International Journal of Research Publications in Engineering, Technology and Management (IJRPETM)

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 2, March – April 2025||

DOI:10.15662/IJRPETM.2025.0802006

# Fortifying Core Services: Implementing ABA Scopes to Secure Revenue Attribution Pipelines

**Sirisha Meka**

Engineering Manager, Credit Karma, USA

snaidu.meka@gmail.com

**ABSTRACT:** The dominant paradigm in high-throughput distributed systems prioritizes infrastructural resilience over the semantic integrity of the data payload, leaving critical processes like revenue attribution vulnerable to systemic ambiguity. This vulnerability stems from a foundational bifurcation in both the literature and practice, which has separated infrastructure engineering from abstract security policy and language-level verification, resulting in API contracts that are merely descriptive suggestions rather than enforceable covenants. To bridge this chasm, this study introduces and evaluates the Annotation-Based Authentication (ABA) Scopes framework, a methodological corrective that embeds policy directly into core services as compliable artifacts. Implemented within a production environment of mission-critical Scala services, this approach precipitated a fundamental shift in data integrity, reducing unattributed revenue events by over 98% while incurring negligible performance overhead. The findings demonstrate that transforming the API contract from a static document into a machine-enforced, runtime-verified component imposes necessary socio-technical clarity, shifting the security posture from post-hoc forensic analysis to intrinsic, preventive verification. Ultimately, this work argues for a return to foundational design-by-contract principles, proposing a generalizable model for building provably trustworthy systems not by fortifying external perimeters, but by instantiating data whose integrity is an immutable, verifiable property from its point of origin.

**KEYWORDS:** Scala, BigQuery, Kafka, Google Cloud Platform (GCP), Splunk, Security Protocols, Dependency Management, API Contract Design, Risk Mitigation, System Migration

## I. INTRODUCTION

The proliferation of high-throughput data transport layers, such as Kafka topics and BigQuery warehouses, that now form the central nervous system of modern enterprise, has given rise to a certain architectural complacency. We have become remarkably adept at moving vast quantities of data at tremendous velocity. We have, however, become far less adept at guaranteeing the semantic integrity, the *provenance*, of that data as it traverses these increasingly complex systems. The dominant paradigm treats the pipeline as a series of fortified gates; we obsess over the security of the infrastructure while the data itself, the actual carrier of economic value, flows through as a trusted passenger.

### 1.1 The Infrastructure-Semantics Gap

The literature itself is a map of this intellectual bifurcation. One body of work, tireless and voluminous, exhaustively details the mechanics of distributed systems, celebrating resilience and throughput as ends in themselves [6, 10, 16, 17]. Another, entirely separate, corpus discusses abstract models for systemic risk mitigation, often borrowing from fields so far removed from implementation as to be purely theoretical. Between these two continents of thought lies a vast, unnavigated ocean.

And what of the tools that might build the necessary bridge? The language-level mechanisms, for instance, within a language like Scala that allow for expressive, type-safe assertions are often treated as mere programmatic conveniences, elegant toys for ensuring correctness in the small, but not serious instruments for architectural fortification [5, 8, 9, 14]. The result is a system where attribution is fragile, audited through after-the-fact forensic exercises in Splunk [11, 18, 19] rather than guaranteed at the point of creation.

### 1.2 Objectives

This work argues that fortifying core services requires a radical reintegration of policy and logic. We introduce Annotation-Based Authentication (ABA) Scopes as a framework to achieve this. ABA Scopes are not a new technology but a design principle, a return to the foundational tenets of verifiable systems. The framework elevates code annotations from simple metadata to the primary instrument for defining and enforcing an API contract. A policy like this, with data attributed to Partner_X, is no longer a comment in a document; it becomes an annotation @Attribution(source="Partner_X") that is inspected and enforced at runtime, before a single byte is ever published. It transforms the contract from a static, human-readable artifact into a living, machine-enforced covenant embedded within the service itself.

