10.48047/jocaaa.2026.35.01.42

A Declarative Orchestration Model for Multi-Tenant Cloud-Native Advertising and Financial Workflows

Sujoy Datta Choudhury
Independent Researcher, USA

Abstract

Enterprise platforms serving advertising and financial services clients face a fundamental challenge: delivering customized workflows without maintaining separate codebases for each tenant. This article examines how declarative orchestration models address this tension by separating platform-defined workflow capabilities from tenant-specific configuration. The proposed article treats workflows as configurable products rather than custom projects, enabling hundreds of clients to operate on shared infrastructure while preserving their unique business logic, approval hierarchies, and integration requirements. The article draws from converging trends in multi-tenant SaaS architectures and cloudnative workflow specifications, demonstrating how platforms can define canonical workflows for processes like campaign management, trade settlement, and compliance automation while tenants control behavior through structured configuration parameters. Implementation patterns span serverless environments for high-throughput event processing and durable orchestration engines for longrunning, human-in-the-loop procedures. The article analyzes governance considerations, including schema evolution, testing strategies, observability requirements, and configuration validation frameworks, essential for operating these systems at scale. Case studies from advertising and financial domains illustrate practical applications, while a discussion of benefits and limitations guides when declarative approaches prove most valuable. Future directions explore AI-assisted configuration optimization, advanced isolation techniques, and industry standardization efforts that could accelerate adoption across enterprise platforms.

Keywords: Multi-tenant architecture, Declarative orchestration, Workflow orchestration, Cloudnative systems, SaaS configuration

Introduction

Enterprise software platforms face a persistent dilemma. On one hand, banks, asset managers, and advertising agencies demand customized workflows that reflect their unique business processes. On the other hand, platform providers cannot afford to maintain separate codebases for each client. This tension has historically forced organizations into uncomfortable compromises—either accepting rigid, one-size-fits-all solutions or drowning in a maintenance nightmare of client-specific forks.

The cloud-native movement offers a way forward. Rather than treating workflows as hardcoded application logic, modern platforms are shifting toward declarative models where business processes become configurable artifacts. This approach separates what the platform can do from how individual tenants choose to use it. A financial services firm might enable strict compliance checks and multilevel approvals, while an e-commerce advertiser prioritizes speed and automated decisioning—yet both operate on identical infrastructure.

This architectural pattern draws from two converging trends. Multi-tenant SaaS architectures have matured to the point where shared infrastructure no longer means sacrificed flexibility. Meanwhile, orchestration technologies have embraced declarative specifications that describe desired outcomes rather than procedural steps. Standards like the CNCF Serverless Workflow specification provide vendor-neutral languages for expressing complex, event-driven processes [6]. Paired with durable execution engines such as Temporal and Cadence, these specifications enable platforms to treat workflows as first-class configuration objects.

10.48047/jocaaa.2026.35.01.42

This paper examines how declarative orchestration models can address the customization challenge in advertising and financial systems. It proposes a clear separation between platform-defined canonical workflows and tenant-specific configuration, analyzes implementation strategies across different runtime environments, and discusses the governance complexities that emerge when workflows become shared products.

2. Background and Related Work

2.1 Multi-Tenant Architecture Principles

Multi-tenant systems consolidate infrastructure while maintaining strict boundaries between customer data. Modern architectures achieve this through database-level isolation, encrypted storage partitions, and identity-aware access controls [1][2]. Unlike traditional hosting models that provision separate application instances per client, true multi-tenancy shares compute, storage, and networking resources across all tenants. This reduces operational overhead and enables uniform security patches and feature releases.

The key advantage lies in configurability rather than customization. Tenants adjust behavior through parameters, feature flags, and policy settings instead of requesting code modifications [3]. This approach has proven particularly valuable in product lifecycle management and other enterprise domains where regulatory requirements and business processes vary significantly across organizations.

Dimension	Platform Concern	Tenant Configuration Example
Workflow graph	Steps, retries, timeouts, compensation logic	Enable or disable optional validation checks
Data contracts	Event and state schemas	Custom fields, mapping to tenant systems
Policies and routing	Generic escalation and backoff strategies	Custom risk tiers, approval hierarchies
Isolation and limits	Multi-tenant throttling, fairness mechanisms	Per-tenant quotas, regional data residency rules

Table 1: Platform vs. Tenant Configuration Separation [1, 2]

2.2 Declarative Infrastructure and Application Patterns

Infrastructure-as-Code has transformed how teams manage cloud resources [4][5]. Rather than executing sequential deployment scripts, operators declare target configurations and let automated controllers reconcile actual state with desired state. This convergence loop pattern reduces drift and makes infrastructure changes auditable and repeatable.

