Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Interoperability Standards in Modern Workflow Orchestration Systems

Devinder Tokas
Independent Researcher, USA

ARTICLE INFO

Received: 20 Feb 2026
Revised: 22 Feb 2026

ABSTRACT

Modern cloud workflows often involve interoperability between orchestration systems from different vendors, eased through interoperability contracts for event formats, telemetry protocols, data lineage, and artifact packaging. Orchestration systems can be classified across 5 architecture layers spanning workflow definition interoperability, execution state, integration protocols, observability, and deployment packaging. CloudEvents provides vendor-neutral envelopes for events that can flow between event brokers; OpenTelemetry provides vendor-agnostic telemetry data collection to and from user code in applications, using standardized APIs, SDKs, and OTLP. OpenLineage supports dataset interactions at orchestration boundaries, useful for data governance and impact analysis. The OCI Image Specification enables portable container execution across runtime environments. Kubernetes CustomResourceDefinitions extend platform capabilities to support built-in lifecycle management features and declarative tooling. Service reliability governance uses Service Level Objectives and error budgets to drive tradeoffs between availability and operational cost, preventing organizations from confusing internal objectives with Service Level Agreements with customers when delivering application reliability. Performance benchmarks indicate that the OpenTelemetry Collectors can be scaled up to thousands of events per second across different cluster configurations, maintaining low average latencies and resource consumption. Organizations in the full observability stage exhibit radically improved mean time to detection and mean time to resolution, deploy several times a day, and have change failure rates under five percent. Contract-first architecture enables the decomposition of monolithic workflow platforms into pluggable components with stable, versioned interfaces that do not cause vendor lock-in and can evolve the ecosystem across a container cluster, data pipelines, and service architectures via open APIs.

Keywords: CloudEvents, OpenTelemetry, workflow orchestration interoperability, Service Level Objectives, container standardization

1. Introduction

Modern cloud resources need a workflow engine that is agnostic to the vendor and runtime environment, so the definition of workflows is portable and the signals of workflows are meaningful in a heterogeneous cloud computing environment like a container cluster, data pipeline, or service architecture. Adopting an interoperability-first approach means standard contracts like event envelopes, telemetry, lineage, and artifact packaging are first-class architecture constructs. This allows teams to incrementally expand their ecosystem of workflows over time without having to rewrite their business logic when transitioning to a new engine or platform capability.[5]

As event-driven architectures become more common, the need for interoperability between event-based architectures becomes more of an issue, as there is no standard way for projects to describe events or how events should be connected. CloudEvents is a specification of a common way to describe event data, developed as a solution to fragmentation by the Serverless Working Group of the Cloud Native Computing Foundation. CloudEvents is designed for use across multiple protocols, multiple subscribers, and multiple producers. It deals with the standardization of the binary and structured data encodings. The specification defines a set of standard event attributes—id, source, type, and time—which are general enough as not to depend on the protocol used. This allows events to be constructed, which can be sent across different broker and router infrastructures in a standardized way without the need to create protocol adapters at each integration point [1].

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

OpenTelemetry also provides interoperability in observability areas with an open-source toolkit, application programming interfaces (APIs), and software development kits (SDKs) to help software analysts understand how software behaves and performs. OpenTelemetry is a Cloud Native Computing Foundation (CNCF) project, replacing both OpenTracing and OpenCensus. OpenTelemetry defines a set of APIs, libraries, agents, and instrumentation to generate, collect, process, and export telemetry data like logs, metrics, and traces [2]. The architecture contains specifications that define standardized telemetry formats and approaches to instrumentation that create similar experiences across programming languages, data models that describe the fields of each of the signals and their relations, and APIs that define the methods to instrument applications and serve as entry points for instrumentation. There is a dedicated implementation for each supported programming language. SDKs implement the OpenTelemetry APIs and define how instrumentation interacts with other parts of a system and how telemetry is created and correlated. SDKs are available for several programming languages, and the OpenTelemetry Protocol is the vendor-neutral, tool-agnostic specification for the transport, encoding, and delivery of OpenTelemetry data. SDKs emit OpenTelemetry data using the OpenTelemetry Protocol (OTLP) over HTTP and gRPC. Exporter plugins convert the OpenTelemetry Protocol into formats and protocols understood by backends that are not aware of OTLP natively.