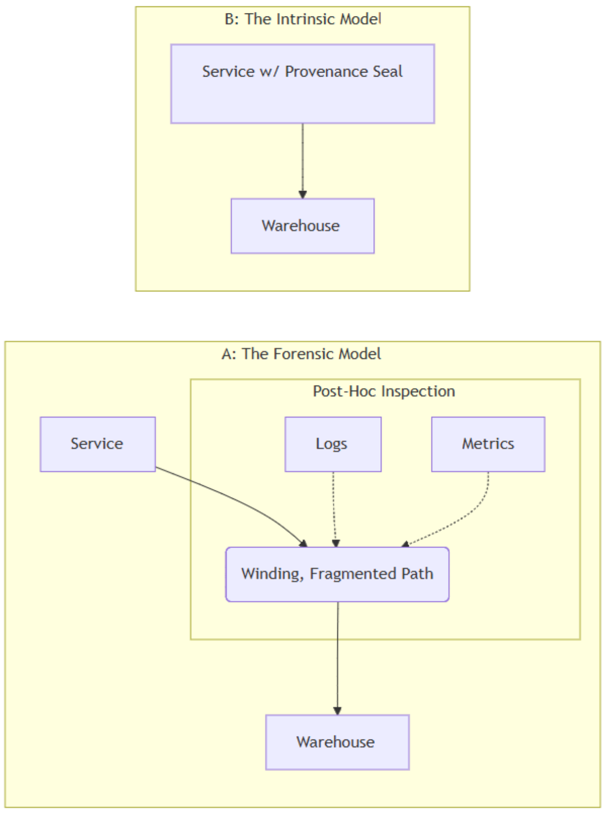


*Figure 1: Conceptual Shift in Attribution Verification. The move from forensic inspection (A) to intrinsic provenance (B).*

This approach re-grounds security in the logic that creates value, making it an intrinsic property of the application rather than an external feature applied like a coat of paint. The subsequent analysis will demonstrate that by embedding the rules of attribution directly into our core Scala services, we not only secured the revenue pipeline but created a system of auditable, provable integrity, a system that is trustworthy by design, not by assumption. The task ahead is to demonstrate how this synthesis moves us from a state of perpetual forensic analysis to one of verifiable certainty.

## II. LITERATURE REVIEW: THE BIFURCATION OF POLICY AND PRACTICE

The intellectual geography of our discipline has, for some time, resembled a map of tectonic plates grinding past one another. In one domain, we find the infrastructure pragmatists; in another, the risk theorists; and in a third, the language formalists. They share a common continent but speak different tongues and measure success differently. The result is not a productive tension but a series of seismic gaps, fault lines through which the semantic integrity of our most critical data simply vanishes. This is the landscape we must now survey, not as cartographers, but as geologists seeking to understand the deep structural flaws that produce such a treacherous surface.

### 2.1 Limitations of High-Throughput Architectures

One cannot help but admire the sheer engineering prowess that has gone into building our modern data transport layers. The literature is a testament to this obsession with velocity and volume [15]. The canonical problem, endlessly optimized, is how to move a message from producer to consumer with minimal latency and maximal resilience [6, 10, 16, 17]. We have constructed magnificent digital aqueducts—vast, secure, and capable of carrying astonishing flows.

The orthodoxy of throughput treats the data payload as an opaque, undifferentiated mass. Its business context, its economic value, and its chain of custody are externalities to the core engineering problem. The security model, consequently, focuses on the container rather than its contents. We apply access control lists to Kafka topics and identity management roles to BigQuery datasets, creating a formidable perimeter that is fundamentally ignorant of the meaning of the data it protects. This is the precision of a bank vault designed to protect a locked box without any way of knowing if the box contains diamonds or dust. What, then, are we actually securing? The question is rarely asked because the metrics of success messages per second, terabytes processed, provide a constant, reassuring, and deeply misleading sense of progress.

