IJIRSET
International Journal of Innovative Research in
SCIENCE | ENGINEERING | TECHNOLOGY

e-ISSN: 2319-8753 | p-ISSN: 2347-6710

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

Volume 13, Issue 9, September 2024

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.524

9940 572 462 | 6381 907 438 | ijirset@gmail.com | www.ijirset.com

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 9, September 2024

|DOI: 10.15680/IJIRSET.2024.1309127|

Real-Time Trader Input Validation Systems Using Low-Latency Java APIs and AWS Kinesis

Bharathvamsi Reddy Munisif

Senior Associate, Macquarie, Houston, TX, USA

ABSTRACT: Accurate and timely validation of trade inputs is critical in high-frequency commodity and financial trading environments. Manual or delayed validation can result in erroneous trades, compliance risks, and operational inefficiencies. This paper presents a real-time input validation system for traders, implemented using low-latency Java APIs and AWS Kinesis. The system ensures that inputs are checked against business rules and market constraints in milliseconds before proceeding to execution workflows. Our architecture leverages AWS Kinesis for high-throughput data streaming, Java-based microservices for synchronous rule enforcement, and DynamoDB for persistent rule management and transaction tracking. The system demonstrates strong scalability, low latency, and adaptability across diverse trading desks.

KEYWORDS: Real-time validation, AWS Kinesis, Java APIs, trader input, microservices, stream processing, financial systems

I. INTRODUCTION

In the domain of financial trading—particularly within commodity, equity, and derivatives markets—the integrity and accuracy of trader-entered data play a pivotal role in determining the success of transaction execution, risk management, and compliance with both internal policies and external regulations. Even minor discrepancies, such as a misplaced decimal point, an invalid contract identifier, or an improperly scaled quantity, can trigger severe downstream consequences. These may include failed trades, profit and loss discrepancies, erroneous risk exposure calculations, regulatory infractions, and reputational damage. As trading volumes increase and execution windows shrink, the tolerance for such errors continues to decrease.

Traditionally, input validation in trading systems has been implemented through basic user interface (UI) constraints or batch-processing validation mechanisms executed after the trade data is submitted. UI-level validation often includes checks for required fields, basic formatting, and dropdown restrictions, while batch-level workflows may perform sanity checks and enforce business rules prior to settlement or risk assessment stages. However, these legacy approaches suffer from critical limitations. They are reactive rather than preventive, lack real-time responsiveness, and often fail to scale under high-throughput trading conditions. More importantly, they provide feedback to traders only after significant time has elapsed—by which point the erroneous data may have already propagated to downstream systems or led to incorrect position updates. With the advent of high-frequency and algorithmic trading, there is an increasing demand for systems that can validate trader inputs in real time, ensuring compliance and correctness before orders are routed to execution engines. Real-time validation frameworks have the potential to prevent errors at the source, thereby enhancing execution quality, reducing operational overhead, and increasing trader confidence.

To address this challenge, this paper introduces a real-time trader input validation architecture that leverages low-latency Java microservices for rule evaluation and AWS Kinesis for scalable event ingestion and stream processing. The proposed system acts as an intelligent gatekeeper between the trader input source and downstream trading or settlement systems. Every trade submission is intercepted and evaluated against configurable business rules, external market data constraints, and compliance thresholds within milliseconds of user input.

Our solution builds on the following core principles:

1. Performance-first architecture with sub-100ms response times.
2. Scalable, decoupled validation logic that supports asynchronous rule updates.
3. Event-driven design using AWS Kinesis to handle bursty, high-volume trading data with ordering guarantees.
4. Feedback mechanisms that immediately alert traders on invalid inputs without delaying valid submissions.

The remainder of this paper details the system architecture, validation logic framework, Kinesis integration, and Java performance tuning strategies used to realize this platform. We further present benchmarking results, discuss deployment

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 9, September 2024

|DOI: 10.15680/IJIRSET.2024.1309127|

outcomes across real-world commodity trading desks, and outline future enhancements, including integration with anomaly detection engines and hybrid rule-ML models.

II. MOTIVATION AND BACKGROUND

In traditional financial institutions and commodity trading firms, legacy trading platforms often consist of large, monolithic architectures where multiple subsystems—such as order entry, trade execution, compliance checks, and risk management—are tightly coupled. Within these platforms, input validation logic is typically embedded within the execution or post-trade processing layers, making it difficult to isolate, modify, or scale independently. This architectural model leads to several operational challenges, especially in modern trading environments where speed, modularity, and responsiveness are critical. One major issue with such tightly coupled validation systems is the lack of real-time feedback. Traders often do not receive input validation results until long after submission, by which point erroneous trades may have already been processed or routed to downstream systems. This introduces not only operational inefficiency but also financial and compliance risk, as invalid data may trigger incorrect margin calculations, fail to meet position limits, or breach regulatory constraints. In some cases, traders must manually reverse transactions or rebook trades—an error-prone and time-consuming process that disrupts workflow and erodes trust in the system.

