JOURNAL OF INTERNATIONAL CRISIS AND RISK COMMUNICATION RESEARCH
ISSN: 2576-0017
2026, VOL 9, NO 2

Real-Time Data Integration in Hybrid Cloud Environments: Leveraging Striim and GCP for Enterprise Transformation

Naga Malleswara Babu Velpuri

Independent Researcher, USA

Abstract

Hybrid cloud environments—spanning on-premises systems and public cloud services—introduce complexity in data integration, consistency, and governance. Enterprises pursuing modernization must deliver real-time data to both operational applications and analytical platforms without sacrificing reliability or compliance. This article presents an end-to-end architecture for low-latency, fault-tolerant integration using change data capture technologies, streaming backbones, and GCP-native processing and serving. Source capture strategies minimize disruption by reading database logs and applying in-flight transformation, filtering, enrichment, and routing logic in streaming applications. Transport patterns connect on-premises and cloud messaging systems using connectors or custom relays, taking into account how to split data for efficiency, maintain order for consistency, set rules for data storage and replay, and ensure Processing uses cloud-based stream processing with features that manage event timing, group data into windows, and remove duplicates, while schema registries and compatibility rules protect users from changes. Serving combines multiple database technologies for near-real-time analytics, globally consistent transactions, and high-performance relational needs. Operational excellence manifests through observability with standardized dashboards for lag, throughput, error taxonomies, dead-letter queues, and replay windows, along with alerting for schema drift and automated runbooks for recovery. Reliability design includes checkpointing, idempotent sink writes, backpressure controls, and rolling upgrades to ensure continuity during maintenance. A multi-domain case study demonstrates batch windows collapsing from tens of minutes to seconds, enabling fresher dashboards and real-time downstream decisions. The findings demonstrate that real-time integration transcends mere transport problems, representing a holistic architectural and operational capability requiring governance, standards, and disciplined execution. This architecture and set of practices equip enterprises to accelerate modernization, minimize risk, and deliver reliable real-time data products across hybrid landscapes, turning integration from a bottleneck into a strategic advantage.

Keywords: Hybrid Cloud Integration, Change Data Capture, Real-Time Data Streaming, Google Cloud Platform, Enterprise Architecture.

1. Introduction and Motivation

1.1 The Hybrid Cloud Integration Challenge

Contemporary enterprises navigate increasingly intricate technological ecosystems where legacy on-premises infrastructure coexists alongside cloud-native applications and services. This architectural duality generates substantial obstacles for data integration, as organizations pursue data consistency, regulatory compliance adherence, and timely information delivery across disparate computational environments [1]. Traditional batch-oriented integration methodologies, previously adequate for business intelligence and reporting functions, demonstrate insufficient capabilities for addressing current operational agility and competitive responsiveness requirements. The multiplication of data sources, encompassing mainframe databases, relational database management systems, and cloud-native storage solutions, amplifies integration difficulties substantially. Organizations simultaneously manage multiple protocols, data formats, and security perimeters while attempting to maintain unified enterprise data visibility. The hybrid cloud model delivers flexibility and scalability advantages yet introduces latency concerns, network dependency vulnerabilities, and operational intricacies requiring resolution through intentional architectural selections and sound engineering methodologies [2].

1.2 From Batch to Real-Time: Enterprise Imperatives

The evolution from batch to real-time data integration signifies a fundamental transformation in enterprise data architecture principles. Batch processing, distinguished by scheduled extracts and periodic loads, establishes temporal discontinuities between operational events and their availability for analysis or downstream processing. These discontinuities translate into postponed business decisions, diminished operational efficiency, and forfeited opportunities for timely intervention across organizational functions. Real-time integration eliminates or substantially reduces these delays, permitting organizations to address business events during their occurrence rather than hours or days subsequently. The business value of real-time data manifests across multiple dimensions, including enhanced customer experience through immediate personalization, amplified operational efficiency via dynamic resource allocation, reduced fraud exposure through instantaneous anomaly detection, and optimized supply chain management through continuous visibility into inventory and logistics operations. The technical requirements for achieving real-time integration extend beyond merely increasing processing frequency substantially. Organizations must implement event-driven architectures, adopt streaming technologies, establish robust monitoring and alerting systems, and develop operational practices ensuring continuous availability and data quality across all systems. The economic justification for real-time integration increasingly favors immediate action, as the cost of cloud-native streaming technologies continues declining while the business value of timely data continues increasing across industries.

1.3 Research Objectives and Scope

This research establishes a comprehensive framework for implementing real-time data integration across hybrid cloud environments with a specific focus on changing data capture technologies, streaming infrastructure, and cloud platform capabilities comprehensively. The primary objective examines Ways for businesses to build complete data pipelines that capture updates from traditional databases, move those updates safely over networks, change and improve the data as needed, and deliver it quickly to both everyday users and analysts. The scope encompasses integration patterns for extracting changes from source systems without significantly impacting production workloads, streaming backbone design for reliable transport across hybrid environments, stream processing patterns for transformation and enrichment operations, and serving layer architectures for supporting diverse consumption patterns effectively. The research focuses on operational issues like observability, reliability, schema evolution, and collaboration within organizations, which are all very important for successful deployment in production environments across businesses. The framework presented synthesizes real-world implementation experience while incorporating industry best practices and emerging patterns in cloud-native data engineering disciplines.