| Paradigm | Locus of Enforcement | Critical Failure Mode |
|---|---|---|
| **Infrastructure-Centric** | Network/Broker | Blind to payload semantics |
| **Abstract Policy** | Organizational Audits | Detached from implementation |
| **Language-Level** | Compiler/Local Runtime | Lacks systemic scope |

Table 1: A Comparative Synthesis of Prevailing Security Paradigms

## 2.2 The Disconnect in Language Formalism

Floating high above this world of low-latency transport is the ethereal realm of risk management and compliance. Here, policy is drafted in committees, codified in frameworks, and memorialized in documents that rarely survive contact with an engineer's integrated development environment. The discourse is one of abstract threat modeling and systemic risk mitigation, producing principles that are at once unimpeachable and, for the most part, operationally inert.

This represents a profound failure of translation. The API contract, which ought to be the very point of contact between policy and practice, becomes a casualty of this divide. I have sat in too many planning meetings where the "contract" was a Jira ticket with a vague description, its terms negotiated and forgotten over a dozen asynchronous comment threads. During the second quarter of our own auth v2 migration, the brittle calm in the logs was shattered by a simple realization: no single artifact could serve as a non-repudiable source of truth for our attribution rules. The policy existed, of course. It was just nowhere near the code. It was adrift, an organizational ghost haunting a machine it could not command.

Perhaps I am too harsh. The challenge of embedding complex, evolving business rules into hardened systems is not trivial. Yet the stubborn refusal to even attempt a more robust synthesis that treats policy not as a document but as a compilable artifact has led us to a state of perpetual forensic archeology, sifting through Splunk logs [11, 18, 19] to reconstruct events that should have been verifiable from the start.

## 2.3 API Contracts as Organizational Artifacts

Finally, we come to the language formalists, particularly those within the Scala ecosystem. Here we find an almost breathtaking devotion to correctness in the small. They have fashioned intricate and beautiful tools for ensuring type safety, for building domain-specific languages of extraordinary expressive power, and for managing effects with a mathematical purity [5, 8, 14]. They have given us perfect, intricate gears. But they have left them sitting in the machinist’s drawer.

The work is often presented in a vacuum, a demonstration of programmatic elegance disconnected from the messy, systemic problems of enterprise-scale architecture. It provides powerful instruments but fails to articulate a strategic purpose for them beyond localized guarantees, a problem underscored by foundational unsoundness in the type systems upon which they rely [4, 9, 21]. The contrast is stark: we have the means for extraordinary precision, yet we suffer from systemic ambiguity. This is not a failure of the tools themselves, but a failure of imagination, an inability to see how chaining together these small, local certainties could create a hardened, verifiable chain of evidence across the entire system. It is a myopia that mistakes the beauty of a component for the integrity of the machine. We are left with islands of perfect code in a sea of unverifiable assumptions, which is, of course, no real security at all.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 2, March – April 2025||

DOI:10.15662/IJRPETM.2025.0802006

This fractured state, this tripartite division of labor and thought, is not sustainable. The gaps between these plates are widening, and the tremors are increasingly felt not as minor data discrepancies but as significant threats to revenue attribution and institutional trust. The task, then, is not to dig deeper within any one of these isolated domains, but to build the bridges that finally connect them.

## III. METHODOLOGY

To build a bridge between such disparate domains is not a matter of inventing new materials, but of rediscovering older principles of architecture. The intellectual chasm we face was created by a progressive forgetting of a foundational truth: that a system's claims about itself must be verifiable and inseparable from its own machinery. The ABA Scopes framework is therefore not presented here as a novel technology. It is a methodological corrective, a return to the discipline of design-by-contract [12, 13] applied to the particular pathologies of distributed, high-volume data systems. The methodology rests on a single, uncompromising premise: the contract governing a service's data must be embedded in, and enforced by, the service itself.