Additionally, scalability becomes a significant bottleneck in monolithic platforms. As trade volumes increase and trading desks diversify across asset classes and regions, the ability to handle spikes in validation requests becomes constrained. These platforms are also inflexible to rule changes—updating a validation rule often requires code changes, regression testing, and full platform redeployment. This rigidity makes it difficult for business teams to adapt quickly to new market conditions, compliance regulations, or operational insights.

To address these limitations, there is a growing need for decoupled, real-time validation systems that can operate as independent microservices, continuously evaluate trader inputs, and deliver feedback with sub-millisecond latency. Such systems must be capable of processing thousands of events per second while maintaining high availability, fault tolerance, and deterministic behavior. Moreover, they should support dynamic rule updates, allowing risk and compliance teams to adjust policies without impacting core trading logic. In this context, Java's robust ecosystem for asynchronous processing (e.g., CompletableFuture, Netty, Reactor) offers an excellent platform for building low-latency validation services. Java is widely adopted in the financial services industry, and its performance, maturity, and ecosystem support make it a strong candidate for handling real-time workloads with high reliability. Complementing Java at the infrastructure level, AWS Kinesis provides a scalable, fully managed event-streaming platform designed to ingest and process large volumes of real-time data. Its capabilities such as partitioned sharding, ordered data streams, and replayable data windows make it an ideal backbone for building stream-driven validation architectures. By integrating AWS Kinesis with Java-based microservices, it is possible to design an input validation system that is both modular and highly performant, offering immediate validation feedback without compromising throughput or fault tolerance.

This paper is motivated by the need to fill this gap—by designing, implementing, and evaluating a real-time trader input validation platform that is decoupled from execution systems, scalable across desks, and capable of operating within the latency budgets of modern electronic trading.

III. SYSTEM ARCHITECTURE OVERVIEW

The proposed real-time input validation system follows a modular, cloud-native architecture, designed for low latency, high throughput, and easy scalability. It decouples validation from trading workflows, enabling asynchronous event processing and real-time feedback. The system comprises the following core components:

- **Trader UI / Input Source**: Front-end systems or APIs where traders submit order details. These interfaces forward input as structured events for validation.
- **Java Validation Microservice**: A Spring Boot–based service that subscribes to AWS Kinesis streams and executes rule-based validations. It supports asynchronous processing, horizontal scaling, and pluggable rule engines per asset class.
- **AWS Kinesis Data Stream**: Acts as the high-throughput event bus. It supports parallelism through sharding, maintains order per partition, and enables multiple consumers (e.g., audit, analytics).
- **AWS Lambda (Optional)**: Handles lightweight tasks such as enrichment or event forwarding. Ideal for elastic, event-driven extensions without managing servers.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 9, September 2024

|DOI: 10.15680/IJIRSET.2024.1309127|

- **Amazon DynamoDB**: Stores validation rules, metadata, and transaction results. Offers low-latency, scalable NoSQL storage with high availability.
- **Notification Engine**: Delivers validation outcomes to traders and monitoring systems in real time. Supports UI feedback, logs, and alerting.

The architecture enables scalable microservice deployment, event decoupling, and real-time validation under high trading volumes. Its design supports flexible rule management, integration with external systems, and fault isolation across components.

IV. RULE FRAMEWORK AND VALIDATION LOGIC

The effectiveness of a real-time input validation system depends heavily on the flexibility, clarity, and execution speed of its rule framework. To achieve these goals, the proposed system adopts a modular, plug-and-play rule architecture where each validation rule is encapsulated as an independent unit conforming to a shared interface. This approach allows the system to dynamically evaluate rule sets, support granular testing, and simplify version management. Each rule module implements a common contract that defines input parsing, validation logic, and output formatting, making it easy to extend the system with new validation categories or logic without impacting existing services. Rule results are standardized to include a rule identifier, status (pass/fail), and optional error messages or resolution hints for the trader or system integrators.

