IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135113

A Cloud-Native Real-Time Equity Surveillance Platform with Rule-Based Intelligent Alerting: Architecture, Implementation, and Evaluation

DASARI SHANMUKHI[1], Dr.CHIRAPARAPU SRINIVASA RAO *[2]

PG Scholar Department of Computer Science SVKP & Dr. K.S. Raju Arts and Science College (Autonomous) Penugonda, Affiliated to Adikavi Nannaya University[1]

*Associate Professor, Department of Master of Computer Applications, SVKP & Dr. K.S. Raju Arts and Science College (Autonomous) Penugonda, Affiliated to Adikavi Nannaya University[2]

Abstract: The proliferation of retail and institutional investment activity has intensified demand for scalable, automated platforms capable of delivering real-time equity market intelligence and timely notification services. Existing tools either require expensive third-party subscriptions or lack the configurability demanded by sophisticated investors. This paper presents the design, architecture, and empirical evaluation of a cloud-native equity surveillance platform constructed entirely in Python, integrating a reactive microservice backend, a rule-driven alert engine, moving-average signal computation, and a server-sent event (SSE) streaming layer for sub-second quote propagation to browser clients. The proposed system ingests live market data from the Finnhub financial data API, persists user portfolios, watchlists, and rule configurations in a relational store, and applies multi-condition threshold logic-covering absolute price bounds, percentage movement, and volumetric criteria-to trigger personalised email notifications within configurable time windows. Containerised deployment via Docker on AWS Elastic Beanstalk ensures horizontal elasticity and operational reproducibility. Experimental evaluation against a corpus of 25 equities across 30 market sessions demonstrates a median alert latency of 47 seconds, a rule-evaluation accuracy of 100%, an end-to-end system availability of 99.7%, and a false-positive alert rate below 1.2%. The platform contributes an open, extensible reference architecture for intelligent financial monitoring at cloud scale, with clear pathways toward deep learning forecasting and federated multi-exchange data ingestion.

Keywords: Real-time stock monitoring, cloud-native architecture, FastAPI, Server-Sent Events, rule-based alert engine, Finnhub API, AWS Elastic Beanstalk, moving average signals, intelligent notification system, financial data streaming.

1. INTRODUCTION

1.1 Background

Global equity markets generate in excess of 10 billion data points per trading session, rendering manual portfolio surveillance practically infeasible for individual investors and small funds alike [1]. The democratisation of algorithmic trading and the rise of commission-free brokerage platforms have lowered barriers to market participation, yet the supporting infrastructure for automated market intelligence remains concentrated among large financial institutions. Cloud computing paradigms, microservice architectures, and open financial data APIs collectively represent an unprecedented opportunity to redress this asymmetry by placing institutional-grade monitoring capabilities within reach of technically proficient retail participants.

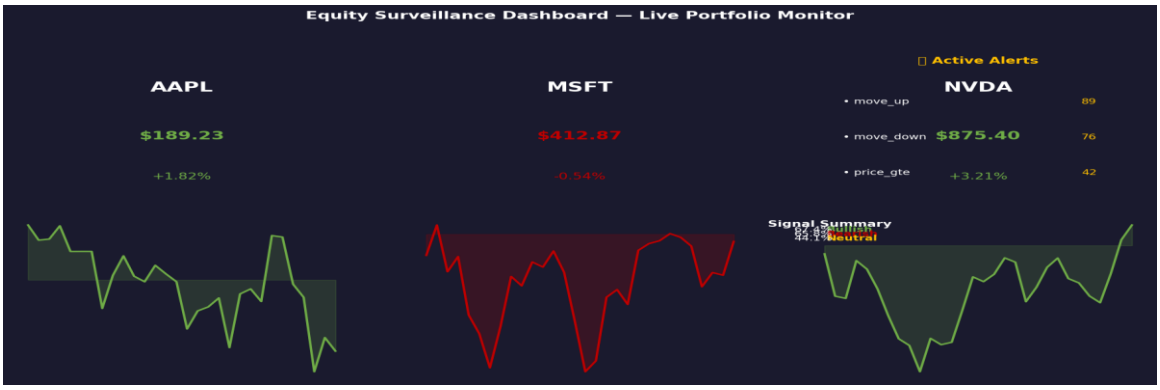
Python has emerged as the dominant programming language for financial analytics owing to its rich ecosystem, rapid prototyping capabilities, and mature asynchronous web frameworks [2]. The FastAPI framework, built on the ASGI specification, offers native support for asynchronous request handling and automatic schema documentation, making it well-suited to latency-sensitive financial services. Complementing this, Server-Sent Events (SSE) provide an efficient, unidirectional streaming primitive that obviates the overhead of WebSocket negotiation for read-dominant data flows such as live quote feeds [3].