The same principles now extend beyond infrastructure. Policy engines implement declarative rules for resource governance, security compliance, and cost management [9]. Controllers continuously evaluate policies against live environments and automatically remediate violations. This shift from imperative to declarative models reflects a broader industry movement toward treating operations as data problems.

2.3 Cloud-Native Workflow Specifications

Standardized workflow languages prevent vendor lock-in while enabling portability across execution platforms. The Serverless Workflow specification defines event-driven processes using JSON and YAML, making workflows readable to both humans and machines [7]. Orchestration tools like Kestra extend this concept to data pipelines, where declarative definitions replace brittle scripting [8]. These specifications abstract away runtime implementation details, letting teams focus on business logic.

10.48047/jocaaa.2026.35.01.42

2.4 Workflow Orchestration Engines

Modern orchestration engines address different points on the code-versus-configuration spectrum. Temporal allows developers to write workflows in familiar programming languages while the platform handles state persistence, automatic retries, and failure recovery [10][11]. Its architecture ensures that long-running processes survive infrastructure failures and code deployments [12]. Cadence and Conductor target similar microservices orchestration challenges with varying approaches to scalability and developer experience [13][14]. All three engines share core capabilities: durable execution guarantees, distributed transaction coordination, and visibility into workflow progress.

Domain	Workflow Type	Key Characteristics	Execution Pattern
Advertising	Impression processing	High volume, low latency, fraud detection	Event-driven, serverless
Advertising	Budget pacing	Real-time monitoring, bid adjustment	Short-lived control loops
Advertising	Multi-channel attribution	Cross-channel aggregation, model application	Batch processing
Financial	Trade capture	Multi-channel ingestion, validation, and routing	Transaction-oriented
Financial	Settlement	Multi-party coordination, strict consistency	Long-running, durable
Financial	Compliance review	Pattern detection, human-in-the-loop	Days-to-weeks duration

Table 2: Workflow Taxonomy Comparison [7-11]

3. Proposed Model: Declarative Multi-Tenant Orchestration

3.1 Architectural Principles

The core insight behind declarative multi-tenant orchestration is simple: workflows should be products, not projects. Rather than building custom process engines for each tenant, platforms define a catalog of canonical workflows that serve as configurable building blocks. This separation between platform capabilities and tenant preferences creates a sustainable architecture where new features benefit all customers simultaneously.

Platform teams own the workflow graph—the sequence of steps, retry policies, timeout behaviors, and compensation logic that ensure reliable execution. Tenants control how these workflows behave in practice through configuration parameters, feature toggles, and policy definitions. This boundary lets engineers focus on hardening core orchestration logic while business teams adapt workflows to diverse requirements without writing code.

3.2 Workflow Taxonomy for Advertising and Financial Systems

Advertising and financial platforms share surprisingly similar orchestration challenges despite serving different domains. Both manage high-volume transactional flows, enforce complex business rules, and require audit trails for regulatory compliance.

In advertising systems, impression processing workflows ingest billions of events daily, applying fraud detection, attribution modeling, and billing calculations. Budget pacing workflows monitor campaign spend in near real-time, adjusting bid strategies to distribute budgets evenly across flight dates. These processes demand low latency and horizontal scalability.

Financial workflows follow different rhythms. Trade capture workflows receive orders from multiple channels, enrich them with reference data, validate against risk limits, and route them for execution

10.48047/jocaaa.2026.35.01.42

[15]. Settlement workflows coordinate multi-party transactions across payment rails with strict consistency requirements. Compliance workflows flag suspicious patterns and orchestrate human review processes that can span days or weeks.

Despite these differences, both domains benefit from declarative orchestration that separates business logic from execution mechanics.

3.3 Configuration Dimensions

The proposed model organizes tenant configuration across four dimensions. Workflow graphs remain largely platform-controlled, though tenants can enable or disable optional validation steps, approval gates, and notification hooks. Data contracts define the schemas for events and state objects, with provisions for tenant-specific custom fields that extend base schemas without breaking platform invariants.