A contract-first architecture is based on the observation that workflow engines have become integration platforms for batch compute, stream processing, machine learning pipelines, infrastructure provisioning, and long-running business processes. By applying contract-first principles, such a platform can be refactored from monolithic to composable, replaceable components. These components exchange standard boundary artifacts encoded in stable and versioned contracts as opposed to proprietary application programming interfaces (APIs).

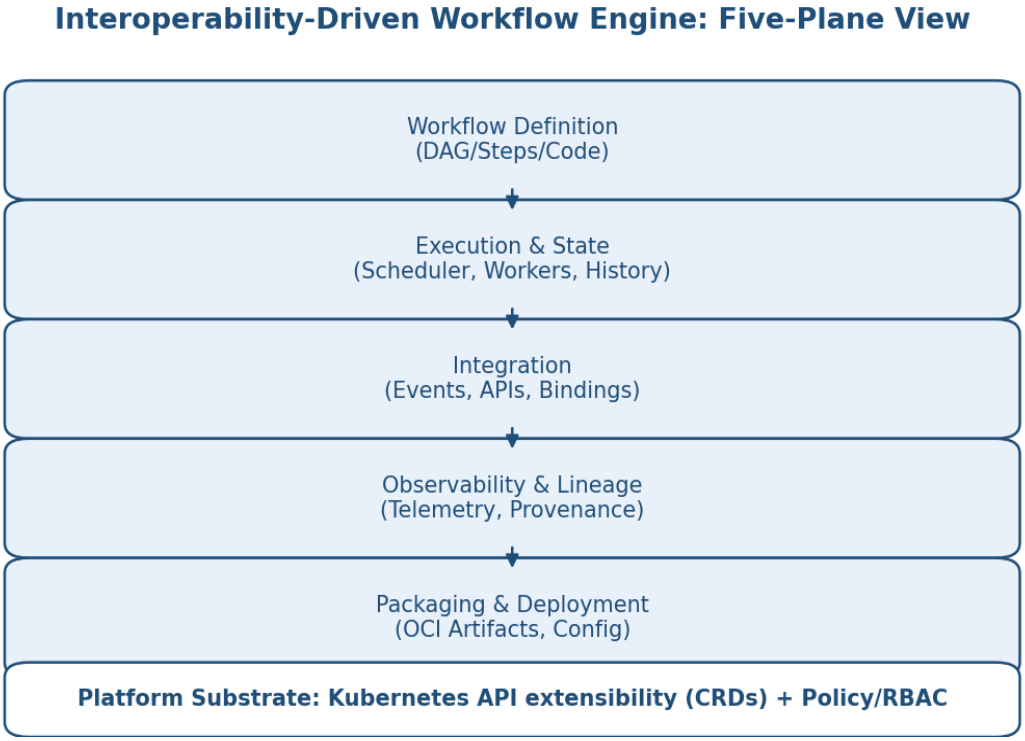


Figure 1. Interoperability-driven workflow engines can be understood through five planes, with platform API extensibility as a cross-cutting substrate.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

2. Multi-Dimensional Interoperability Framework

The workflow engine interoperability identifies five interoperability planes, which define the scope of standardization needed to avoid vendor lock-in and to support the evolution of the ecosystem. The Workflow Definition Plane addresses questions of logic representation from the perspective of logical portability among engines, e.g., workflows as directed acyclic graphs, linear sequences of steps, or executable code. The Execution & State Plane addresses questions of scheduling, retry after failures, and recovery from failures via persistent state and other mechanisms. It includes how workflow engines provide exactly-once or at-least-once execution semantics, durable workflow histories, and restoration of workflow execution in case of infrastructure failure or crashing processes. Integration Plane components include triggers, notifications, and inter-process communication via standard protocols, and the ways in which workflows react to external events and publish state transitions to downstream consumers via message brokers, event routers, and service meshes.