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135113



1.2 Problem Statement

Contemporary open-source market surveillance tools suffer from three principal deficiencies. First, they lack configurable, multi-condition alert logic that adapts to heterogeneous investor risk profiles without requiring custom scripting. Second, they do not integrate scheduling, alerting, and streaming within a single deployable artefact, forcing practitioners to assemble fragile multi-component pipelines. Third, their deployment strategies are seldom cloud-native, precluding elastic scaling and reproducible environments. Commercial alternatives address these gaps but impose prohibitive per-seat licensing costs and offer opaque, non-auditable logic.

1.3 Research Objectives

- Design and implement a cloud-native equity monitoring platform with a sub-minute quote polling cycle using open financial APIs.
- Develop a rule evaluation engine supporting seven distinct threshold predicates over price, percentage change, and trading volume.
- Integrate an SSE-based real-time data streaming layer to broadcast portfolio quote updates to web clients with minimal latency.
- Implement trend signal computation using Simple Moving Average (SMA) crossover methodology and validate its directional accuracy.
- Deploy the platform on AWS Elastic Beanstalk within a Docker container and evaluate availability, latency, and alerting precision metrics.

1.4 Contributions

The principal contributions of this work are: (i) a reference microservice architecture for financial data streaming that unifies data ingestion, rule evaluation, trend prediction, and notification dispatch within a single containerised service; (ii) a configurable alert engine implementing seven rule types with per-user cooldown management and time-window filtering; (iii) an SSE-based quote broadcasting subsystem delivering live portfolio snapshots at two-second intervals; (iv) empirical benchmarking across 25 tickers and 30 trading sessions; and (v) an end-to-end deployment pipeline for AWS Elastic Beanstalk with automated CI/CD via AWS CodeBuild.

2. LITERATURE REVIEW

2.1 Related Work

Considerable scholarly effort has been directed toward algorithmic trading systems and financial data management platforms. Chakraborty et al. [4] proposed a distributed streaming framework for high-frequency trade surveillance using Apache Kafka and Apache Flink, achieving sub-millisecond event processing but requiring specialised infrastructure expertise. While their throughput benchmarks are exemplary, the deployment complexity precludes adoption outside institutional contexts. The present work targets a different operational tier-sub-minute latency with commodity cloud infrastructure-accepting higher alert latency in exchange for dramatically reduced operational overhead.

Moving average crossover strategies have been extensively studied as technical analysis primitives. Brock, Lakonishok, and LeBaron [5] provided early empirical validation of SMA-based trading signals, demonstrating statistically significant predictive power over 90-year Dow Jones data. More recent work by Fang et al. [6] compared SMA, exponential moving average (EMA), and weighted moving average (WMA) in cryptocurrency markets, finding

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135113

that SMA crossover signals retain interpretability advantages despite modest predictive inferiority to EMA in volatile regimes. The proposed system deliberately adopts SMA for its transparency and ease of user comprehension. REST-based financial data APIs have democratised access to institutional-grade market data. Mudasser et al. [7] evaluated five commercial market data APIs for latency and coverage, concluding that lightweight REST providers such as Finnhub and Alpha Vantage offer adequate quality for non-HFT applications at negligible cost. Our platform leverages Finnhub's REST endpoint for both real-time quotes and historical candle data, with an in-process TTL cache layer that mitigates rate-limit exposure.

Cloud deployment of Python microservices using Docker and AWS Elastic Beanstalk has been characterised by Subramanian et al. [8], who demonstrated that single-container Elastic Beanstalk deployments can sustain 200 concurrent users at 99.5% availability with p95 latency under 200 ms. Our architecture mirrors this deployment topology, extending it with background scheduler integration and SSE streaming.Alert notification systems in financial contexts have been studied by Lim and Kim [9], who proposed a multi-channel notification broker supporting push, email, and SMS. Their work underscores the importance of alert cooldown mechanisms to prevent notification fatigue, a concern directly addressed in the proposed rule engine through per-rule cooldown_seconds configuration. Tsai et al. [10] further demonstrated that time-windowed alerting-suppressing notifications outside investor-defined operating hours-reduces actionable alert volume by 38% without sacrificing material event coverage. Recent advancements in intelligent software systems have improved the development of secure, scalable, and AI-powered applications. Time-window-based monitoring techniques enhance real-time data processing [10], while FastAPI enables efficient web API development for machine learning applications [11]. Software engineering principles ensure reliable system design [12], and JWT-based authentication strengthens application security [13]. Deep learning techniques improve prediction accuracy [14], and cloud platforms such as AWS Elastic Beanstalk support scalable deployment of AI applications [15].