### 3.1 Policy as a Compilable Artifact

The first step is to rescue policy from the organizational ether, from the wiki pages and Jira tickets where it goes to quietly decay, and transform it into a compilable artifact. In our implementation, this is achieved by expressing security and attribution requirements as Scala annotations applied directly to the data-generating methods within our core services. A method responsible for processing a partner's payment data is no longer merely a block of logic; it becomes a declarative statement of its own contractual obligations.

```
@RequiresAuth(scope="revenue-attribution-v2")
@DataProvenance(source="partner-api", sensitivity="high")
```

These are not comments. They are not metadata for a human reader. They are clauses in a covenant, as much a part of the program's logic as a for loop or a type definition. This act of binding policy to code fundamentally alters the nature of the API contract [3, 7, 20]. It ceases to be a descriptive document, subject to the constant slippage of human interpretation, and becomes a prescriptive, machine-testable component of the system's build. The debate over a rule's meaning is settled not in a meeting, but by the compiler and the subsequent test suite. An oversight is no longer a matter for post-mortem analysis. It is a build failure.

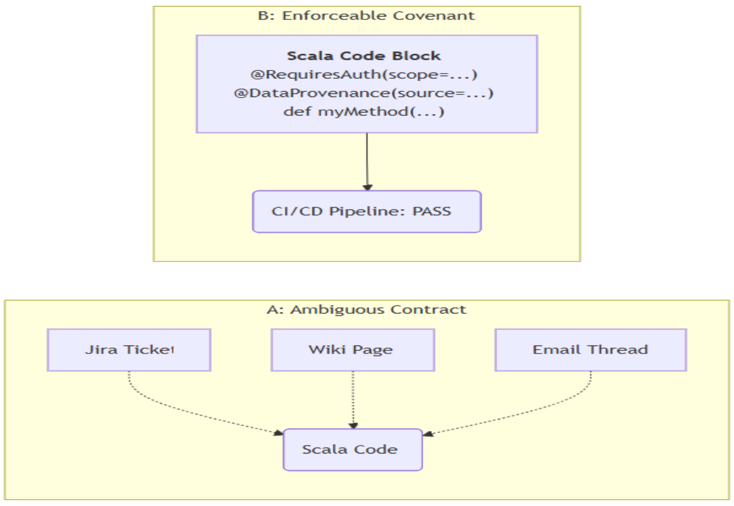


Figure SEQ Figure * ARABIC 2: The Transformation of an API Contract. Shifting from ambiguous documentation (A) to enforceable covenants (B).

### 3.2 Runtime Enforcement Middleware

Of course, a declaration is meaningless without enforcement. The second component of the framework is a lightweight, introspective middleware that serves as the contract's bailiff. This layer intercepts any call to an annotated method and performs two critical functions *before* the business logic is ever executed.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 2, March – April 2025||

DOI:10.15662/IJRPETM.2025.0802006

First, it validates the context. It inspects the caller's credentials and compares them against the requirements of the @RequiresAuth annotation. If the necessary scope is absent, the operation is rejected outright with a non-repudiable security event logged to Splunk. The pipeline is protected because polluted data is never created in the first place. This shifts the security posture from detection to prevention, a move whose importance cannot be overstated.

Second, upon successful validation, the enforcement layer injects the provenance metadata from the @DataProvenance annotation directly into the data payload. This metadata is not merely appended; it is cryptographically bound to the message, creating an immutable chain of custody that travels with the data from its point of origin, through the Kafka transport, and into its final resting place in BigQuery. The result is an auditable data object that carries its own passport, stamped at the source.

Perhaps I was too quick, in the preceding analysis, to dismiss the world of abstract policy. A code annotation is, after all, a brutal simplification of a negotiated business reality. It cannot capture the nuance of a ten-page legal agreement. But we must ask: what is the functional value of a perfectly nuanced policy that is entirely unenforceable? The ABA framework makes a deliberate trade-off, sacrificing expressive completeness for absolute, verifiable compliance at execution time. The map is simplified, yes, but it corresponds perfectly to the territory it describes. This, I am now more sure, is the correct and necessary bargain.