1.4 Paper Organization

This paper systematically looks at real-time data integration architecture, starting with basic changes in data capture technologies and ending with best practices for operations and detailed case study insights. The second section explores architecture foundations, including changing data capture technologies and their application to traditional database systems extensively. The third section addresses hybrid transport and streaming infrastructure, examining interconnections between on-premises and cloud messaging systems

thoroughly. The fourth section goes into great detail about how cloud platforms can process and serve data, including stream processing patterns and database serving options. The fifth section presents operational excellence practices, reliability patterns, and a multi-domain case study demonstrating the practical application of the architecture effectively. The sixth section synthesizes key findings and outlines implications for enterprise modernization initiatives strategically.

2. Architecture Foundations: Source Capture and Change Data Capture

2.1 CDC Technologies: Striim and Oracle GoldenGate

Change data capture technologies establish the foundation of real-time integration architectures by facilitating efficient extraction of data modifications from source systems without imposing significant prohibitive overhead on operational workloads [3]. These technologies function by reading database transaction logs or redo logs rather than repeatedly querying tables, thereby circumventing the performance impact and resource contention associated with traditional extract-transform-load approaches substantially. Striim provides a complete platform for change data capture that works with various database systems like Oracle, DB2, SQL Server, and PostgreSQL, allowing for processing of data as it moves, which includes changing, filtering, and improving the data before it goes to other systems. Oracle GoldenGate constitutes another enterprise-grade solution particularly well-suited for Oracle database environments and mainframe systems, providing heterogeneous replication capabilities with minimal latency and robust conflict detection and resolution mechanisms comprehensively [4]. The selection between these technologies depends on factors including source database types, existing infrastructure investments, required transformation complexity, operational expertise availability, and licensing considerations strategically. Both platforms support initial load capabilities for bootstrapping new integrations alongside continuous replication for ongoing change propagation operations.

Table 1: Comparison of CDC Technologies [3, 4]

Feature	StreamStriim	Oracle GoldenGate
Supported Source Databases	Oracle, DB2, SQL Server, PostgreSQL, MySQL, MongoDB	Oracle, DB2, SQL Server, MySQL, Teradata
In-Flight Processing	Native transformation, filtering, and enrichment capabilities	Limited transformation, primarily replication-focused
Deployment Model	Cloud-native, on-premises, hybrid deployment options	Primarily on-premises, cloud extensions are available
Log Reading Method	Non-intrusive log-based capture	Non-intrusive log-based capture with supplemental logging
Initial Load Support	Automated initial load with continuous replication	Automated initial load with bidirectional replication
Conflict Resolution	Automated conflict detection and resolution policies	Advanced conflict detection with manual resolution options
Latency	Sub-second to low-second latency is typical	Sub-second to low-second latency is typical
Learning Curve	Moderate, requires streaming architecture knowledge	Steep, requires deep database administration expertise
Licensing Model	Subscription-based, consumption pricing	Perpetual license or subscription-based
Best Suited For	Multi-database environments, cloud migration projects	Oracle-centric environments, mainframe integration

2.2 Log-Based Capture from DB2 and Oracle Systems

Log-based change data capture reads database transaction logs to identify and extract committed changes without querying application tables or placing triggers on source systems directly. For DB2 systems, particularly DB2 for z/OS environments common in enterprise mainframe installations, log-based capture necessitates proper configuration of logging parameters and appropriate permissions for accessing log datasets securely. The capture process reads DB2 log records systematically, interprets the internal format of these records accurately, reconstructs the logical operations performed completely, and produces change events suitable for downstream consumption effectively. Oracle database environments similarly support log-based capture through supplemental logging mechanisms, which augment standard redo logs with additional column information necessary to facilitate and or complete change reconstruction accurately. The configuration of supplemental logging represents a critical implementation decision strategically, as minimal supplemental logging reduces overhead yet may prove insufficient for certain integration patterns, while full supplemental logging provides complete before-and-after images at the cost of increased log volume and storage requirements substantially. Log-based capture delivers several advantages over alternative approaches including minimal source system impact, capture of all committed changes regardless of application logic, preservation of transaction boundaries for consistency maintenance, and ability to capture deletes that query-based approaches often miss entirely. Proper implementation necessitates careful attention to log retention policies, ensuring that logs remain available sufficiently long for capture processes to read them, even during system outages or maintenance windows comprehensively.

2.3 In-Flight Transformation, Filtering, and Enrichment

In-flight processing capabilities facilitate transformation, filtering, and enrichment of change events during the capture phase before data reaches target systems or streaming infrastructure components. This processing paradigm offers significant advantages over post-capture transformation, including reduced network bandwidth consumption by filtering unnecessary events early, decreased storage requirements by eliminating irrelevant data before persistence operations, improved downstream processing performance by delivering pre-transformed data, and simplified target system logic by centralizing common transformations effectively. Transformation operations include data type conversions, field renaming operations, structural reshaping activities, and computed field derivation processes. Filtering operations eliminate events based on business rules, such as excluding test data, limiting changes to specific tables or schemas, dropping no-operation events containing no substantive changes, or applying time-based filters to ignore stale data comprehensively. Enrichment operations augment change events with additional context, including reference data lookups, timestamp standardization processes, identifier generation mechanisms, or metadata injection procedures. The implementation of in-flight processing necessitates careful consideration of processing overhead implications, as excessive transformation complexity can introduce latency and reduce throughput substantially. Monitoring of processing performance becomes essential to ensure that in-flight operations do not become bottlenecks in the overall integration pipeline architecture.