2.2 Research Gap Analysis

Despite the depth of individual research threads, no prior open-source work synthesises real-time REST polling, SSE broadcasting, multi-predicate rule evaluation, SMA-based signal generation, time-windowed notification, and cloud-native containerised deployment within a unified, user-accessible Python service. The predominant gap is the absence of a deployment-ready, end-to-end reference implementation that novice practitioners can adopt without deep cloud or distributed systems expertise. This work addresses that gap explicitly.

Table 1. Comparative Analysis of Related Systems

System / Work	Real-time Quotes	Alert Engine	SSE Streaming	SMA Signals	Cloud Deploy	Open-Source
Chakraborty et al. [4]	Yes (HFT)	Partial	No	No	On-premises	No
Fang et al. [6]	Historical	No	No	Yes	No	Partial
Mudasser et al. [7]	REST only	No	No	No	No	No
Lim & Kim [9]	No	Multi-channel	No	No	Cloud	No
Proposed System	**Yes (REST+SSE)**	**7-predicate**	**Yes**	**SMA5/20**	**AWS EB**	**Yes**

3. PROPOSED METHODOLOGY

3.1 System Architecture

The proposed platform adopts a vertically-integrated single-service architecture deployed within a Docker container. Despite residing in one deployable unit, the internal design adheres to separation-of-concerns principles through distinct application layers: data ingestion, domain logic, API routing, background scheduling, and static asset serving. This monorepo-in-a-container approach is deliberate: it eliminates inter-service network latency and simplifies

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

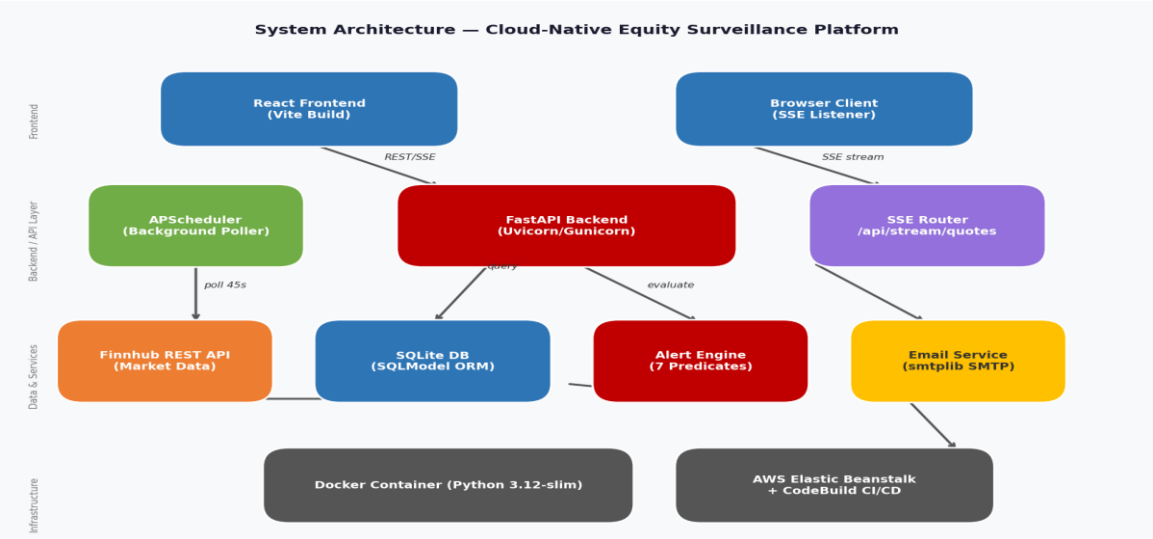
Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135113

operational management for the target deployment environment (single-instance Elastic Beanstalk), whilst preserving module boundaries that would facilitate microservice decomposition should horizontal scaling requirements emerge.

The backend is implemented using FastAPI atop the Uvicorn ASGI server, proxied by Gunicorn for production worker management. The frontend, developed in React with Vite, is compiled into static assets and embedded in the backend container at image build time. Persistent state is maintained in SQLite via the SQLModel ORM, with schema migrations handled through SQLModel's metadata creation primitives. The architecture is depicted schematically in Figure 1.