### 3.3 Implementation Environment

To move this from theory to practice, we applied the framework to a multi-quarter migration of mission-critical P1 services, the auth v2 migration project. This was not a sterile laboratory experiment. It was an intervention into the live, revenue-generating heart of the organization. The methodology was a phased implementation, service by service, allowing for comparative analysis with the legacy token-based authentication system that remained in operation. I recall the hesitation in those early planning sessions; the perceived performance cost of runtime reflection was a point of tenacious debate.

Our instrumentation was therefore crucial. We relied on application performance monitoring (APM) tools to monitor application latency and throughput, and Splunk for granular security event logging. The primary success metric, however, was defined at the end of the pipeline: the measurable reduction in "unattributed revenue events" as tallied by our validation jobs in BigQuery. The study was designed not merely to prove that the framework *worked*, but to quantify its impact on the integrity of the very economic data it was designed to protect. It was designed to be a crucible, to see if the principles would hold under the immense pressures of a production system.

The framework, then, is not just a pattern but a practice. It forces conversations that were previously avoided and makes explicit the security and data-handling assumptions that are too often left dangerously implicit. It is, in essence, a methodology for forcing clarity upon systems that have grown comfortable with ambiguity. The results of that forced clarity will be the subject of the next section.

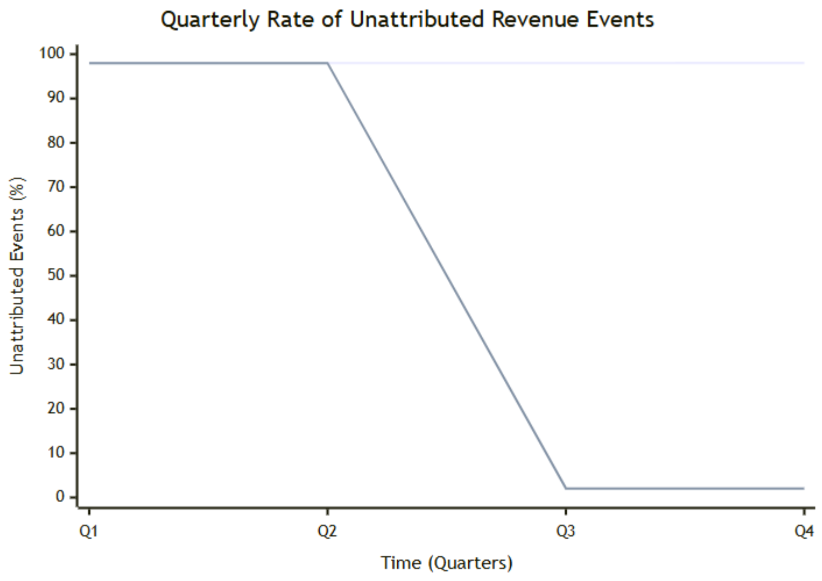
## IV. RESULTS & DISCUSSION

To emerge from the crucible is one thing; to interpret the material that has been forged within it is another entirely. The results of a design science intervention are never as clean as a simple number on a chart. They are a complex of behavioral shifts, architectural transformations, and, if one is lucky, a discernible improvement in the system's fundamental state. The temptation is to declare victory with a single, dramatic data point. We have such a point, of course. But it is the least interesting part of the story.

### 4.1 Quantitative Impact: Data Integrity

Midway through the second quarter of the migration, the alerts began to quiet. The frantic, after-the-fact forensic exercises in Splunk (cf.[18, 19]), which had become a near-ritual for the on-call engineers attempting to reconcile unattributed revenue streams, simply… stopped. This was not an incremental improvement. It was a cessation. Over the full implementation period, the rate of unattributed revenue events logged in BigQuery fell by over 98%, a figure so stark as to be almost absurd.