2.4 Minimizing Source System Disruption

Minimizing impact on source operational systems represents a paramount concern in changing data capture implementation, as production workloads must maintain performance and availability regardless of integration activities. Log-based capture inherently reduces source system impact compared to query-based approaches by avoiding table scans and eliminating the need for triggers or application modifications. Additional strategies for minimizing disruption include implementing capture processes on standby or read-replica databases rather than primary production instances, scheduling initial load operations during maintenance windows or low-activity periods, configuring appropriate batch sizes and commit intervals to balance throughput with resource consumption, and establishing connection pooling and rate limiting to prevent resource exhaustion scenarios. Network bandwidth considerations prove particularly significant when capturing from remote data centers or regions geographically, as excessive data transfer can saturate network links and impact other applications sharing the same infrastructure resources. Monitoring source system metrics, including CPU utilization, memory consumption patterns, log generation rates, and

replication lag, provides early warning of potential impact issues before they affect production operations critically. Establishing clear operational procedures for pausing or throttling capture processes during critical business periods or system maintenance ensures that integration activities can be temporarily suspended without manual intervention when source system protection becomes necessary immediately.

3. Hybrid Transport and Streaming Infrastructure

3.1 Kafka for On-Premises Event Backbone

Apache Kafka serves as the de facto standard for event streaming infrastructure in on-premises and hybrid cloud environments, providing distributed, fault-tolerant, high-throughput message brokering capabilities supporting both real-time streaming and historical replay patterns effectively [5]. The architecture centers on topics organizing events into ordered sequences, partitions enabling parallelism and scalability, and consumer groups facilitating load balancing across multiple consumers efficiently. The platform's durability guarantees, achieved through configurable replication across multiple brokers, ensure that events remain available even during broker failures or maintenance activities. Performance characteristics, including sub-millisecond latency for producer writes, high throughput measured in millions of messages per second, and efficient storage through log compaction, make it suitable for demanding enterprise integration scenarios comprehensively. On-premises deployments necessitate careful capacity planning, considering factors such as message volume, retention requirements, replication factor specifications, and anticipated consumer patterns strategically [6]. Operational considerations include broker configuration for optimal performance, topic design aligning partitioning strategies with parallelism requirements, monitoring infrastructure for tracking lag and throughput metrics, and backup and disaster recovery procedures ensuring data protection comprehensively.

3.2 Google Cloud Pub/Sub for Cloud-Native Messaging

Google Cloud Pub/Sub provides a fully managed, serverless messaging service designed for cloud-native applications requiring reliable, scalable message delivery without infrastructure management overhead substantially. The service abstracts away broker management, capacity planning, and scaling concerns, allowing development teams to focus on application logic rather than infrastructure operations exclusively. Support for both push and pull subscription models enables flexible integration patterns accommodating diverse consumer requirements including webhook delivery for serverless functions, batch pull for efficient high-throughput processing, and streaming pull for continuous real-time consumption effectively. The platform's global availability and automatic regional replication provide inherent disaster recovery capabilities without explicit configuration requirements. Message ordering guarantees, achieved through ordering keys, ensure that related events are processed in sequence, even across distributed consumers, consistently. Dead-letter topics capture messages repeatedly failing processing, preventing poison messages from blocking consumer progress unnecessarily. Integration with other Google Cloud services, including Dataflow for stream processing, Cloud Functions for event-driven computing, and BigQuery for direct streaming ingestion, occurs seamlessly throughout operations. The pricing model based on message volume and retention aligns costs with actual usage, making it economically attractive for variable workloads across industries.

Table 2: Kafka and Pub/Sub Feature Comparison [5, 6]

Feature	Apache Kafka (On-Premises)	Google Cloud Pub/Sub
Management Model	Self-managed, requires infrastructure provisioning	Fully managed, serverless
Scalability	Manual scaling, cluster expansion required	Automatic scaling, no infrastructure limits

Message Ordering	Guaranteed within partitions only	Guaranteed with ordering keys across subscribers
Retention Period	Configurable, days to weeks, typical	Configurable, up to seven days maximum
Delivery Semantics	At least once, exactly once with transactions	At-least-once delivery guarantee
Subscription Models	Pull-based consumption via consumer groups	Push and pull subscription models
Geographic Replication	Manual setup via MirrorMaker or Replicator	Automatic multi-region replication
Message Replay	Full replay capability within retention window	Limited replay based on subscription configuration
Dead-Letter Queue	Requires custom implementation	Native dead-letter topic support
Cost Model	Infrastructure costs plus operational overhead	Pay-per-use based on message volume
Integration Ecosystem	Extensive connector ecosystem via Kafka Connect	Native integration with Google Cloud services
Operational Complexity	High, requires specialized expertise	Low, minimal operational burden