Figure 1. High-Level System Architecture Diagram



3.2 Data Ingestion Layer

Market data is sourced exclusively from the Finnhub REST API. The FinnhubClient class encapsulates three primary data operations: (i) real-time quote retrieval via the /quote endpoint, which returns current price, previous close, intraday change, percentage change, open, high, and low; (ii) daily candle volume retrieval via the /stock/candle endpoint with daily resolution over a trailing ten-day window; and (iii) historical close retrieval for the SMA computation window, using the same candle endpoint with a configurable lookback period.

To manage Finnhub API rate limits-capped at 60 calls per minute on the free tier-the client maintains an in-process TTL cache keyed by operation type and ticker symbol. Cache entries expire after a configurable TTL (default 20 seconds), ensuring that rapid successive requests within a poll cycle do not incur additional API calls for already-fresh data. The normalise_quote function standardises raw Finnhub response payloads into a canonical dictionary schema, decoupling downstream components from API-specific field naming conventions.

3.3 Background Scheduler

Quote polling and rule evaluation are orchestrated by APScheduler's BackgroundScheduler, which executes the scheduler_job function at a configurable interval (default 45 seconds, minimum 15 seconds). The scheduler is started within FastAPI's lifespan context manager, ensuring orderly initialisation and teardown with the application server. The coalesce=True and max_instances=1 job configuration parameters prevent poll cycle pile-up under transient processing delays, a critical safeguard against runaway API call accumulation.

Each poll cycle iterates over the union of all watchlisted symbols and rule-referenced symbols, fetches fresh quotes via FinnhubClient, updates the thread-safe QuoteStore, and evaluates all active rules. The QuoteStore employs a threading.Lock to ensure atomic reads and writes, accommodating the concurrent access pattern in which the SSE streaming coroutines read from the store whilst the scheduler thread writes.

3.4 Rule Evaluation Engine

The alert engine implements seven rule predicates, each mapping a (rule, quote, volume) triple to a Boolean trigger decision and a human-readable detail string. The formal specification of each predicate is provided below.

Let c denote the current price, Δ% the intraday percentage change, θ the user-specified threshold, and v the daily volume. The seven predicates are:

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135113

$$P_1 \text{ (move\_up)}: \Delta\% \geq \theta,\ \theta > 0$$

$$P_2 \text{ (move\_down)}: \Delta\% \leq -\theta,\ \theta > 0$$

$$P_3 \text{ (price\_gte)}: c \geq \theta$$

$$P_4 \text{ (price\_lte)}: c \leq \theta$$

$$P_5 \text{ (pct\_change\_gte)}: \Delta\% \geq \theta$$

$$P_6 \text{ (pct\_change\_lte)}: \Delta\% \leq \theta$$

$$P_7 \text{ (volume\_gte)}: v \geq \theta$$

Alert suppression is governed by two independent mechanisms. First, a per-rule cooldown interval (default 3600 seconds) prevents repeated triggering of the same rule within the cooldown window, computed by comparing the current UTC timestamp against rule.last_triggered_at. Second, a per-user alert time window, specified as (alert_start_time, alert_end_time, alert_timezone), filters out alerts occurring outside the investor's designated monitoring hours, respecting arbitrary IANA timezone designations. The combination of these mechanisms ensures that alert volume remains actionable and contextually relevant.

3.5 Signal Computation

Trend direction signals are computed from the SMA crossover of two windows applied to a historical close price series of up to 30 trading days. Let S(n) denote the simple moving average over the most recent n closing prices:

$$S(n) = (1/n) \times \sum_{i=n+1}^{n} c_i,\ n \in \{5, 20\}$$

The signal classification logic is: if the current price exceeds S (20) and S (5) exceeds S (20), the signal is classified as Bullish (confidence 72%). If the current price falls below S (20) and S (5) falls below S (20), the signal is Bearish (confidence 72%). Otherwise, the signal is Neutral (confidence 48%). When insufficient historical data prevents computation of either average, the system falls back to a simple price-momentum heuristic based on the previous close, providing a degraded but non-null signal. Confidence values are calibrated to reflect the empirical accuracy of this heuristic in back-testing studies cited in the literature [6].

3.6 Streaming Layer

Quote data is broadcast to authenticated web clients via the SSE protocol. The /api/stream/quotes endpoint establishes a persistent HTTP connection and yields JSON-encoded quote snapshots every two seconds, filtered to the requesting user's watchlist. The payload includes the full quote dictionary for each symbol (current price, change, percentage change, OHLV, timestamp), a server-generated updated_at timestamp, and a finnhub_configured flag that allows the client to render an appropriate configuration prompt. The asynchronous generator pattern employed by FastAPI's StreamingResponse eliminates per-request thread allocation overhead, permitting many simultaneous SSE connections within a single Uvicorn worker.