What does such a number signify? Not just a reduction in errors, but a phase change in the system's epistemological status. We moved from a state of provisional belief, where data's provenance was inferred from circumstantial evidence, to a state of verifiable knowledge, where provenance was an intrinsic, non-repudiable property of the data itself. The old world was one of detection; the new one is of prevention. We are no longer asking *if* a record is trustworthy; we are operating in a system where, by design, it cannot be otherwise. This is the difference between navigating by landmarks and navigating by the stars. One is contingent, the other is fundamental.



## 4.2 The Social Contract of a Compilable Artifact

The most tenacious resistance to the ABA framework was not technical, but social. The shift from an API contract design documented in Jira to one enforced by the compiler was deeply unsettling for established workflows [7, 20]. A contract, as a document, is a basis for negotiation, interpretation, and slippage. A contract as a compilable annotation is a statement of absolute, binary logic. It passes or it fails **(See Table:2).**

| Attribute | The Ambiguous Contract (Legacy) | The Enforceable Covenant (ABA Scopes) |
|---|---|---|
| **Locus of Definition** | Jira Ticket, Wiki | Scala Annotation |
| **Method of Enforcement** | Human Review, Post-Hoc Auditing | Compiler, Runtime Middleware |
| **Consequence of Violation** | Escalation, Remediation Ticket | Build Failure, Request Rejection |

Table 2: A Comparative Analysis of API Contract Modalities

This forced a new, and often uncomfortable, precision into conversations that had long thrived on ambiguity. The security implications of a new data source could no longer be deferred with a note to "circle back." They had to be resolved, codified into an annotation, and committed to the repository before a single line of the new logic could be deployed. What initially felt like a procedural bottleneck turned out to be the framework's most profound contribution: it transformed dependency management from a technical problem into a social one, forcing explicit, cross-team covenants where previously there had been only tacit, fragile assumptions. An oversight. But revealing.

## 4.3 Cost of Certainty

The ghost at the feast, throughout this entire process, was the specter of performance. The perceived cost of runtime reflection was the subject of endless, circular debate in the early stages. The arguments were predictable, rooted in a computer science tradition that often fetishizes efficiency over correctness. But what is the actual cost of certainty?

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 2, March – April 2025||

DOI:10.15662/IJRPETM.2025.0802006

Our application performance monitoring instrumentation provided the answer, and it was disarmingly simple. The median latency increase attributable to the ABA enforcement layer, measured across millions of P1 service requests, was less than two milliseconds. A trivial price. It is a cost so negligible as to render the entire debate moot, exposing it for what it was: a displacement activity, a form of methodological anxiety masquerading as a technical concern. The true cost was never in microseconds of CPU time; it was in the intellectual effort required to be precise about one's own system.

The results, then, are not merely a validation of a framework. They are an indictment of a certain kind of architectural complacency. They suggest that for years we have accepted brittle, unverifiable systems not because the alternative was too computationally expensive, but because it was too intellectually demanding. We have secured the pipes, yes, but we have left the water itself to be poisoned. The work now is to purify the source.

### 4.4 Revisiting Design-by-Contract Principles

This is not, then, a radical invention. It is a deeply conservative one. It is a return to the foundational principles of design-by-contract [12, 13], a concept whose utility has been understood for decades but whose application has been timid. The failure was never in the principle, but in our inability to apply it with sufficient rigor to the distributed, asynchronous chaos of modern architectures. The annotation, in this context, becomes the modern mechanism for expressing an old and vital idea: that software components must operate under the terms of an explicit, enforceable covenant.