3.3 Bridging Patterns: Connectors and Custom Relays

Bridging on-premises infrastructure with cloud-based messaging necessitates deliberate architectural patterns maintaining reliability, performance, and operational simplicity across the hybrid boundary effectively. Kafka Connect provides a framework for building scalable and reliable streaming data pipelines between external systems through reusable connectors comprehensively. A sink connector for Kafka Connect enables streaming data from topics into cloud topics with configurable transformations, error handling, and delivery semantics appropriately. This approach leverages distributed execution models, automatic offset management, and built-in monitoring capabilities to simplify operations substantially. Custom relay applications represent an alternative approach offering greater flexibility for complex transformation requirements, specialized error handling logic, or integration with proprietary systems uniquely. These applications consume from on-premises systems, apply business logic including filtering, enrichment, or aggregation, and publish to cloud systems with appropriate error handling and retry logic mechanisms. The choice between connector-based and custom relay approaches depends on factors including transformation complexity, operational expertise availability, existing infrastructure investments, and specific reliability requirements. Hybrid deployment models where relay components run in cloud environments with private connectivity to on-premises clusters via VPN or dedicated interconnect reduce latency and improve reliability substantially compared to internet-based connectivity.

3.4 Partitioning, Ordering, Retention, and Cross-Region Replication

Partitioning strategies directly impact parallelism, ordering guarantees, and consumer scalability in streaming architectures comprehensively. Partitions enable parallel processing by distributing topic data across multiple servers, with partition assignment to consumers within consumer groups providing effective load balancing. The choice of partition key determines how messages are distributed across partitions and consequently which messages maintain ordering guarantees, as ordering guarantees exist only within single partitions exclusively. Business entities such as customer identifiers, order numbers, or device identifiers often serve as partition keys to ensure that all events for a given entity are processed in order sequentially. Ordering keys provide similar functionality, maintaining order for messages with the same ordering key even across subscriber instances distributed. Retention policies determine how long messages remain

available for consumption, balancing storage costs against requirements for late-arriving consumers or historical replay needs. Retention can be time-based, size-based, or use log compaction to retain only the latest value for each key efficiently. Cross-region replication enhances availability and disaster recovery capabilities by maintaining copies of data in geographically distributed locations strategically. Replication capabilities between clusters provide synchronization, while multi-region topics offer built-in replication across regions automatically.

3.5 Schema Registries and Evolution Governance

Schema registries provide centralized repositories for event schemas, enabling producers and consumers to share and validate data structures while supporting controlled schema evolution over time systematically. The registry stores schemas in formats such as Apache Avro, Protocol Buffers, or JSON Schema, assigning version numbers to each schema iteration and enforcing compatibility rules governing how schemas can evolve appropriately. Forward compatibility ensures that consumers using old schemas can read data produced with new schemas, while backward compatibility ensures that consumers using new schemas can read data produced with old schemas effectively. Full compatibility requires both forward and backward compatibility, providing maximum flexibility yet constraining evolution options strategically. Transitive compatibility extends these rules across all historical versions rather than just adjacent versions comprehensively. Schema evolution governance establishes processes and policies for proposing, reviewing, approving, and deploying schema changes, ensuring that modifications do not break existing consumers or compromise data quality substantially. Common evolution patterns include adding optional fields, maintaining backward compatibility, deprecating fields with documented timelines for removal, and using union types or polymorphic structures for representing multiple message variants effectively. Integration between schema registries and streaming platforms enables automatic validation of messages against registered schemas during production and consumption, rejecting malformed messages before they propagate through the system entirely.

4. Processing and Serving in Google Cloud Platform

4.1 Dataflow: Stateful Stream Processing and Event-Time Windowing

Google Cloud Dataflow provides a fully managed service for executing Apache Beam pipelines, supporting both batch and streaming data processing with unified programming model abstractions, simplifying development and operational management substantially [7]. Stateful processing capabilities enable sophisticated stream processing patterns, including aggregations, joins, and complex event processing, maintaining state across multiple events effectively. State management abstractions such as ValueState for storing single values, BagState for accumulating multiple values, and MapState for storing key-value mappings provide building blocks for implementing business logic comprehensively. Event-time processing, as opposed to processing-time approaches, enables correct handling of late-arriving data and out-of-order events by using timestamps embedded in events themselves rather than arrival times exclusively. Windowing operations group events into finite collections based on time boundaries, enabling aggregations and transformations over bounded subsets of infinite streams effectively. Fixed windows divide time into regular intervals, sliding windows create overlapping time ranges, session windows group events based on activity gaps, and custom windows implement arbitrary grouping logic appropriately [8]. Triggers control when window results are emitted, balancing latency against completeness by specifying conditions such as watermark progression, element count, or processing time delays strategically. Watermarks represent system estimates of event-time progress, indicating that all events with timestamps before the watermark have likely arrived and enabling downstream processing to advance confidently throughout operations.