The validation logic is grouped into the following four categories:

- **Syntactic Validation**
  This layer ensures that all required fields are present and follow the correct format. Common checks include validating data types (e.g., numeric or alphanumeric), field lengths, and mandatory attributes such as trader ID, instrument code, and trade date. Syntactic rules are typically executed first to avoid unnecessary processing of malformed inputs.
- **Semantic Validation**
  These rules evaluate the meaning and correctness of values within context. Examples include verifying that trade quantities fall within allowed trading limits, price fields are within acceptable market ranges, or that the selected instrument is tradable in the current session. Semantic validations often reference static reference data or pre-configured business thresholds.
- **Cross-Field Validation**
  Cross-field validation ensures logical consistency between related fields in a trade input. For example, a trade's quantity must be compatible with the unit of measure, or the execution date must align with the contract maturity. These rules help identify contradictions or structural errors within the data that single-field checks may miss.
- **External Validation**
  For real-time market enforcement and regulatory compliance, the system supports external validation rules that invoke third-party services or internal APIs. Examples include checking instrument status against market data feeds, ensuring regulatory flags are present for certain jurisdictions, or verifying counterparty risk limits from compliance platforms. These rules are executed asynchronously where possible to reduce latency impact.

Each rule module is designed to be independently versioned and maintained via automated CI/CD pipelines. This enables fast deployment of updated business logic without requiring full system redeployment. The rule engine supports hot-swapping of rules, allowing trading desks or compliance teams to introduce new validation logic on the fly while preserving system uptime.

Furthermore, all rule execution results are logged for auditability and diagnostics, with trace IDs allowing analysts to review which rule failed and under what conditions. Rules are also tagged with metadata such as asset class, applicable regions, and risk category, enabling context-sensitive rule activation and fine-grained control.

In summary, the rule framework offers a flexible, extensible, and high-performance validation pipeline, ensuring that trader inputs are rigorously evaluated across syntactic, semantic, logical, and external dimensions—before reaching critical downstream systems.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 9, September 2024

|DOI: 10.15680/IJIRSET.2024.1309127|

V. AWS KINESIS INTEGRATION

The validation system uses Amazon Kinesis Data Streams as its event backbone to ingest and distribute trade input events in real time. Kinesis enables scalable, low-latency, and decoupled processing, making it ideal for high-throughput trading environments.

Key features utilized include:

- **Sharding**: Events are partitioned by keys (e.g., trader ID) across shards, allowing parallel processing by multiple validation microservice instances. This ensures scalability during peak trading volumes.
- **Ordered Delivery**: Kinesis maintains event order within each shard, which is critical for consistency in validation workflows tied to sequential inputs.
- **Checkpointing with KCL**: The Kinesis Client Library (KCL) ensures fault-tolerant, exactly-once processing by tracking the last processed event, enabling smooth recovery and consistent state.

Kinesis also supports multiple downstream consumers—such as audit systems, analytics tools, and dashboards—allowing real-time validation data to be shared across the organization without tightly coupling services.

By integrating Kinesis, the system achieves a robust, extensible, and event-driven architecture capable of supporting evolving trading workloads and validation requirements.

VI. JAVA API DESIGN AND PERFORMANCE OPTIMIZATION

To meet the stringent performance requirements of real-time trade validation, the system's microservices were designed using Spring Boot and optimized with a Netty-based non-blocking web server. This combination provides a lightweight, production-grade environment for building RESTful APIs with support for high concurrency and low-latency request handling.

The design emphasizes asynchronous, event-driven processing, allowing the validation service to respond rapidly to incoming trade events from AWS Kinesis while avoiding thread-blocking or resource contention. Several critical optimization strategies were implemented to ensure consistent performance under high load conditions:

A. Connection Pooling and Thread Tuning

Efficient use of HTTP connection pools and thread executors is essential in high-throughput environments. The system leverages configurable connection pooling mechanisms (via Apache HttpClient and Netty) to reduce connection overhead when communicating with external systems (e.g., compliance APIs, market data services). Thread pools are fine-tuned based on CPU core count, memory usage, and expected concurrency levels to ensure optimal resource utilization without overprovisioning.

B. Asynchronous Processing with CompletableFuture and Reactor

To prevent blocking I/O operations, the microservice pipeline uses Java's CompletableFuture and Project Reactor for asynchronous execution. Each trade event is validated via a non-blocking, reactive workflow, allowing multiple validations to run in parallel across different rule categories. This architecture significantly improves throughput while minimizing context switching and memory pressure under heavy loads.

C. JSON Parsing Optimization

Given the high volume of JSON-formatted input, efficient parsing is a key performance driver. The system uses Jacksonas the primary parser, selected for its speed and memory efficiency. Benchmark comparisons were conducted against Gson, confirming Jackson's superior performance in deserializing nested trade structures. For latency-critical fields, custom deserializers and precompiled schema mappings are used to further reduce serialization overhead.