Policies and routing rules provide the richest customization surface. A generic escalation strategy might define three severity tiers, but tenants specify dollar thresholds, approval chains, and SLA targets for each tier. Isolation and limit settings govern resource consumption, ensuring that one tenant's workload cannot starve others.

3.4 Tenant Configuration Model

Configuration manifests capture all tenant-specific behavior in structured documents. A campaign management workflow might include optional fraud checks that conservative advertisers enable while performance-focused clients skip. Field mappings connect platform schemas to tenant systems—one client's "customer_id" becomes another's "account_reference."

Approval hierarchies reflect organizational structures. Some financial institutions require dual authorization for trades above certain thresholds; others delegate more authority to individual traders. These policies live in configuration, not code, making them auditable and easier to update as organizations evolve.

4. Implementation Patterns

4.1 Runtime Selection Strategy

Not all workflows suit the same execution environment. Short-lived, event-driven processes that complete in seconds fit naturally on serverless platforms where functions scale automatically and costs align with usage. Longer processes involving human approvals or external system callbacks need durable orchestration engines like Temporal or Cadence that preserve state across failures and restarts [11][13].

Sophisticated platforms often adopt hybrid architectures. Real-time bidding workflows run on serverless infrastructure for maximum throughput, while dispute resolution workflows use Temporal to reliably shepherd cases through multi-week investigation processes.

4.2 Data Isolation and Identity

Multi-tenant orchestration platforms must enforce strict data boundaries. Each workflow execution operates within a tenant context that governs database access, API credentials, and encryption keys [15]. Messaging systems employ tenant-aware routing to prevent cross-contamination. Identity providers map workflow actions to human users, enabling detailed audit logs that satisfy regulatory requirements.

4.3 Domain-Specific Building Blocks

Declarative orchestration gains power from domain-specific abstractions. Advertising platforms expose pacing rules as configurable components rather than forcing tenants to implement control loops manually. Financial platforms provide pre-built regulatory reporting modules that adapt to different jurisdictions through parameter selection. These building blocks reduce time-to-value for new tenants while concentrating platform investment in high-leverage capabilities.

10.48047/jocaaa.2026.35.01.42

Engine	Approach	Best Use Case	Key Strength
Temporal	Code-first with durable execution	Long-running, complex workflows with retries	State persistence across failures
Cadence	Microservices orchestration	Distributed transactions, saga patterns	Proven scalability at Uber
Conductor	Configuration-driven	Event-driven microservices coordination	Visual workflow designer
Serverless Workflow	Declarative specification	Event-driven, cloud-agnostic processes	Vendor neutrality, portability

Table 3: Workflow Orchestration Engine Comparison [10, 11]

5. Governance and Operational Considerations

5.1 Schema Evolution and Versioning

Declarative workflows create a dependency web that demands careful change management. When a platform team modifies a workflow schema or adds new mandatory fields, hundreds of tenant configurations may need updates. The challenge intensifies because tenants operate on different release schedules—some upgrade eagerly while others lag months behind.

Versioned workflow definitions provide one solution. Each workflow carries a semantic version number, and the platform maintains multiple versions simultaneously. Tenants reference specific versions in their configurations, giving them control over when to adopt changes. This approach prevents breaking updates from cascading across the entire tenant base unexpectedly.

Migration strategies vary based on change severity. Additive changes like new optional parameters require minimal coordination. Breaking changes that remove fields or alter behavior require deprecation cycles, automated migration tools, and sometimes direct tenant engagement. Platform teams must balance innovation velocity against stability commitments.

5.2 Testing and Validation

Multi-tenant platforms face an exponential testing problem. With dozens of configurable workflow parameters and hundreds of tenants, exhaustively testing every permutation becomes impractical. Smart testing strategies focus on high-risk configurations and common patterns rather than attempting complete coverage.

Synthetic tenant profiles help manage this complexity. Test suites create representative configurations spanning conservative, moderate, and aggressive parameter choices. Integration tests exercise workflows with these profiles against realistic data volumes. Chaos engineering techniques deliberately inject failures—network partitions, database slowdowns, message delivery delays—to verify that retry logic and compensation mechanisms work across tenant configurations [9].