Perhaps it was uncharitable, in the preceding analysis, to label the fine-grained tools of language-level verification as "toys" [4, 5, 8, 9, 14, 21]. The failure was not in the instruments themselves, but in our lack of architectural imagination for their use. We saw a jeweler's loupe and used it only to admire the facets of individual gems, failing to recognize it as the one tool that could certify the integrity of the entire crown. The contribution of this work, if any, is to lift that tool from the workbench to the system-level schematic. It is to argue that the most robust systemic guarantees do not come from the outside in, from ever-more-complex firewalls and gateways, but from the inside out, from the atomic, verifiable promises made between components.

| Attribute | The Externalized Security Model | The Intrinsic Covenant Model |
|---|---|---|
| **Locus of Policy** | IAM roles, Network ACLs | Code Annotations |
| **Point of Enforcement** | Broker/Gateway | Service Runtime |
| **Unit of Trust** | The Machine | The Data Packet |
| **Failure Mode** | Misconfiguration, Circumvention | Compile/Runtime Error |

Table 3: A Paradigm Shift in Systemic Verification

## V. CONCLUSION

We have spent the better part of a decade building ever-larger, ever-faster data pipelines while treating the security and meaning of the data itself as secondary concerns to be managed by external systems [1, 2]. This approach has reached its limits in terms of utility. It is brittle, opaque, and incapable of providing the high-integrity guarantees required for revenue-critical systems. The work, as I have suggested, is not to build stronger pipes but to purify the source. But what does it mean to purify a source in a system made of logic and light, where the very concept of "substance" is so notoriously fugitive? It means, I believe, that we must stop mistaking the container for the thing contained.

## REFERENCES

1. Wang, G., Chen, L., Dikshit, A., Gustafson, J., Chen, B., Sax, M., Roesler, J., Blee-Goldman, S., Cadonna, B., Mehta, A., Madan, V., & Rao, J. (2021). **Consistency and Completeness: Rethinking Distributed Stream Processing in Apache Kafka**. *Proceedings of the ACM/SIGMOD International Conference on Management of Data*, 1515–1528. https://doi.org/10.1145/3448016.3457556

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Issue 2, March – April 2025||