D. Precompiled Rule Execution

To avoid runtime interpretation delays, rule logic—especially for frequently executed or latency-sensitive validations—is precompiled and cached in memory during service startup. This reduces execution overhead and improves the determinism of response times. Rules are structured as lightweight function interfaces, enabling fast invocation and easy substitution during rule version upgrades.

E. Performance Benchmarking and Load Testing

Comprehensive performance testing was performed using synthetic trade events at varying volumes and complexity levels. Key results include:

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 9, September 2024

|DOI: 10.15680/IJIRSET.2024.1309127|

- **Median response time**: < 50 milliseconds per validation request.
- **99th percentile latency**: < 100 milliseconds, even during burst conditions.
- **Throughput**: Sustained rate of 5,000 events per second per service instance, with linear scalability as more instances are added.

These benchmarks confirm that the system meets the real-time requirements for modern trading platforms and can scale predictably across commodity desks or regional deployments.

In summary, the combination of non-blocking API design, reactive processing, and lightweight Java frameworks enables the validation microservices to operate with high efficiency and reliability. These design choices ensure the platform can handle the demands of high-frequency trading environments without sacrificing performance or flexibility.

VII. RESULTS AND BENCHMARKS

To evaluate the effectiveness and performance of the proposed real-time trader input validation system, we conducted a series of controlled benchmarks within a simulated environment that replicated typical trading scenarios. The test environment was configured to mimic realistic production conditions, including asynchronous event ingestion, multiple concurrent trader sessions, and diverse trade structures across commodities such as energy, metals, and agriculture.

The validation engine was deployed using AWS infrastructure components (Kinesis, Lambda, DynamoDB) in conjunction with containerized Java microservices running on Amazon ECS. Synthetic trade events were generated using randomized, rule-based templates to simulate both valid and invalid data patterns commonly encountered in live trading desks.

The key performance metrics and outcomes are summarized below:

A. Scalability

The system exhibited linear scalability as additional Kinesis shards and validation microservice replicas were added to the deployment. As trade input volume increased, the throughput capacity of the system scaled proportionally, with no degradation in performance or increased latency. This confirms that the event-streaming and microservice-based architecture can efficiently accommodate high-volume trading scenarios, including global desks operating in parallel.

- Test Scenario: 1–10 Kinesis shards; 2–20 microservice instances.
- Observation: Throughput increased linearly, validating horizontal scalability.

B. Latency

Real-time responsiveness is critical in trading workflows, where delayed validation feedback can negatively affect execution speed and trader decision-making. Across all test cases, the system consistently returned validation results within tight latency thresholds:

- **95% of responses were delivered within 75 milliseconds**, including rule evaluation and feedback generation.
- The **median latency** remained under 50 milliseconds, with low variability even under burst loads.
- **99th percentile latency** stayed below 100 milliseconds, meeting real-time trading performance benchmarks.

These results confirm the system's suitability for low-latency environments such as algorithmic trading platforms and high-frequency commodity desks.

C. Resilience

To test fault tolerance and stress behavior, the system was subjected to sudden bursts of up to 10 times the average event volume, simulating real-world surges during market openings, breaking news, or macroeconomic events.

- Despite extreme traffic spikes, the system maintained full availability and processing continuity.
- Kinesis-based backpressure mechanisms and autoscaling policies allowed the system to absorb and recover from high loads without event loss or service disruption.
- No system crashes, data corruption, or processing deadlocks were observed during load testing.

D. Accuracy

Accuracy in validation is essential to ensure that valid trades are not incorrectly blocked, and that all invalid trades are correctly identified before reaching downstream systems.

- During testing, all intentionally malformed or invalid inputs were accurately flagged, with no missed violations.
- The system achieved 0% false positives, meaning no valid trades were incorrectly rejected.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 9, September 2024

|DOI: 10.15680/IJIRSET.2024.1309127|

- Rule execution logs were reviewed for each failure case, and outcomes were consistent with expected business logic.

This level of accuracy ensures trustworthiness and compliance alignment, giving traders and compliance teams confidence in the system's judgments.

In summary, the proposed validation engine demonstrates excellent performance across key operational metrics—scalability, low latency, resilience, and rule accuracy. These results validate the system's readiness for production deployment in mission-critical financial environments where speed and correctness are paramount.
valid inputs were correctly flagged, with 0% false positives during testing.