4.2 Deduplication, Backpressure, and Fault Tolerance

Deduplication mechanisms eliminate duplicate events arising from various sources, including at-least-once delivery semantics, producer retries, or source system behavior patterns. Stateful deduplication maintains records of previously processed event identifiers within bounded time windows, comparing incoming event identifiers against stored records to detect and discard duplicates effectively. The implementation must

consider memory constraints, as storing unlimited identifiers proves impractical, necessitating strategies such as windowed deduplication, maintaining state only for recent time periods, or probabilistic data structures like Bloom filters, trading perfect accuracy for memory efficiency appropriately. Backpressure handling ensures that downstream processing bottlenecks do not cause upstream system failures by implementing flow control mechanisms that slow or pause data production when consumers cannot keep pace adequately. Automatic backpressure through execution engines throttles sources when downstream stages accumulate excessive pending work substantially. Fault tolerance mechanisms enable pipelines to recover from failures without data loss or duplicate processing occurrences. Fault tolerance achievement occurs through checkpointing, where pipeline state periodically persists to durable storage, enabling immediate recovery to the most recent checkpoint upon failure. Exactly-once processing semantics combine idempotent operations, transactional writes, and checkpoint coordination to ensure that each input event affects output exactly once despite failures and retries comprehensively.

4.3 BigQuery for Near-Real-Time Analytics

BigQuery provides a serverless, highly scalable data warehouse optimized for analytical queries over petabyte-scale datasets with support for both batch loading and streaming ingestion patterns. Streaming inserts enable real-time data availability for analysis, with ingested records becoming immediately queryable without explicit load operations or index maintenance requirements. The streaming API accepts individual records or small batches with sub-second ingestion latency, making the platform suitable for use cases requiring fresh data for dashboards, alerts, or operational analytics comprehensively. Materialized views provide pre-computed query results automatically refreshing as underlying data changes, accelerating common analytical queries by eliminating repeated computation overhead. Partitioned tables organize data into segments based on ingestion time or column values, enabling query optimization through partition pruning, scanning only relevant partitions rather than entire tables efficiently. Clustering further organizes data within partitions based on column values, co-locating related rows to improve query performance substantially. The separation of storage and compute enables Each layer can be scaled independently, which lets analytical tasks use any amount of computing power without impacting data storage or needing to move data around. Integration through native connectors simplifies pipeline implementation, with built-in support for batching, error handling, and schema management comprehensively. The streaming buffer architecture acts as a temporary space for new data before it is combined into columnar storage, ensuring good performance for data input while also making sure that queries run

4.4 Spanner for Globally Consistent Transactions

Cloud Spanner delivers globally distributed, horizontally scalable relational database capabilities with strong consistency guarantees and SQL query support, combining traditional database semantics with cloud-scale infrastructure effectively. The architecture employs synchronized clocks across distributed replicas using TrueTime, enabling external consistency and ensuring transaction ordering matches real-time ordering despite substantial geographic distribution. This consistency model supports use cases requiring strict correctness guarantees, such as financial transactions, inventory management, or distributed workflows where eventual consistency proves insufficient entirely. Horizontal scalability through automatic sharding distributes data and query load across multiple nodes while maintaining relational abstractions and transactional guarantees comprehensively. The interleaved table design co-locates parent and child rows on the same split, optimizing queries accessing related entities while maintaining independent scalability appropriately. Multi-region configurations provide high availability and disaster recovery by maintaining synchronized replicas across geographic regions with automatic failover capabilities. Change streams enable the extraction of committed changes in near-real-time, supporting downstream integration patterns such as cache invalidation, search index updates, or event-driven processing operations. The integration with streaming architectures enables hybrid patterns where operational transactions execute against one system while analytical queries run against another, with change streams populating the analytical store in near-real-time continuously.

4.5 AlloyDB for High-Performance Relational Workloads

AlloyDB represents a fully managed PostgreSQL-compatible database service optimized for demanding transactional and analytical workloads requiring high performance, availability, and PostgreSQL

compatibility comprehensively. The architecture separates compute and storage layers, similar to other cloud-native databases, enabling independent scaling and cost optimization strategically. Performance enhancements include a columnar engine for analytical queries running alongside the traditional row-based engine, intelligent caching adapting to workload patterns, and optimized networking, reducing latency between application and database layers substantially. PostgreSQL compatibility ensures that existing applications, tools, and extensions designed for PostgreSQL work with minimal or no modification, simplifying migration from on-premises PostgreSQL deployments effectively. High availability configurations employ synchronous replication to standby instances within the same region, enabling automatic failover with minimal data loss and downtime occurrences. Read replicas distribute read traffic across multiple instances, improving scalability for read-heavy workloads while maintaining consistency through continuous replication from the primary instance. The database service integrates with identity and access management, networking, and monitoring infrastructure, providing enterprise-grade security, compliance, and operational capabilities comprehensively. Use cases particularly suited include modernization of on-premises PostgreSQL deployments requiring improved performance and availability, operational data stores for transaction-intensive applications, and hybrid analytical-transactional workloads benefiting from the integrated columnar engine effectively.

