consume raw\_news, raw\_reddit\_posts.
  + Clean text, apply VADER sentiment analysis. Filter for relevance (e.g., symbol in headline).
  + Aggregate sentiment scores over a time window.
  + Map aggregated score to categories: Very Bearish, Bearish, Neutral, Bullish, Very Bullish.
* **Short-term Analysis Calculation:**
  + Consume stock\_market\_data (LTP, timestamp).
  + Calculate short-term moving averages or RSI over a rolling window.
  + Determine trend: Bullish, Bearish, Neutral.
* **Volatility Calculation:**
  + Consume stock\_market\_data.
  + Calculate the standard deviation of price returns over a rolling window.
  + Map the calculated volatility value to categories (Very Low to Very High).
* **Risk Calculation:**
  + Consume stock\_market\_data for the stock AND the Nifty 50 index (requires Nifty 50 data in the topic).
  + Calculate Beta over a rolling window.
  + Alternatively, use volatility as a proxy for risk.
  + Map score to categories (Very Low to Very High).
* **Liquidity Crunch Detection:**
  + Consume market\_depth\_data.
  + Analyze changes in bid-ask spread and order book depth changes over short time windows.
  + Define rules to flag a potential crunch. Output Boolean (True/False).
* **Overall Stock Mood Index Calculation:**
  + Combine outputs: Aggregated Sentiment Score, Volatility Score, Short-term Trend direction.
  + Define a weighting formula to create a composite Mood Index score.
  + Map the score to categories (Very Low to Very High).
* **State Management:** Flink's state backends will be crucial for storing intermediate values.

**C. Flink Sink:**

* Combine all calculated outcomes for each stock symbol per time window.
* Use the JDBC Sink to write these final results into the market\_analysis\_results table in TimescaleDB. Ensure idempotency or use transactional sinks if needed.
* *(Optional):* Sink results also to the processed\_analysis Kafka topic.

**D. Flink Deployment and Monitoring:**

* Package Python job and dependencies.
* Submit job to Flink cluster via CLI.
* Monitor via Flink Web UI (backpressure, checkpoints, task failures).

## **VII. Phase 5: Frontend Development - UI and Integration (Estimated Time: 3-4 Weeks)**

*(Updates to Sentiment Panel)* Build UI, integrate charts, connect to backend.

**A. UI Layout Implementation:** (Largely Unchanged)

* Component structure, SCSS styling, responsive layout.

**B. Lightweight Charts Integration:** (Largely Unchanged)

* Install library, create chart component (StockChart.js).
* Fetch historical data from backend.
* Handle **real-time Ticker updates** (LTP for chart's latest price) received directly from the backend WebSocket (/ws/stocks) using series.update().

**C. Nifty 50 List Implementation:** (Largely Unchanged)

* Component (Nifty50List.js), fetch initial list.
* Render list, handle selection.
* Update list items with **real-time Ticker updates** (LTP, change) received directly from the backend WebSocket (/ws/stocks).

**D. Enhanced Analysis Panel Implementation:**

* Component (AnalysisPanel.js - renamed from SentimentPanel.js).
* **Data Fetching:** When selected stock changes, fetch the **5 analysis outcomes** from the new backend endpoint (/api/analysis/{symbol}).
* **Rendering:** Display the 5 outcomes clearly.
* **Real-time Updates (Optional but Recommended):** Modify backend WebSocket to *also* push updates from the processed\_analysis Kafka topic (if implemented) to the frontend. The AnalysisPanel.js would listen for these specific messages and update its state.

**E. WebSocket Client Integration:** (Largely Unchanged for Basic Tickers)

* Establish connection to /ws/stocks.
* Handle incoming messages:
  + Nifty 50 Ticker updates: Update list state.
  + Selected Stock Ticker update: Update chart's latest price.
  + (New - Optional): Listen for analysis\_update messages and update the AnalysisPanel.js state.
* Use Context or state management library for distributing data.
* Handle disconnection/errors.

## **VIII. Phase 6: Integration, Testing, and Deployment (Estimated Time: 4-6 Weeks + Ongoing)**

*(Enhanced Scope)* Connect pieces, test thoroughly, deploy.

**A. Component Integration:**

* **Backend <-> TimescaleDB:** Implement database connection logic in the backend. Implement the GET /api/analysis/{symbol} endpoint to query the market\_analysis\_results table.
* **(Optional) Backend <-> Kafka (Processed Analysis):** Implement a Kafka consumer in the backend listening to processed\_analysis topic and push updates via WebSocket.
* **Frontend <-> Backend:** Connect frontend AnalysisPanel.js to fetch data from /api/analysis/{symbol}. Ensure all API calls and WebSocket message handling work correctly.