VIII. BUSINESS IMPACT AND ADOPTION

The implementation of the real-time trader input validation system had a measurable and positive impact on day-to-day trading operations and compliance oversight. By validating trade inputs at the point of entry—before they reached execution systems—the solution significantly improved trader productivity and reduced the operational overhead associated with trade correction and rework. Early deployment across energy and metals trading desks showed particularly strong results. The system led to a 40% reduction in manual corrections, which previously occurred due to misentered values, inconsistent field formats, or missing regulatory flags. Traders were able to receive immediate, actionable feedback on invalid inputs, allowing them to resolve issues in real time without needing follow-up from back-office or compliance personnel.

From a compliance perspective, the system also improved audit traceability. Every validation event—pass or fail—was logged with metadata including rule version, timestamp, input data, and resolution status. This allowed compliance teams to quickly trace and investigate validation failures or anomalies and provided a clear audit trail for internal governance and external regulatory inspections.

Furthermore, incident response times were shortened, as the system's feedback loop enabled early detection of problematic patterns such as frequent entry errors, incorrect instrument codes, or repeated boundary violations. This allowed for proactive intervention, rule adjustment, or trader retraining, reducing the likelihood of systemic issues.
The modular nature of the validation engine also made it easier to onboard new products and desks, as rule sets could be customized and deployed without altering the core application logic. This flexibility accelerated the system's adoption across multiple teams and laid the groundwork for broader enterprise integration.

Overall, the system demonstrated not just technical robustness but also tangible business value—streamlining workflows, enhancing data quality, improving compliance readiness, and supporting a culture of automation and accountability within trading operations. Compliance teams reported improved audit traceability and incident response capabilities.

IX. FUTURE WORK AND ENHANCEMENTS

To extend the system's flexibility and intelligence, several improvements are planned:

- **Rule DSL (Domain-Specific Language)**: A custom language will allow business users to define or modify validation rules without code changes, enabling faster, more transparent rule updates.
- **Machine Learning for Anomaly Detection**: Anomaly detection models will supplement rule-based checks by flagging unusual trader inputs based on historical patterns, enhancing detection of subtle errors or outliers.
- **Streaming Aggregates**: The system will support validations based on rolling metrics—such as cumulative trade volume or per-trader limits—enabling context-aware rule enforcement using real-time data streams.
- **Kafka Compatibility**: In addition to AWS Kinesis, the system will integrate with Apache Kafka to support hybrid or on-premise infrastructures, improving portability across enterprise environments.

X. CONCLUSION

Real-time validation of trader inputs plays a critical role in ensuring the accuracy, compliance, and efficiency of modern trading systems. As trade volumes increase and regulatory scrutiny intensifies, the need for low-latency, high-throughput validation systems becomes more apparent. This paper presented a robust architecture for trader input validation, built using Java microservices and integrated with AWS Kinesis for scalable, event-driven data streaming.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 9, September 2024

|DOI: 10.15680/IJIRSET.2024.1309127|

The proposed solution addresses common challenges found in legacy trading platforms, including delayed feedback loops, tight coupling between validation and execution logic, and poor scalability. By adopting a modular microservices approach with asynchronous Java APIs, the system achieves sub-100ms response times while maintaining high availability and operational resilience. The integration with AWS Kinesis ensures reliable ingestion and distribution of trade events, enabling parallel processing, ordering guarantees, and support for downstream consumers such as audit systems and analytics engines. Through comprehensive testing and benchmarking, the system has demonstrated strong performance in terms of scalability, fault tolerance, and validation accuracy. The ability to process thousands of trade events per second with consistent low-latency responses makes it well-suited for high-frequency trading environments. Moreover, the framework's extensibility—through pluggable rule engines, external API hooks, and planned DSL and ML integrations—positions it as a future-ready platform capable of evolving with business and regulatory needs.

In summary, this work provides a reliable, scalable, and adaptable foundation for real-time input validation in financial systems. It not only enhances trader productivity and reduces operational risk but also reinforces regulatory compliance and organizational auditability. Future enhancements, such as streaming aggregates, anomaly detection, and Kafka compatibility, will further strengthen the platform's role as a core component of next-generation trading infrastructures proves to be a reliable, modular, and scalable foundation for modern trading platforms.

REFERENCES

[1] Amazon Web Services, "Amazon Kinesis Documentation," 2024. [Online].
Available: https://docs.aws.amazon.com/kinesis/
[2] Spring Boot Documentation, VMWare, 2024. [Online]. Available: https://spring.io/projects/spring-boot
[3] R. Sedgewick and K. Wayne, "Algorithms," 4th ed., Addison-Wesley, 2011.
[4] M. Fowler, "Patterns of Enterprise Application Architecture," Addison-Wesley, 2003.
[5] J. Bloch, "Effective Java," 3rd ed., Addison-Wesley, 2018.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details