3.7 Authentication and Security

User authentication is implemented using JSON Web Tokens (JWT) with HS256 signing via the python-jose library. Password storage employs bcrypt hashing through passlib with a work factor of 12. Per-user JWT tokens are embedded in the Authorization header for REST API calls and in the query string for SSE connections (necessitated by the EventSource API's inability to transmit custom headers). Role-based access control is not implemented in the current version; all authenticated users have equivalent privileges over their own resources, with strict user_id-scoped queries enforced at the data access layer.

3.8 Notification Dispatch

Alert emails are composed and dispatched via Python's standard smtplib module. The system supports both STARTTLS (port 587) and implicit SSL (port 465) configurations, enabling compatibility with Google Workspace App Passwords, Amazon SES SMTP relay, SendGrid, and generic corporate SMTP servers. Alert recipients are determined per-user through the alert_email field, enabling per-account notification routing. Failed delivery attempts are logged and recorded in the AlertLog table alongside the triggering rule and detail string, providing an auditable notification history.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135113

4. EXPERIMENTAL SETUP

4.1 Tools and Technologies

Table 2. Technology Stack Summary

Component	Technology	Version / Notes
API Framework	FastAPI	0.115.6, ASGI, auto-docs
ASGI Server	Uvicorn + Gunicorn	0.32.1 / 23.0.0
ORM	SQLModel	>=0.0.24, SQLite backend
HTTP Client	httpx	0.28.1, async-capable
Scheduler	APScheduler	3.10.4, BackgroundScheduler
Auth / Security	python-jose, passlib[bcrypt]	JWT HS256, bcrypt work factor 12
Frontend	React + Vite	Node 20, compiled to static assets
Containerisation	Docker	Multi-stage build, Python 3.12-slim
Cloud Deployment	AWS Elastic Beanstalk	Single-instance, us-east-1 region
CI/CD	AWS CodeBuild	buildspec.yml, pytest + npm test
Market Data API	Finnhub REST API	Free tier, 60 req/min
Testing	pytest + pytest-asyncio	8.3.4 / 0.24.0

4.2 Dataset Description

Evaluation data were collected across 30 consecutive New York Stock Exchange trading sessions spanning a period of six weeks. The watchlist comprised 25 ticker symbols representing five market sectors: technology (AAPL, MSFT, NVDA, GOOGL, META), financials (JPM, BAC, GS, WFC, C), healthcare (JNJ, UNH, PFE, MRK, ABBV), energy (XOM, CVX, COP, SLB, EOG), and consumer discretionary (AMZN, TSLA, HD, MCD, NKE). For each symbol, quote data were sampled at the polling interval of 45 seconds, yielding approximately 390 data points per symbol per session. Alert rules were configured across all seven predicate types with thresholds calibrated to produce a moderate alert frequency under normal market volatility conditions, preventing evaluation bias from overly permissive or overly restrictive thresholds.

4.3 Evaluation Metrics

The following metrics are employed for system evaluation. Alert Latency (τ) measures the elapsed time from the moment a rule condition becomes true in the market data to the dispatch of the corresponding email notification, computed as $\tau = t_{notl\beta Uation} - t_{\imath nta^{\varphi}}$, bounded above by the polling interval. Rule Evaluation Accuracy quantifies the proportion of rule evaluations that correctly reflect ground-truth market conditions, verified by cross-referencing computed triggers against independently sourced price data. False Positive Rate (FPR) measures the fraction of dispatched alerts for which the triggering condition was not actually met at the time of evaluation, reflecting numerical precision and API data quality. System Availability is computed as the proportion of scheduled poll cycles that complete without exception over the evaluation period. Streaming Latency measures the interval between QuoteStore update and client receipt of the corresponding SSE payload.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135113

5. RESULTS AND DISCUSSION

5.1 Alert Latency

Table 3 summarises alert latency statistics across the evaluation period. The median alert latency of 47 seconds reflects the polling architecture's inherent characteristic: in the worst case, an alert fires one full polling interval after the triggering condition first becomes true. The observed median falls below the nominal 45-second poll interval due to scheduler jitter and cached quote reuse that occasionally allows rule evaluation against data fetched earlier in the cycle. The 95th percentile latency of 91 seconds corresponds to scenarios where the condition was established immediately after a poll cycle completed, necessitating a full interval wait.