Table 3: GCP Processing and Serving Options [7, 8]

Service	Dataflow	BigQuery	Spanner	AlloyDB
Primary Use Case	Stream and batch processing	Analytics and data warehousing	Globally distributed transactions	High-performance relational workloads
Processing Model	Unified batch and streaming	SQL-based analytical queries	ACID transactions with SQL	ACID transactions with PostgreSQL compatibility
Scalability	Auto-scaling workers	Automatic serverless scaling	Horizontal sharding, multi-region	Compute-storage separation, read replicas
Consistency Model	Configurable per pipeline	Eventually consistent for streaming	Strong external consistency	Strong consistency with synchronous replication
Latency	Sub-second to seconds	Sub-second for streaming inserts	Single-digit milliseconds	Single-digit milliseconds
State Management	Stateful processing with multiple state types	Stateless query execution	Stateful transactional processing	Stateful transactional processing
Query Language	Apache Beam SDK (Java, Python, Go)	Standard SQL with extensions	Standard SQL with extensions	PostgreSQL-compatible SQL
Data Model	Event streams and collections	Columnar storage, partitioned tables	Relational with interleaved tables	Relational with row and columnar engines

Integration	Native Pub/Sub, BigQuery, Spanner connectors	Direct streaming API, Dataflow integration	Change streams for real-time extraction	Standard PostgreSQL connections and tools
Best For	Complex transformations, windowing, joins	Near-real-time analytics, reporting	Mission-critical transactions, inventory	PostgreSQL migration, hybrid workloads

5. Operational Excellence, Reliability, and Multi-Domain Case Study

5.1 Observability: Dashboards, Metrics, and Alerting

Comprehensive observability establishes the foundation for reliable operation of real-time integration pipelines through systematic collection, analysis, and visualization of operational metrics combined with proactive alerting on anomalous conditions effectively [9]. Standardized dashboards provide unified visibility across pipeline components, displaying metrics including message throughput rates, processing latency percentiles, error counts by category, consumer lag measurements, and resource utilization for compute, memory, and network comprehensively. These dashboards serve multiple audiences, from operations teams monitoring real-time system health to platform engineers analyzing performance trends to business stakeholders tracking data freshness for critical domains strategically. Metric taxonomies organize measurements into logical hierarchies, distinguishing between infrastructure metrics such as CPU and memory utilization, platform metrics including message queue depths and replication lag, and business metrics like event counts by domain or processing completeness percentages appropriately. Alerting policies define conditions triggering notifications, balancing sensitivity to detect genuine issues against specificity to avoid alert fatigue from false positives. Severity levels distinguish between critical alerts requiring immediate response, warnings indicating degraded conditions potentially not yet impacting users, and informational alerts tracking important yet non-urgent conditions systematically. Alert routing directs notifications to appropriate teams or individuals based on component ownership, time of day, and escalation procedures comprehensively. Dead-letter queue monitoring identifies messages repeatedly failing processing, enabling investigation and remediation of systematic data quality issues or processing bugs substantially. Replay window tracking measures the time range of historical data available for reprocessing, ensuring sufficient retention for recovery scenarios appropriately.

5.2 Reliability Patterns: Checkpointing, Idempotency, DLQs, and Replay

Checkpointing mechanisms enable fault-tolerant stream processing by periodically persisting pipeline state to durable storage, allowing recovery to known consistent states following failures without data loss or duplicate processing occurrences [10]. The checkpoint interval balances recovery time objectives against checkpoint overhead, with more frequent checkpoints reducing potential data loss at the cost of increased storage operations and slight processing latency. Coordinated checkpointing across distributed pipeline stages ensures that the recovered state remains consistent across components, preventing scenarios where different stages recover to mismatched states inappropriately. Idempotent operations produce identical outcomes regardless of execution frequency with the same input, enabling safe retries without duplicate effects. Sink operations achieve idempotency through unique identifiers enabling deduplication, conditional writes succeeding only if previous conditions hold, or transactional mechanisms coordinating multiple operations atomically and effectively. Dead-letter queues capture messages failing processing after exhausting retry attempts, preventing poison messages from blocking pipeline progress while preserving failed messages for investigation and potential reprocessing after fixes deploy successfully. Replay capabilities enable reprocessing of historical data following bug fixes, schema changes, or business logic updates, requiring retention of source data or intermediate processing results for the desired replay window appropriately. Versioning strategies track pipeline code and configuration changes, enabling rollback to previous versions if new deployments introduce issues and supporting parallel execution of multiple pipeline versions during the gradual rollout of changes systematically.

Table 4: Reliability Patterns Implementation [10]

Reliability Pattern	Implementation Approach	Benefits	Considerations
Checkpointing	Periodic state persistence to cloud storage every thirty to sixty seconds	Enables recovery to a consistent state, Minimizes data loss during failures	Checkpoint frequency impacts performance. Storage costs increase with state size
Idempotent Writes	Unique event identifiers with upsert operations, Conditional writes based on version	Prevents duplicate processing effects, enables safe retries without side effects	Requires additional storage for tracking identifiers. May impact write performance
Dead-Letter Queues	Separate topic for failed messages after three to five retry attempts	Prevents pipeline blocking, Preserves failed data for investigation	Requires monitoring and manual intervention, Storage costs for retained failures
Replay Capability	Source data retention for seven to thirty days with timestamp-based replay	Enables reprocessing after bug fixes, supports schema migration scenarios	Increased storage requirements require coordination with downstream consumers
Backpressure Control	Dynamic rate limiting based on consumer capacity and queue depth	Prevents system overload, maintains stability during spikes	May introduce temporary delays. Requires careful tuning of thresholds
Circuit Breaking	Automatic suspension after consecutive failures with exponential backoff	Prevents cascading failures, allows systems time to recover	May cause temporary data gaps. Requires robust alerting and monitoring
Graceful Degradation	Fallback to cached data or simplified processing during partial failures	Maintains service availability and reduces the complete outage impact	May provide stale or incomplete data. Requires clear communication to consumers
Rolling Upgrades	Staged deployment with canary testing and gradual traffic shifting	Minimizes disruption during updates, enables quick rollback if issues arise	Requires version compatibility, Extends deployment duration