Contract testing adds another layer of protection. When workflows interact with external systems, contract tests validate that tenant-specific integrations respect agreed-upon interfaces. This prevents configuration errors from causing runtime failures in production.

5.3 Observability and Monitoring

Operating multi-tenant orchestration platforms requires visibility at multiple levels. Platform-level dashboards track aggregate metrics: workflow completion rates, queue depths, error budgets, and resource utilization. These indicators reveal systemic issues affecting all tenants.

Per-tenant observability matters equally. Individual dashboards show each tenant their workflow performance, SLA compliance, and cost attribution. This transparency builds trust and helps tenants

10.48047/jocaaa.2026.35.01.42

optimize their configurations. Distributed tracing connects workflow executions across microservices, making it possible to diagnose latency issues that span multiple systems [11].

Noisy neighbor detection prevents resource monopolization. Monitoring systems flag tenants whose workloads consume disproportionate compute, memory, or network bandwidth. Fairness algorithms can then throttle aggressive tenants or trigger capacity scaling to protect platform stability. These mechanisms preserve the multi-tenant promise that all customers receive consistent service quality.

5.4 Governance Framework

Configuration freedom requires guardrails. Centralized approval processes review new tenant workflows before production deployment, checking for security vulnerabilities, resource abuse potential, and compliance violations. Automated validation tools catch common mistakes: infinite retry loops, missing timeout values, or references to non-existent services.

Lint rules codify best practices. They might enforce naming conventions, require documentation for custom fields, or flag workflows that lack appropriate monitoring hooks. These rules reduce configuration drift and make tenant setups more maintainable over time.

6. Discussion

6.1 Benefits of the Declarative Approach

Declarative orchestration transforms workflows from scattered code into managed assets. This consolidation eliminates redundant implementations of common patterns—pacing logic, approval chains, retry strategies—that previously existed in slightly different forms across codebases. Platform teams maintain a single authoritative version that all tenants consume.

Tenant onboarding accelerates dramatically. New customers configure existing workflows rather than waiting for custom development. What once took months of engineering effort now requires days or weeks of configuration work. This velocity advantage compounds as platforms mature and workflow libraries expand.

Auditability improves because configuration files create explicit records of business rules and processing logic. Regulators can inspect workflow definitions to understand how transactions flow through systems. Version control systems track every configuration change with timestamps and authorship data [8].

6.2 Challenges and Limitations

Success shifts complexity rather than eliminating it. Configuration management becomes a specialized discipline requiring dedicated tooling and expertise. Teams need infrastructure to validate configurations, manage dependencies between workflows, and safely roll out changes across tenant populations.

Declarative languages introduce abstraction costs. Engineers accustomed to procedural code may struggle with declarative thinking. Debugging becomes more challenging when workflow definitions generate runtime behavior indirectly. Performance tuning can be difficult when abstraction layers obscure underlying execution details.

6.3 When Not to Use This Model

Declarative orchestration fits poorly when tenant requirements diverge wildly. If each customer needs fundamentally different workflow topologies, configuration grows so complex that procedural code becomes simpler. Platforms serving a handful of large enterprise clients may find that custom implementations per tenant remain more practical than building elaborate configuration systems.

Legacy constraints also matter. Systems built on older architectures may lack the infrastructure— event buses, state stores, identity systems—that multi-tenant orchestration requires. Retrofitting these capabilities can be more expensive than maintaining existing patterns [15].

10.48047/jocaaa.2026.35.01.42

7. Case Studies and Applications

7.1 Multi-Tenant Advertising Platform

Modern advertising platforms handle millions of campaign operations daily across diverse client portfolios. Declarative orchestration proves particularly valuable in managing campaign lifecycle workflows that span creation, activation, optimization, and reporting phases. Each advertiser brings different requirements—some demand aggressive bidding strategies while others prioritize brand safety and fraud prevention.

Real-time bidding orchestration represents one of the most demanding applications of multi-tenant workflows. When ad exchanges broadcast bid requests, platforms have milliseconds to evaluate inventory, check budget constraints, apply pacing rules, and submit competitive bids. Declarative workflows define these decision trees as configurable state machines where tenants specify bid multipliers, audience targeting criteria, and budget allocation strategies without touching platform code [7].