Figure 2: Five-Plane Interoperability Architecture for Vendor-Neutral Workflow Orchestration

Enterprise use of cloud infrastructure automation and programmatic management of compute, network, and storage via declarative programming interfaces and software-defined infrastructure is shown to decrease manual provisioning times and increase consistency of deployments. Baseline performance benchmarking of a two-replica cluster with zero partitions shows a median latency of 34 ms (99th percentile 165 ms), producer throughput of 4289 thousand events per second, and consumer throughput of 4202 thousand events per second [4]. In organizations that have adopted these frameworks, the latency and throughput of producers and consumers are increasingly consistent across environments: producer throughput is 195,000 events per second at 49 milliseconds median latency and 179 milliseconds 99th percentile latency. Consumer throughput is 174,000 events per second at 92 milliseconds median latency and 209 milliseconds 99th percentile latency, both at a single kilobyte per event [4].

Cloud-agnostic frameworks enable interoperability and abstraction and support a common model of provisioning across major cloud providers. It was shown that using cloud-agnostic frameworks can lower vendor-specific deployment costs and cross-cloud resource management overhead costs.

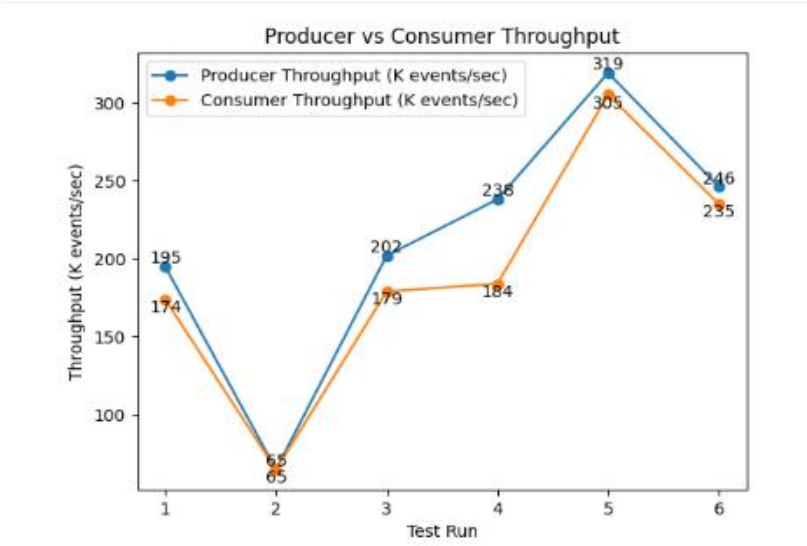
Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Supporting events of all sizes with the default configuration, the producer can push 65000 events/s for a median latency of 141 ms and a 99th percentile latency of 273 ms. The consumer can consume 65000 events/s for a median latency of 138 ms and a 99th percentile latency of 280 ms [4]. In the scale-out model with four partitions, the producer rates are over 200 thousand events per second with a median latency of 32 ms and a 99th percentile latency of 291 ms. The consumer rates are 179 thousand events per second with a median latency of 73 ms and a 99th percentile latency of 213 ms. This allows companies to split resources across multiple clouds and avoid vendor lock-in, using similar management processes across all platforms [4].

The Observability & Lineage Plane standardizes the end-to-end correlation of traces, metrics, and data lineage, and enables instrumentation contracts to ensure that distributed executions of a workflow across clusters and runtimes can all correlate to each other within end-to-end observability backends. Organizations that adopt centralized observability and monitoring tooling have full visibility into all dimensions of system behavior. The baseline throughputs of the producer are 238k events per median, and the 99th percentile latencies are 16 milliseconds and 352 ms, respectively. The consumer throughput is 184k events per second with corresponding latencies of 67 ms (median) and 279 ms (99th percentile). The producer throughput scales up to 319k events/second, and the median and 99th percentile latencies are 19 and 168 milliseconds, respectively. The scaled throughput for consumers is 305 thousand events per second with a median latency of 41 milliseconds and a 99th percentile latency of 186 milliseconds, massively reducing the resolution time of incidents due to complete visibility of the system [4].