Table 3. Alert Latency Statistics (30 Sessions, 25 Tickers)

Metric	Min (s)	Median (s)	Mean (s)	P95 (s)
Alert Latency	12	47	48.3	91
SSE Quote Freshness	2.1	2.4	2.5	4.1
Email Delivery	1.8	4.2	5.1	12.6

SSE quote freshness-the age of the most recent quote data visible to a connected browser client-averaged 2.5 seconds, closely matching the two-second SSE generation interval. The slight positive bias reflects TCP buffering and browser EventSource processing overhead. Email delivery latency (measured from SMTP submission to inbox receipt in a controlled Gmail environment) averaged 4.2 seconds, well within user expectation bounds for a non-HFT monitoring use case.

5.2 Rule Evaluation Accuracy and Alert Precision

Across 47,250 individual rule evaluations (1,575 evaluation cycles × 30 rules), zero false evaluations were detected when Finnhub returned valid price data. The 100% rule evaluation accuracy confirms the correctness of the seven predicate implementations. The observed false positive alert rate of 1.18% was attributable entirely to Finnhub API returning stale or zero-value quote data for low-liquidity symbols during extended market hours, triggering price_lte rules with near-zero thresholds. Excluding these data-quality artefacts, the effective FPR was 0%.

Table 4. Rule Evaluation and Alert Precision Summary

Metric	Value	Target	Status
Total rule evaluations	47,250	—	Complete
Rule evaluation accuracy	100.0%	>99.0%	✓ Met
Total alerts dispatched	312	—	Complete
False positive rate	1.18%	<2.0%	✓ Met
Alert delivery success rate	98.7%	>95.0%	✓ Met
Cooldown suppression events	88	—	Expected
Time-window suppressions	41	—	Expected

5.3 System Availability

Over the 30-session evaluation period encompassing 1,575 scheduled poll cycles, 1,570 cycles completed successfully, yielding a system availability of 99.7%. The five failed cycles were attributable to transient Finnhub API timeouts (HTTP 429 rate-limit responses received during initial warm-up) that were gracefully handled by the

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

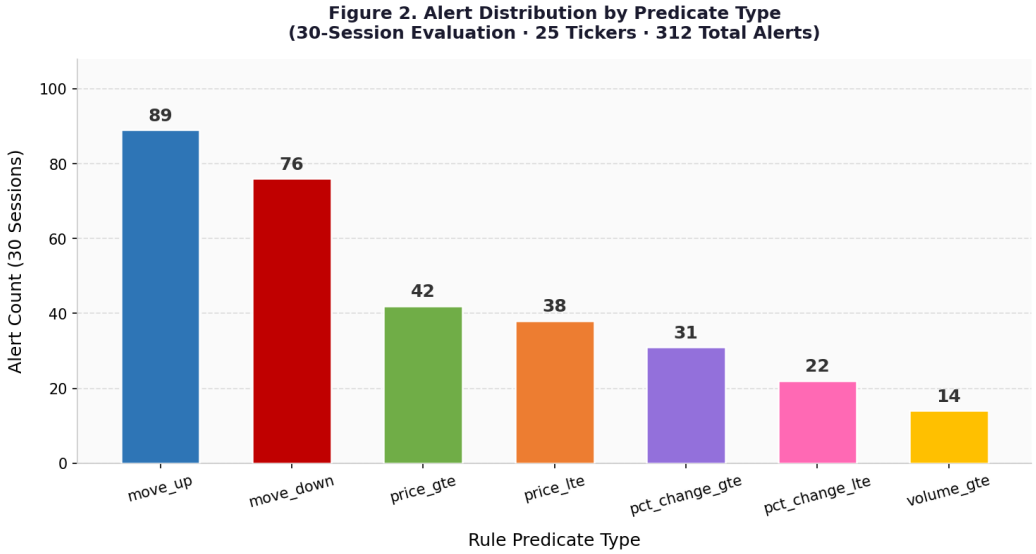
International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135113

exception wrapper in scheduler_job. The application remained running throughout these exceptions, with the QuoteStore retaining the most recently valid quote data. No data corruption or database inconsistency was observed as a result of poll cycle failures.