Multi-channel attribution workflows present different challenges. These processes aggregate conversion data across display, search, social, and offline channels, then apply attribution models to credit campaigns appropriately. Different advertisers prefer different models—last-click, linear, timedecay, or custom algorithmic approaches. Declarative orchestration lets the platform support all these variants through configuration while maintaining a single codebase for data collection and model execution [8].

Campaign reporting workflows demonstrate the operational benefits. Tenants configure which metrics matter, how frequently reports are generated, and where results are delivered. The platform handles scheduling, data aggregation, and delivery mechanics uniformly across all clients.

7.2 Financial Services Platform

Financial institutions operate under strict regulatory requirements that vary by jurisdiction and institution type. Trade processing workflows must capture orders from multiple channels, validate them against position limits and risk parameters, enrich them with reference data, and route them to appropriate execution venues. Each institution configures these workflows differently based on its risk appetite, regulatory obligations, and trading strategies [15].

Regulatory compliance automation showcases declarative orchestration's audit advantages. Know Your Customer workflows, anti-money laundering checks, and suspicious activity reporting follow institution-specific thresholds and escalation procedures. Compliance officers configure detection rules and review processes through structured parameters rather than requesting code changes. This approach creates clear audit trails showing exactly which rules applied to each transaction and when those rules changed [11].

Cross-border settlement workflows coordinate complex multi-party transactions involving correspondent banks, payment networks, and regulatory reporting systems. Different currency pairs and country combinations trigger different validation and notification requirements. Declarative models let financial institutions define routing rules, cutoff times, and fallback procedures that the platform executes reliably across thousands of daily transactions.

The declarative approach particularly benefits institutions operating across multiple markets. A single platform deployment serves regional subsidiaries that each configure workflows to match local regulations and operational preferences.

Workflow Characteristics	Recommended Runtime	Rationale	Example Use Case
Duration: Seconds, Event-driven	Serverless (Lambda, Cloud Functions)	Cost efficiency, automatic scaling	Real-time bid processing

10.48047/jocaaa.2026.35.01.42

Duration: Minutes-Hours, Stateful	Temporal/Cadence	Durable state, automatic retries	Trade validation workflow
Duration: Days-Weeks, Human approvals	Temporal/Cadence with timers	Long-term state persistence	Compliance investigation
Hybrid: Fast path + slow path	Serverless + Temporal	Optimize for the common case, reliability for edge cases	Campaign approval (auto vs. manual)

Table 4: Runtime Selection Decision Matrix [7, 13]

8. Future Work

8.1 AI-Assisted Configuration

Current multi-tenant orchestration requires manual configuration tuning that relies heavily on domain expertise. Machine learning techniques could optimize workflow parameters automatically by analyzing historical execution patterns. Reinforcement learning algorithms might adjust timeout values, retry intervals, and resource allocations based on observed success rates and latency distributions.

Automated configuration recommendations represent another promising direction. When onboarding new tenants, platforms could analyze their business characteristics and suggest workflow configurations similar to successful peers. Natural language processing might extract requirements from unstructured documentation and generate initial configuration templates, reducing setup time and errors.

8.2 Advanced Isolation Techniques

Multi-tenant security continues evolving as threat models grow more sophisticated. Serverless compute isolation models offer stronger guarantees by executing each tenant's workflow activities in ephemeral, sandboxed environments that reset between invocations. This approach limits the blast radius of security vulnerabilities and prevents information leakage between tenants [7].

Zero-trust networking principles extend isolation to workflow communication patterns. Rather than assuming internal network traffic is safe, platforms could authenticate and authorize every interservice call, encrypt all data in transit, and continuously verify workflow execution contexts [11]. These techniques add overhead but significantly strengthen security postures for platforms handling sensitive financial or personal data.

8.3 Standardization Efforts

Industry-specific workflow specifications would accelerate platform development and improve interoperability. Financial services could benefit from standard workflow definitions for common processes like trade settlement, reconciliation, and compliance reporting. Advertising platforms might adopt shared specifications for campaign management and attribution modeling.

Open-source reference implementations would lower adoption barriers and foster community-driven improvements. Projects demonstrating declarative orchestration patterns for specific verticals—with realistic example configurations and integration patterns—would help teams evaluate approaches before committing to particular technologies [8][14]. These reference architectures could establish best practices around schema design, testing strategies, and operational patterns that benefit the broader cloud-native community.