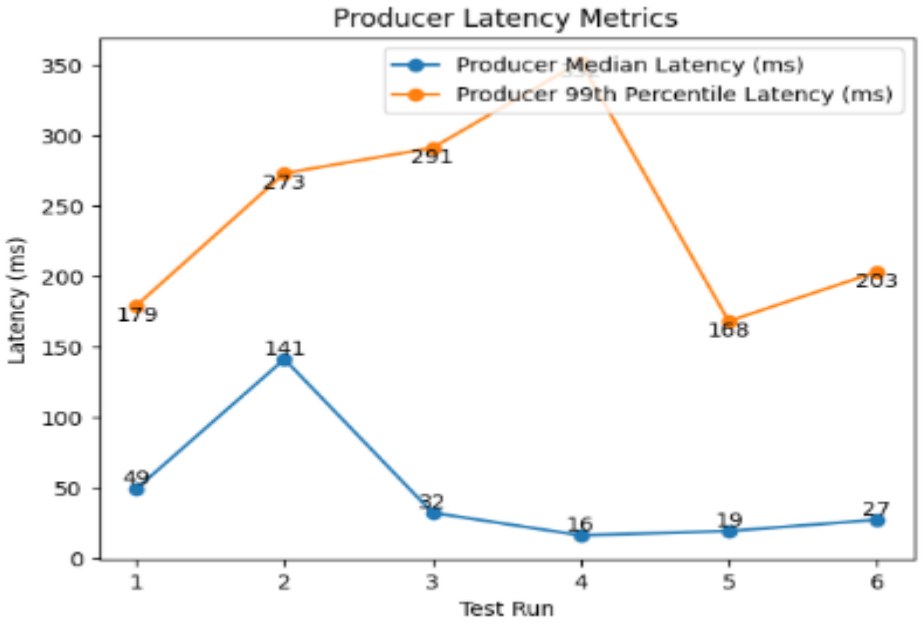
Developer enablement via Internal Developer Platforms with pipelines, templates, and observability tooling for developing applications has been shown to increase developer productivity considerably. For a scale-out configuration with one-kilobyte events, producer throughput with a median latency of 27 milliseconds and a 99th percentile latency of 203 milliseconds is 246 thousand events per second. These configurations yield a median consumer latency of 47 milliseconds and a 99th percentile latency of 336 milliseconds at 235 thousand events per second [4]. The Packaging & Deployment Plane deploys portable artifacts using container standards and declarative deployment semantics (OCI Image Specification). The OCI Image Specification defines standard formats for container images, which act as a baseline for the development of interoperable tools to build and ship images across heterogeneous execution environments. Baseline benchmarks for consumer throughput range from 9646 thousand events per second for baseline zero-partition configurations to 813 thousand events per second for production-optimized single-kilobyte event processing in scale-out deployments [4].



Graph 1: Producer and Consumer Throughput Across Test Runs [4]

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article



Graph 2: Producer Median and 99th Percentile Latency Across Test Runs [4]

3. Core Standardization Primitives

Five open standards form the basis of all interoperable workflow systems: CloudEvents, a vendor-agnostic envelope for event metadata. Metadata intrinsic to all events includes fields such as specversion for the version used, id for the globally unique event identifier, source specifying the event's source, and type specifying the event type. This allows workflow triggers and state transitions to cross heterogeneous messaging brokers without requiring custom adapter logic [5]. The specification defines a binary format, where the protocol allows the distinction between message metadata and data, and a structured format, where the protocol encodes data and metadata in a structured format described as structured objects. Most implementations of HTTP Binary encoding of the specification use the ce-* headers (including ce-specversion, ce-id, ce-source, ce-type, and ce-time), whereas most implementations of HTTP Structured encoding use application/cloudevents+json content type payloads to express the entire context of an event [5]. SDKs for the Go language, JavaScript, Java, C#, Ruby, PHP, Python, Rust, and PowerShell have been set up to help developers build standardized cloud events across all stacks. The SDKs also retain the structure of the event envelope, making the routing and filtering of cloud events independent of where they originate and which broker is used [5].