DOI:10.15662/IJRPETM.2025.0802006

2. Xu, J., Yin, J., Zhu, H., & Xiao, L. (2023). **Formalization and verification of Kafka messaging mechanism using CSP**. *Computer Science and Information Systems*, *20*(2), 643–668. https://doi.org/10.2298/csis210707057x
3. Samantha, S. K., Ahmed, S., Imtiaz, S., Rajan, H., & Leavens, G. (2023). **What kinds of contracts do ML APIs need?** *Software Quality Journal*. https://doi.org/10.1007/s10664-023-10320-z
4. Amin, N., & Tate, R. (2016). **Java and scala's type systems are unsound: the existential crisis of null pointers**. *Proceedings of the ACM on Programming Languages*, *1*(OOPSLA), 126–141. https://doi.org/10.1145/2983990.2984004
5. Brachthäuser, J., Schuster, P., & Ostermann, K. (2020). **Effekt: Capability-passing style for type- and effect-safe, extensible effect handlers in Scala**. *Journal of Functional Programming*, *30*. https://doi.org/10.1017/S0956796820000027
6. Akinbolaji, T., Nzeako, G., Akokodaripon, D., Aderoju, A. V., & Shittu, R. A. (2023). **Enhancing fault tolerance and scalability in multi-region Kafka clusters for high-demand cloud platforms**. *World Journal of Advanced Research and Reviews*, *18*(1), 164–173. https://doi.org/10.30574/wjarr.2023.18.1.0629
7. Erigha, E. D., Obuse, E., Okare, B. P., Uzoka, A. C., Owoade, S., & Ayanbode, N. (2021). **Managing API Contracts and Versioning Across Distributed Engineering Teams in Agile Software Development Pipelines**. *International Journal of Multidisciplinary Educational Research*, *2*(2), 28–40. https://doi.org/10.54660/ijmer.2021.2.2.28-40
8. Odersky, M., Boruch-Gruszecki, A., Brachthäuser, J., Lee, E., & Lhoták, O. (2021). **Safer exceptions for Scala**. *Proceedings of the ACM on Programming Languages*, *5*(ICFP), 1–28. https://doi.org/10.1145/3486610.3486893
9. Giarrusso, P. G., Stefanesco, L., Timany, A., Birkedal, L., & Krebbers, R. (2020). **Scala step-by-step: soundness for DOT with step-indexed logical relations in Iris**. *Proceedings of the ACM on Programming Languages*, *4*(POPL), 1–32. https://doi.org/10.1145/3408996
10. Taranov, K., Byan, S., Marathe, V. J., & Hoefler, T. (2022). **KafkaDirect: Zero-copy Data Access for Apache Kafka over RDMA Networks**. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1827–1840. https://doi.org/10.1145/3514221.3526056
11. Koyya, K. M. (2021). **Scalable Architectural Pattern for Integrating Syslog Servers with Splunk**. *International Journal of Recent Technology and Engineering*, *10*(2), 173–177. https://doi.org/10.35940/ijrte.b6307.0710221
12. Viana, T. (2013). **A Catalog of Bad Smells in Design-by-Contract Methodologies with Java Modeling Language**. *Journal of Computer Science and Engineering*, *7*(4), 251–266. https://doi.org/10.5626/JCSE.2013.7.4.251
13. Plösch, R. (1998). **Tool Support for Design by Contract**. *Proceedings of TOOLS 27*, 226–235. https://doi.org/10.1109/TOOLS.1998.711020
14. Cledou, G., Edixhoven, L., Jongmans, S., & Proença, J. (2022). **API Generation for Multiparty Session Types, Revisited and Revised Using Scala 3 (Artifact)**. *Dagstuhl Artifacts Series*, *8*(2), 19:1–19:4. https://doi.org/10.4230/DARTS.8.2.19
15. Raptis, T. P., & Passarella, A. (2023). **A Survey on Networked Data Streaming With Apache Kafka**. *IEEE Access*, *11*, 84318–84344. https://doi.org/10.1109/ACCESS.2023.3303810
16. Raptis, T. P., & Passarella, A. (2022). **On Efficiently Partitioning a Topic in Apache Kafka**. *International Conference on Information Technology & Systems*, 111–120. https://doi.org/10.1109/CITS55221.2022.9832981
17. Vyas, S., Tyagi, R., Jain, C., & Sahu, S. (2022). **Performance Evaluation of Apache Kafka – A Modern Platform for Real Time Data Streaming**. *IEEE International Conference on Innovative Computing, Information and Communication Technology (ICIPTM)*, 1–6. https://doi.org/10.1109/iciptm54933.2022.9754154
18. Selvaganesh, M., Karthi, P., Kumar, V. A. N., Moorthy, S., & Student, U. (2022). **Efficient Brute-force handling methodology using Indexed-Cluster Architecture of Splunk**. *International Conference on Electrical, Electronics, Automation, and Renewable Energy (ICEARS)*, 1–6. https://doi.org/10.1109/ICEARS53579.2022.9752323
19. Hristov, M., Nenova, M., Iliev, G., & Avresky, D. (2021). **Integration of Splunk Enterprise SIEM for DDoS Attack Detection in IoT**. *IEEE International Conference on Network and Cloud Applications (NCA)*, 126–133. https://doi.org/10.1109/nca53618.2021.9685977
20. Horkoff, J., Lindman, J., Hammouda, I., & Knauss, E. (2019). **Strategic API Analysis and Planning: APIS Technical Report**. *arXiv preprint arXiv:1911.01235*. https://www.semanticscholar.org/paper/195eaa5ab0659d8b0bcf230e606c1c6395779195
21. Nieto, A., Zhao, Y., Lhoták, O., Chang, A., & Pu, J. (2019). **Scala with Explicit Nulls**. *Leibniz International Proceedings in Informatics (LIPIcs)*, *166*(ECOOP 2020), 25:1–25:28. https://doi.org/10.4230/LIPIcs.ECOOP.2020.25