**B. Testing Methodologies:**

* **Unit Testing:** Continue testing individual components. Add tests for new calculation logic in Flink operators. Test the new backend endpoint's database query logic.
* **Integration Testing:**
  + Backend <-> TimescaleDB (Test /api/analysis endpoint).
  + Backend <-> Kafka (Test market data producer, optional analysis consumer).
  + Flink <-> Kafka <-> TimescaleDB (Test full pipeline flow).
  + Frontend <-> Backend (Test fetching analysis data, ticker updates).
* **End-to-End (E2E) Testing:** Enhance scenarios to include analysis updates. Simulating realistic market data streams is crucial. Requires robust async testing strategies. Use tools like Cypress/Playwright.

**C. Deployment Strategies:** (Largely Unchanged)

* Containerize services (Frontend, Backend, Producers, Flink).
* Orchestrate using Docker Compose (local/simple) or Kubernetes (scalable/cloud).
* Consider Cloud options (Managed Kafka/Flink/TimescaleDB vs. self-hosted).
* Implement CI/CD pipelines.

## **IX. Learning Resources and Upskilling**

*(Additions)*

* **Real-time Data Pipelines:** As before (Kafka, Flink docs/tutorials).
* **Backend Development:** As before (FastAPI/Spring docs/tutorials). Add resources for database interaction (SQLAlchemy, asyncpg, Spring Data JPA).
* **WebSockets Integration:** As before.
* **Sentiment Analysis & NLP:** As before (NLTK, TextBlob, etc.).
* **Stock Market Regime/Quantitative Analysis:** **(Increased Importance)**
  + **Technical Analysis Libraries:** TA-Lib (requires careful installation), pandas-ta.
  + **Statistical Libraries:** numpy, scipy.stats, statsmodels.
  + **Financial Time Series Analysis:** Books and online courses covering Moving Averages, Volatility calculation (Standard Deviation, ATR), Beta calculation.
  + **Flink Stateful Processing:** Deep dive into Flink's documentation on state management, windowing, ProcessFunction, and AggregateFunction.

## **X. Alternative Technologies and Data Sources**

*(Largely Unchanged, but context shifts)*

* **Real-time Stock Data APIs:** Dhan/Zerodha are now primary candidates due to budget. Alternatives (Alpaca, Polygon, IBKR) remain options.
* **News/Sentiment Data Sources:** Finnhub (free) and Reddit (free) remain primary. Paid news APIs are likely out of budget. StockTwits API worth investigating. RSS feeds still an option.
* **Open Financial Datasets:** Still relevant for backtesting/historical context (Nasdaq Data Link, yfinance, EDGAR, Kaggle).

## **XI. Open Source References and Further Reading**

*(Largely Unchanged)* Search GitHub/blogs for relevant examples. Add searches for "flink time series analysis", "flink technical indicators". Prioritize official docs and reputable sources.

## **XII. Conclusion**

**Summary:** The project scope has expanded significantly towards a comprehensive analysis dashboard. The roadmap now incorporates using the **paid Dhan data feed** for essential real-time market data (price, volume, depth), enabling the calculation of short-term trends, volatility, risk, and liquidity indicators, alongside refined sentiment analysis from free news/social media sources.

**Key Challenges (Updated):**

1. **Dhan API Integration & Cost:** Reliant on obtaining and potentially paying for the Dhan data feed. Robust handling of the binary WebSocket protocol is critical.
2. **Flink Complexity:** Implementing the quantitative calculations requires significant Flink expertise (state, windowing) AND financial domain knowledge.
3. **Real-time System Complexity:** Managing multiple data streams and stateful processing adds complexity.
4. **Testing:** E2E testing becomes more challenging, requiring simulation of realistic market data and validation across the stack.

**Next Steps:**

1. **Confirm Dhan API Access:** Ensure you can get keys and subscribe to the required real-time data feeds.
2. **Proceed Incrementally:** Focus on integrating Dhan market data into Kafka first (Revised Phase 2 & 3). Then, tackle Flink processing logic incrementally.
3. **Refine Definitions:** Clearly define calculation methods and thresholds for the 5 analysis outcomes before implementing in Flink.

**Final Encouragement:** While more demanding, the revised scope leads to a much more powerful dashboard. The budget increase makes it feasible. Careful planning, incremental implementation, and focused learning are key.