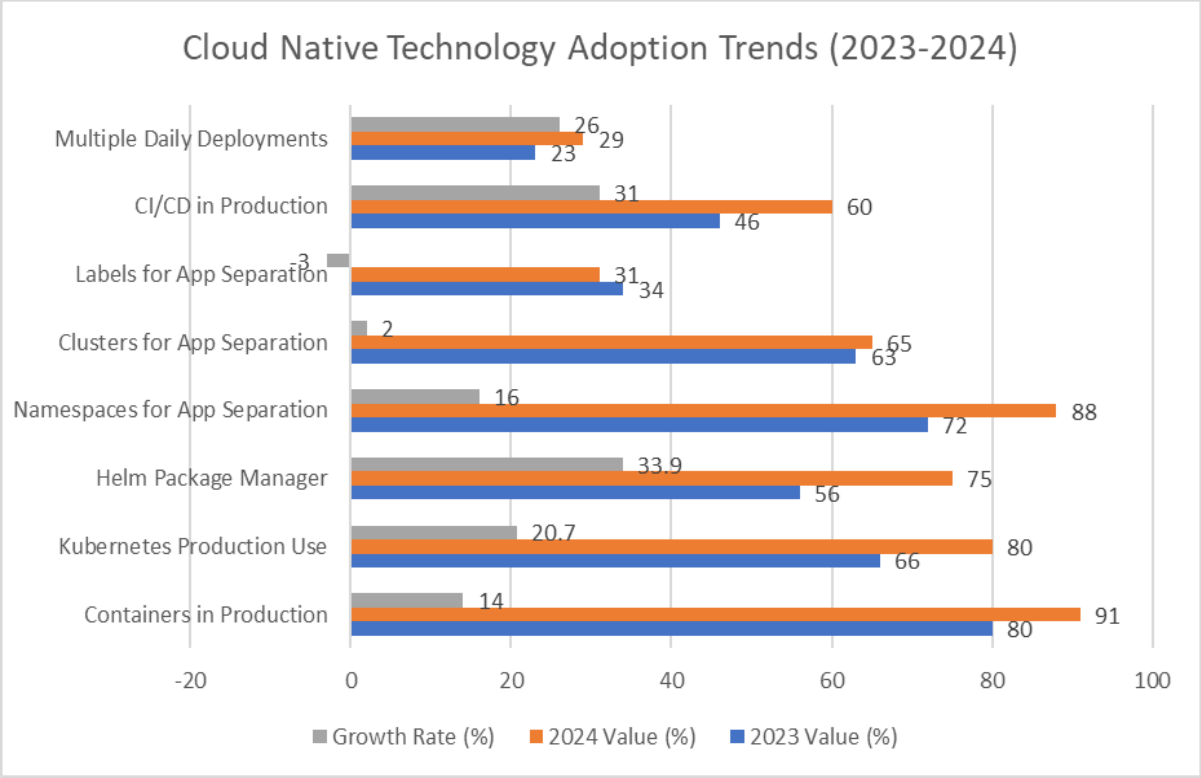
In the 2024 CNCF Annual Survey of 689 respondents, cloud native adoption reached 89 percent in 2024, defined as some, much, or nearly all application development and deployment being cloud native. Ninety-one percent of organizations used containers in production for most or for a few applications (compared to 80 percent in 2023), a growth of 14 percent year-on-year. Kubernetes usage in production reached 80 percent, compared to 66 percent in 2023. This represented a growth rate of 20.7 percent. Ninety-three percent of the respondents were either using, piloting, or actively evaluating Kubernetes. The average number of containers per organization was 2,341, up from 1,140, representing a 27 percent increase. Helm continued to be the most used package manager for Kubernetes. The percentage of organizations using Helm for Kubernetes rose from 56 percent in 2023 to 75 percent in 2024, representing a year-over-year growth rate of 33.9 percent. For isolation strategies, namespaces increased from 72 percent in 2023 to 88 percent in 2024, while clusters grew by 2 percent to 65 percent, and labels decreased by 3 percent to 31 percent [6].