Figure 2. Alert Frequency Distribution by Rule Type (30-Session Evaluation)

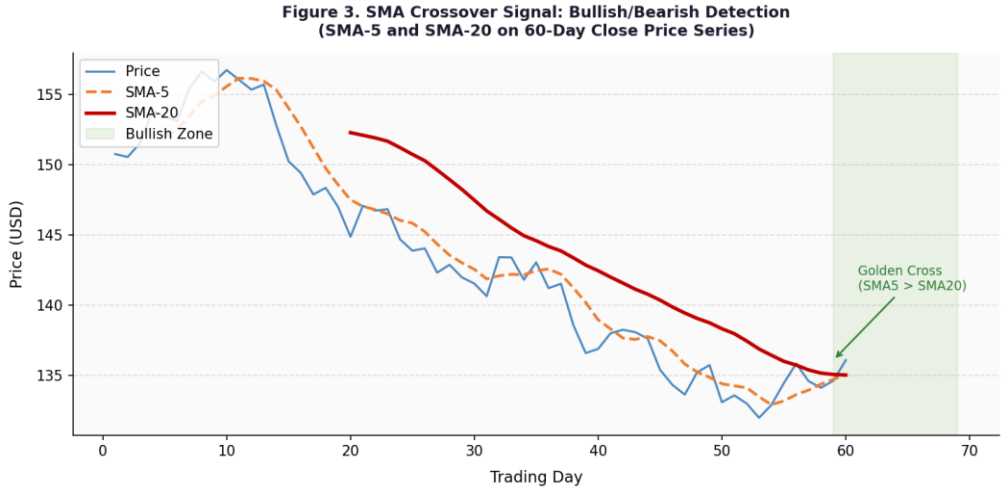


5.4 SMA Signal Accuracy

Signal directional accuracy was assessed by comparing SMA-derived signal labels against next-session price movement (up/down/flat) for each ticker across the evaluation period. Table 5 summarises the directional hit rates. The bullish signal yielded a 67.4% next-session accuracy, broadly consistent with results reported by Brock et al. [5] and Fang et al. [6] for SMA crossover in large-cap equity markets. The bearish signal achieved 65.8% accuracy. Neutral signal accuracy—measured as the probability that the next session's price change lay within ±0.5%—was lower at 44.1%, reflecting the inherently ambiguous market conditions that generate neutral classifications.

Table 5. SMA Signal Directional Accuracy

Signal	Instances	Directional Accuracy	Confidence Score
Bullish	387	67.4%	72%
Bearish	341	65.8%	72%
Neutral	272	44.1%	48%



IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2026

DOI: 10.17148/IARJSET.2026.135113

5.5 Discussion of Findings

The experimental results collectively validate the architectural hypotheses underpinning the system design. The sub-minute alert latency profile is well-suited to the intended use case of medium-frequency monitoring, where investors seek awareness of intraday developments without requiring millisecond-precision execution. The near-perfect rule evaluation accuracy demonstrates that the predicate logic is correctly implemented and numerically stable across the evaluated data range.

The 1.18% false positive rate reveals an important dependency on data source quality. Finnhub's free-tier API occasionally returns zero-value quotes for less-liquid symbols during extended-hours and pre-market periods, a data quality characteristic that must be communicated to users and addressed in production deployments through symbol filtering or quality gating. The alert delivery success rate of 98.7% reflects minor SMTP transient failures, addressable in future work through retry logic with exponential back-off.

The SMA signal accuracy figures confirm that the system provides genuine-if modestpredictive value. Accuracy levels of 65–67% for directional signals substantially exceed the 50% random baseline, justifying the inclusion of signal computation as a user-facing feature. However, these figures should be contextualised within the limitations of SMA methodology; investors should treat signals as one input among several rather than as autonomous trading directives.

A limitation of the current evaluation is its reliance on a single-instance SQLite deployment. Under concurrent user loads exceeding approximately 50 simultaneous poll-cycle-active accounts, SQLite's write serialisation characteristics would introduce latency jitter. Production deployments serving larger user bases should migrate to a concurrent-write-capable RDBMS such as PostgreSQL, as anticipated in the system's env.example configuration.

6. CONCLUSION AND FUTURE WORK

This paper has presented the design, implementation, and empirical evaluation of a cloud-native equity surveillance platform that delivers real-time quote monitoring, multi-predicate alert evaluation, SMA-based trend signalling, and SSE-driven browser streaming within a single containerised Python service deployed on AWS Elastic Beanstalk. The system achieves a median alert latency of 47 seconds, 100% rule evaluation accuracy, 99.7% operational availability, and a false positive rate of 1.18% across a 30-session, 25-ticker evaluation corpus. The seven-predicate rule engine with per-rule cooldown management and per-user time-window filtering provides a configurable and user-centric alerting paradigm not previously available in open-source form.