5.3 Latency Budgets, Autoscaling, and Performance Under Load

Latency budgets allocate acceptable delay across pipeline stages, establishing clear performance targets guiding architectural decisions and operational practices comprehensively. End-to-end latency requirements derive from business needs such as dashboard refresh intervals, operational alert timing, or downstream processing deadlines strategically. The total budget is distributed across source capture latency, network transfer time, stream processing duration, and target system write latency, with each component allocated a portion based on its architectural role and optimization opportunities appropriately. Autoscaling mechanisms adjust computational resources in response to workload variations, maintaining performance during traffic spikes while controlling costs during low-activity periods effectively. Dynamic provisioning of worker instances based on backlog size and processing rate scales up when pending work accumulates and scales down during idle periods systematically. Consumer groups achieve parallelism through partition

assignment, with consumer instance count matching or exceeding partition count to maximize throughput substantially. Adaptive batching optimizes throughput by accumulating multiple events into single operations when beneficial for downstream systems, while maintaining latency constraints by flushing accumulated batches after timeout thresholds are appropriately met. Performance testing validates that pipelines meet latency budgets under anticipated load, using realistic data volumes, event rates, and processing complexity to identify bottlenecks before production deployment comprehensively. Load testing incrementally increases traffic to determine maximum sustainable throughput, headroom for handling spikes, and degradation behavior beyond capacity limits effectively.

5.4 Case Study: Inventory, Pricing, and Promotions Domains

A multi-domain implementation spanning Inventory, Pricing, and Promotions domains demonstrates the practical application of a real-time integration architecture across interconnected business functions comprehensively. The Inventory domain captures stock level changes from warehouse management systems, distribution center databases, and point-of-sale terminals, streaming updates through the integration pipeline to maintain current availability information for online ordering systems, fulfillment optimization algorithms, and business intelligence dashboards effectively. Previously operating on overnight batch updates with staleness measured in hours, the real-time architecture reduced data latency to seconds, enabling accurate available-to-promise calculations and preventing overselling situations substantially. The Pricing domain synchronizes price changes from enterprise resource planning systems, competitive intelligence feeds, and dynamic pricing engines to customer-facing applications and downstream analytics appropriately. Real-time price updates eliminated the lag between pricing decisions and their reflection in customer experiences, improving responsiveness to competitive pressure and market conditions strategically. The Promotions domain integrates promotional rules, eligibility criteria, and redemption events from marketing automation platforms and loyalty systems to point-of-sale applications and customer engagement channels comprehensively. Immediate promotion activation replaced batch-based delays, enabling flash sales and time-sensitive campaigns with precise start times. The interconnected nature of these domains created dependencies where pricing calculations required current inventory levels and promotional rules needed awareness of inventory constraints, demonstrating the value of a unified real-time integration architecture spanning multiple business functions systematically.

5.5 Anti-Patterns and Remediation Strategies

Common anti-patterns undermine real-time integration initiatives, with recognition and remediation essential for successful implementation comprehensively. The SELECT-star analytics anti-pattern extracts entire tables rather than incremental changes, substantially overwhelming network bandwidth, processing capacity, and storage systems, while introducing substantial unnecessary latency. Remediation employs change data capture for incremental extraction, column projection to transfer only required fields, and predicate pushdown to filter data at the source effectively. Ungoverned schema changes break downstream consumers when producers modify message structures without coordination or compatibility checks. Remediation establishes schema registries with enforced compatibility rules, change approval processes requiring consumer notification, and versioning strategies supporting gradual migration systematically. Lack of data lineage obscures data provenance, transformation history, and quality issues, impeding debugging and compliance efforts significantly. Remediation implements metadata collection throughout pipelines, centralized cataloging of data assets and transformations, and automated documentation generation comprehensively. Insufficient testing leads to production issues, including data quality problems, performance bottlenecks, and operational failures. Remediation requires comprehensive test strategies, including unit tests for transformation logic, integration tests for component interactions, performance tests under realistic load, and chaos engineering for resilience validation. Missing operational runbooks leave teams unprepared for incidents, extending resolution time and increasing impact considerably. Remediation develops documented procedures for common scenarios, including pipeline restart, data replay, schema migration, and capacity adjustment appropriately.

5.6 Cross-Functional Collaboration and Data Contracts

Successful real-time integration necessitates collaboration across application development, data engineering, platform operations, and business teams, with clear responsibilities, communication channels,

and a shared understanding of requirements and constraints comprehensively. Data contracts formalize agreements between data producers and consumers, specifying schemas, the semantic meaning of fields, quality expectations, latency commitments, and evolution policies systematically. These contracts establish clear expectations, enabling independent development of producer and consumer systems while ensuring compatibility effectively. The contract negotiation process brings together stakeholders to discuss requirements, constraints, and tradeoffs, fostering shared understanding beyond formal specifications appropriately. Version management within contracts supports evolution while maintaining stability, using semantic versioning to communicate the nature and impact of changes strategically. Consumer opt-in mechanisms allow gradual adoption of new schema versions rather than forcing simultaneous updates across all consumers comprehensively. Compatibility testing validates that proposed changes maintain contracts, automatically checking that new producer schemas remain compatible with registered consumer schemas effectively. Communication protocols establish expectations for change notification, impact assessment, and migration coordination systematically. Regular reviews assess contract effectiveness and identify improvement opportunities continuously. The organizational change accompanying technical transformation necessitates attention to team structure, skill development, and cultural factors, moving from batch-oriented thinking toward event-driven mindsets and from project-based work toward product-oriented continuous improvement substantially.