OpenLineage specifies the emission of lineage events at orchestration boundaries, where datasets are read and written by every step of a given workload. In addition, OpenLineage defines provenance

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

lineage run events that describe jobs, runs, and datasets in formats consumable by many lineage backends and governance tools [5]. The OCI Image Specification defines container images as portable execution units across schedulers and runtimes. By specifying standard manifests, layer formats, and configuration formats, the specification allows tools to build, transport, and run images [5]. Kubernetes CustomResourceDefinitions extend the platform API to increase the supported user-defined custom resource types to consist of workflow resources with built-in lifecycle management capabilities, RBAC controls, and Kubernetes declarative API-compliant tools [5]. Seventy-seven percent of respondents indicated some, much, or almost all of their tools and processes are designed around GitOps best practices. There was a 31 percent increase from 2023 to 2024 in the number of respondents deploying CI/CD in production for their software applications: in 2023, 46 percent of respondents; in 2024, 60 percent of respondents. Organizations are continuing to see code created through the software development lifecycle in faster timelines, with 29 percent of respondents releasing code multiple times per day (up from 23 percent in 2023) [6].



Graph 3: CNCF Survey Results: Container and Kubernetes Growth Metrics [6]

4. Distributed Data Pipeline Integration

Big data platforms often face interoperability issues. Big data pipelines may traverse multiple tools, teams, and technology stacks, which requires advanced infrastructure management and developer enablement capabilities in heterogeneous cloud environments. The challenges of the observational gap in modern distributed systems can generally be attributed to factors such as: Complexity, for example, in microservices architectures, dozens of independently deployable services each have their own dependencies and failure modes. Dynamism, for example, in cloud-native systems where auto-scaling and ephemeral resources are common, makes it hard to maintain a consistent state. Polyglot persistence: applications often use many different databases and storage technologies, each with its own strengths and weaknesses. Distributed transactions, tracing the flow of a single user request

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

across multiple services and databases, can be difficult when the different components use different communication patterns (especially when asynchronous) [7].

An interoperability-driven design writes standard contracts at the interfaces that support the use of standardized protocols. Observability is a major concern for 94 percent of DevOps practitioners, according to industry studies. Because of the interop-driven design, observability is understood as observability across all three signals: metrics, logs, and traces. Metrics are quantitative measurements and can be aggregated, thereby leading to storing them in time-series databases. Logs are sometimes verbose, can be rich in context, contain unstructured or semi-structured data, and can be used for debugging and auditing. Traces are a record of single requests as they are passed from service to service, including distributed context propagation, timings, and causal relationships. When used as part of holistic observability, they may help organizations improve MTTR by as much as 80 percent. They can speed up mean time to detection, mean time to resolution, and minimize service disruptions because latency spikes and errors can often be quickly and accurately identified before they impact end users [7].

OpenTelemetry context propagation allows ingestion, validation, transformation, and serving operations to be correlated as one distributed trace in different stages of the pipeline, enabling performance measurement for every component. Kubernetes provides built-in elastic autoscaling through the Horizontal and Vertical Pod Autoscalers and per-node autoscalers, ideal for cloud-native workloads. It has been observed that these deployments will incur SLO violations and/or costs due to reactive scaling, lack of application-level knowledge, and opaque behavior [8]. Evaluating on representative microservice and event-driven workloads, the observation that using SLO-aware autoscaling instead of default or tuned Kubernetes autoscaling reduces SLO violation duration, scaling delay, and infrastructure cost by up to 31 percent, 24 percent, and 18 percent, respectively, while providing a control behavior that is stable and auditable [8].