Conclusion

Declarative orchestration offers a pragmatic resolution to a persistent tension in enterprise software: how to serve diverse tenant requirements without fragmenting platform codebases into unmaintainable collections of custom implementations. By separating canonical workflow definitions from tenant-specific configuration, platforms can evolve their core capabilities uniformly while giving customers meaningful control over business logic, approval hierarchies, and integration patterns. The advertising

10.48047/jocaaa.2026.35.01.42

and financial services domains demonstrate this model's versatility—both handle highvolume transactional flows and complex regulatory requirements, yet their operational rhythms differ substantially. Declarative approaches accommodate these differences through configuration rather than code forks. The benefits are tangible: faster tenant onboarding, reduced engineering duplication, improved audit trails, and centralized quality improvements that propagate to all customers simultaneously. However, success demands investment in governance frameworks, testing infrastructure, and operational tooling that manage configuration complexity at scale. Organizations must also accept that abstraction layers introduce debugging challenges and learning curves for teams accustomed to procedural code. As workflow specifications mature and reference implementations emerge, declarative orchestration will likely become standard practice for cloud-native platforms serving multiple tenants. The vision of workflows as programmable, composable products—rather than bespoke projects—represents a fundamental shift in how enterprise systems balance customization with operational sustainability.

References

[1] Rishi Kumar Sharma, "Multi-Tenant Architectures in Modern Cloud Computing: A Technical Deep Dive," International Journal of Scientific Research in Computer Science, Engineering and Information Technology, vol. 25, no. 1, Jan 2025. [Online]. Available: https://www.researchgate.net/publication/387867858

[2] Silver Touch Technologies, "Enterprise Architecture. https://www.silvertouch.com/enterprisesoftware-services/multi-tenant-saas-development/

[3] Oleg Shilovitsky, "Why Is Multi-Tenant SaaS the Future Standard for PLM?" OpenBOM, as of Nov. 2025. [Online]. Available: https://www.openbom.com/blog/plm-and-bom-management/why-ismulti-tenant-saas-the-future-standard-for-plm-day-20-of-30

[4] Flavius Dinu, "16 Most Useful Infrastructure as Code Tools for 2026," Spacelift, Dec. 2025. [Online]. Available: https://spacelift.io/blog/infrastructure-as-code-tools

[5] Sumanth P., "Cloud Orchestration in 2025: Top Tools, Benefits & AI Trends", Clarifai, Sep. 2025. [Online]. Available: https://www.clarifai.com/blog/cloud-orchestration

[7] Kuberkai Ionflow Docs, "Serverless Workflow Specification. [Online]. Available: https://docs.kuberkai.com/concepts/serverless-workflow-spec/

[8] Kestra, "Declarative Data Orchestration." [Online]. Available: https://kestra.io/features/declarative-data-orchestration

[9] Bill Doerrfeld, "9 CNCF Tools for Automation and Configuration," Cloud Native Now, May 2022. [Online]. Available: https://cloudnativenow.com/features/9-cncf-tools-for-automation-andconfiguration/

[10] Suraj Subramanian, "Temporal: Revolutionizing Workflow Orchestration in Microservices Architectures," Medium, June 2024. [Online]. Available: https://medium.com/@surajsub_68985/temporal-revolutionizing-workflow-orchestration-inmicroservices-architectures-f8265afa4dc0

[11] Temporal Technologies, "Welcome to Temporal" [Online]. Available: https://temporal.io/

[12] Contrary Research, "Temporal Technologies. [Online]. Available: https://research.contrary.com/company/temporal-technologies

[13] Cadence Workflow, "Cadence: Orchestrate with Confidence". [Online]. Available: https://cadenceworkflow.io/

[14] Conductor OSS Foundation, "Conductor – Scalable Workflow Orchestration," 2025. [Online]. Available: https://conductor-oss.org/

10.48047/jocaaa.2026.35.01.42

[6] Cloud Native Computing Foundation, "Serverless Workflow," Project Overview, 2020–2025. [Online]. Available: https://serverlessworkflow.io/

[15] Parag Solanki, "How to Build a Multi-Tenant SaaS Inventory System on AWS Cloud," WeblineIndia, Jun 2025. [Online]. Available: https://www.weblineindia.com/blog/multi-tenant-saasinventory-system-on-aws/