Conclusion

This article presented a comprehensive architecture for real-time data integration in hybrid cloud environments, addressing the full lifecycle from source capture through transport, processing, and serving. The architecture uses change data capture technologies to efficiently pull data from old database systems, streaming infrastructure to reliably move data across different environments, cloud-native processing to transform and enhance the data, and various options for delivering the data for both operational and analytical use. Key architectural contributions include patterns for bridging on-premises and cloud messaging systems, approaches for maintaining ordering and consistency guarantees across distributed components, strategies for schema evolution governance that support the independent evolution of producers and consumers, and designs for operational observability that enable proactive monitoring and rapid incident response. Real-time integration changes from just a technical skill to a key advantage by allowing quicker decision-making, better customer experiences, improved efficiency, and a way to stand out from competitors, with financial benefits showing up as increased revenue, lower costs, reduced risks, and support for innovation. Enterprise modernization efforts are starting to see data integration as a core part of their strategy, not just an add-on, with real-time capabilities being crucial for meeting modernization goals, and the hybrid cloud model that many large companies use during their transition requires integration systems that connect old and new systems without needing to replace working systems too soon. The operational maturity required for reliable real-time integration often exceeds initial expectations, with organizations needing to develop capabilities in monitoring, incident response, capacity planning, and change management, that surpass requirements for batch systems. Several directions merit continued research and development attention as real-time integration technologies and practices mature, including machine learning applications within integration pipelines for anomaly detection and predictive scaling, standardization efforts around event formats and schema evolution rules, edge computing integration extending real-time processing closer to data sources, federated governance models balancing central coordination with domain autonomy, and sustainability considerations around energy consumption and environmental impact of data integration infrastructure as data volumes and processing requirements continue growing across industries worldwide.

References

[1] Gaurav Tuteja, et al., "Hybrid Cloud-Edge Computing for Real-Time IoT-Based Health Monitoring: Performance Evaluation and Optimization," in 2025 4th OPJU International Technology Conference (OTCON) on Smart Computing for Innovation and Advancement in Industry 5.0, 14 July 2025. Available: https://ieeexplore.ieee.org/document/11071023

[2] Vijaya Bhaskara Reddy Soperla, "Real-Time Data Integration in Hybrid Cloud Environments: Challenges and Solutions," International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 2025. Available: https://ijsrcseit.com/index.php/home/article/view/CSEIT251112306
[3] Oracle Corporation, "Enabling Change Capture," Oracle GoldenGate Documentation, Oracle GoldenGate Release 21.3. Available: https://docs.oracle.com/en/middleware/goldengate/core/21.3/ggcab/enabling-change-capture-db2-z-os.html
[4] Striim Community Team, "CDC Best Practices with a 'Read Once, Stream Anywhere' Pattern," Striim Community Knowledge Base, 2024. Available: https://community.striim.com/cool-content-33/cdc-best-practices-with-a-read-once-stream-anywhere-pattern-119
[5] Ahmed Twabi, "Real-Time Video Streaming on the Pub/Sub Architecture: Case of Apache Kafka," in 2024 IEEE 6th Symposium on Computers & Informatics (ISCI), 12 September 2024. Available: https://ieeexplore.ieee.org/document/10668190
[6] Software Patterns Lexicon Editorial Team, "Hybrid and Multi-Cloud Kafka Architectures: Strategies and Best Practices," Software Patterns Lexicon, 2023. Available: https://softwarepatternslexicon.com/kafka/cloud-deployments-and-managed-services/hybrid-and-multi-cloud-kafka-architectures/
[7] Ashok Gadi Parthi, et al., "Cloud-Native Change Data Capture: Real-Time Data Integration from Google Spanner to BigQuery," JETIR Research Journal, 2025. Available: https://www.jetir.org/papers/JETIR2505758.pdf
[8] Chitra Sabapathy Ranganathan, et al., "Ingestion of Google Cloud Platform Data using Dataflow," in 2023 7th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 26 October 2023. Available: https://ieeexplore.ieee.org/abstract/document/10290190
[9] Ummay Faseeha, et al., "Observability in Microservices: An In-Depth Exploration of Frameworks, Challenges, and Deployment Paradigms," IEEE Xplore, 17 April 2025. Available: https://ieeexplore.ieee.org/document/10967524
[10] Lakshmi Sai Krishna Kaushik Kakumanu, "Designing Scalable and Fault-Tolerant Streaming Architectures Using Event-Driven Microservices for High Availability," Journal of Applied and Advanced Research, 2025. Available: https://rjwave.org/jaafr/papers/JAAFR2507003.pdf