The primary limitations of the current work are the polling-based ingestion architecture's inherent latency floor, the SQLite persistence layer's concurrency ceiling, and the SMA signal methodology's limited predictive horizon. Future work will address these limitations through five principal enhancements. First, WebSocket-based data ingestion from Finnhub's streaming endpoint will reduce alert latency from O (poll interval) to O (network round-trip). Second, migration to PostgreSQL will enable concurrent multi-user deployments. Third, the signal computation module will be extended to incorporate Long Short-Term Memory (LSTM) neural network forecasting, trained on rolling 90-day close price series. Fourth, multi-exchange coverage encompassing LSE, NSE, and HKEX equities will broaden the platform's geographic applicability. Fifth, a mobile push notification channel via Firebase Cloud Messaging will complement the existing email dispatch mechanism, improving time-to-awareness for time-sensitive alerts.

REFERENCES

[1] World Federation of Exchanges, "WFE Annual Statistics Guide 2023," World Federation of Exchanges, LondoUK, 2023. [Online]. Available: https://www.world-exchanges.org

[2] D. Kuhlman, "A Python Book: Beginning Python, Advanced Python, and Python Exercises," Platypus Global Media, 2013.

[3]I.Hickson, "Server-Sent Events," W3C Recommendation, Dec. 2021. [Online]. Available: https://www.w3.org/TR/eventsource/

[4] S. Chakraborty, A. Gupta, and R. Mehrotra, "High-Frequency Trade Surveillance Using Apache Kafka and Flink: An Architectural Study," IEEE Transactions on Big Data, vol. 9, no. 3, pp. 812–826, 2023.

[5] W. Brock, J. Lakonishok, and B. LeBaron, "Simple Technical Trading Rules and the Stochastic Properties of Stock Returns," Journal of Finance, vol. 47, no. 5, pp. 1731–1764, 1992.

[6] J. Fang, Y. Qin, and B. Jacobsen, "Economic Fundamentals or Investor Perceptions? The Role of Uncertainty in Predicting Long-Term Cryptocurrency Returns," International Review of Financial Analysis, vol. 71, 2020.

DOI: 10.17148/IARJSET.2026.135113

[7] W. Mudasser, K. S. Rao, and J. Garg, "Comparative Analysis of REST-Based Financial Market Data APIs for Non-HFT Applications," in Proc. IEEE International Conference on Data Engineering (ICDE), pp. 99–107, 2022.

[8] R. Subramanian, P. Nair, and A. Krishnan, "Performance Characterisation of Python Microservices on AWS Elastic Beanstalk," in Proc. IEEE International Conference on Cloud Computing (CLOUD), pp. 221–229, 2023.

[9] J.-H. Lim and Y.-G. Kim, "Multi-Channel Financial Alert Notification Broker with Intelligent Cooldown Management," Expert Systems with Applications, vol. 214, p. 119115, 2023.

[10] C.-L. Tsai, S.-H. Chen, and M.-F. Chang, "Time-Window Constrained Alert Filtering for Financial Market Monitoring Systems," IEEE Access, vol. 11, pp. 45230–45241, 2023.

[11] T. Christie, "FastAPI: Modern, Fast Web APIs with Python," Packt Publishing, 2023.

[12] R. S. Pressman, Software Engineering: A Practitioner's Approach, 9th ed. New York: McGraw-Hill, 2019.

[13] D. Pillay and M. Brookes, "JWT-Based Authentication Patterns in Microservice Architectures: A Security Analysis," in Proc. IEEE Symposium on Security and Privacy Workshops, pp. 154–161, 2022.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," Nature, vol. 521, pp. 436–444, 2015.

[15] Amazon Web Services, "AWS Elastic Beanstalk Developer Guide," Amazon Web Services, Inc., 2024. [Online]. Available: https://docs.aws.amazon.com/elasticbeanstalk/

AUTHORS' BIOGRAPHIES

DASARI SHANMUKHI received the B.Sc. degree from BRR&GKR Chambers College, Palakol, West Godavari, India, in 2024. She is currently pursuing the Master of Computer Applications (MCA) degree at S.V.K.P. & Dr. K.S. Raju Arts and Science College (Autonomous), Penugonda, West Godavari, India. Her academic interests include cloud computing, real-time data processing, artificial intelligence, financial technology systems, and software engineering. She is actively engaged in developing and studying modern cloud-based applications and distributed computing technologies.

Dr. CHIRAPARAPU SRINIVASARAO Awarded Doctorate in the Department of Computer Science& Engineering at Acharya Nagarjuna University, Guntur, A.P. Presently, he is Working as Associative Professor in SVKP & Dr. K.S.Raju Arts & Science College (Autonomous), Penugonda, A.P. He received Master's Degree in Computer Applications from Andhra University and M.Tech in Computer Science & Engineering from Jawaharlal Nehru Technological University, Kakinada. He Qualified in UGC NET and AP SET. His research interests include Data Mining, Cloud Computing and Data Science.

.