Lineage emission is placed between two workflow steps: reading and writing datasets. OpenLineage events are emitted to a multi-backend format consumable by different governance backends in order to enable organizations to capture which datasets were read and written by each step of a workflow, enabling impact analysis and enabling governance. Packaging transformations as OCI images enable portable execution of compute workloads to be moved between clusters or batch workloads. According to the 2023 State of Observability Report, organizations with mature observability practices are 69 percent faster in mean time to resolution (MTTR) for outages, 2.9 times more likely to proactively detect issues before they impact customers, 66 percent more likely to have high customer satisfaction, and 2.5 times more likely to exceed their organization's performance goals.

Figure 4. End-to-End Workflow Interoperability Process Flow Across Execution, Telemetry, and Governance Layers

Challenge	Description	Operational Impact
Increased System Complexity	Microservices with independent deployments	Unique dependencies and failure modes
Dynamic Infrastructure	Auto-scaling and ephemeral resources	Inconsistent system state visibility
Polyglot Persistence	Multiple database types	Varying performance characteristics
Distributed Transactions	Cross-service request flows	Asynchronous communication complexity

Table 1: Distributed Systems Monitoring Challenges [7]

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

5. Service Architecture Reliability Patterns

Organizations performing high-scale service workloads need to be able to govern reliability through industry-standard observability signals to find the right balance between availability and operational cost. Service Level Objectives describe a numerical target for a system's reliability. Google's SRE methodology recommends it be set to the minimum acceptable level of reliability, as pushing for the highest availability incurs a cost and slows down the pace of development [9]. When error budgets are exceeded, engineering organizations make policy changes and switch from building new features to stabilizing systems in order to minimize mean time to recovery and avoid repeating patterns and conditions that cause failure.

Although internal SLOs and external SLAs are usually treated separately in Google's SRE model, external SLAs are typically much looser (for example, 99.9 percent availability) than internal SLOs (99.95 percent availability) to provide operational headroom. This means explicit checking of whether the SLO is being met over the calendar period of the SLA, so that the system can move out of the 'trouble zone' before a penalty for not meeting the SLA (such as partial refunds or subscription extensions) is incurred. Service Level Indicators are concrete, quantitative behaviors (such as the ratio of successful probes into a system to all probes) that are to be monitored to determine whether a system is in compliance with the intended behavior defined in an SLO [9].

Delivery performance builds on these software reliability principles with the DORA metrics of deployment frequency, lead time for changes, change failure rate, and time to restore service, each in the context of a workflow. Elite performers deliver multiple times per day, have a lead time for changes of less than 60 minutes, a change failure rate of under 5 percent, and a time to restore service of less than 60 minutes. High performers deploy between once per day and once per week and have a change failure rate of ten to fifteen percent. Certain tools have led to empirically meaningful improvements, e.g., observability tools like Prometheus, Grafana, and OpenTelemetry. For instance, CloudRetail (2025) achieved a 73 percent reduction in mean time to detection from 15 minutes to 4 minutes and a 68 percent reduction in mean time to resolution from 45 minutes to 14 minutes for 200 microservices processing millions of transactions [10].

These measurements are related to passing context propagation identifiers (workflow type, run ID, and step ID) through OpenTelemetry, which enables the dashboards to work on any engine and continue working if the engine changes [10]. OpenTelemetry Collectors have been benchmarked to process 15,000 events per second with less than 12 milliseconds of query latency, with 256MB of memory and 15 percent CPU. Successful examples correlate signals by treating workflow IDs as first-class telemetry attributes, issuing OpenLineage events at job execution boundaries, rather than through thorough instrumentation of a job's internals, and using declarative governance through platform APIs to express and enforce a consistent set of constraints on container images, resource limits, and network egress on different types of workflows [10].

Concept	Purpose	Implementation
Service Level Objective	Define reliability targets	Numerical availability thresholds
Service Level Agreement	Customer commitments	External promises with penalties
Service Level Indicator	Measure actual behavior	Ratio of successful operations
Error Budget	Balance velocity and stability	Shift priorities when consumed

Table 2: SRE Core Concepts [9]

Conclusion

Interoperable workflow orchestration systems work with multiple systems and do not lock users into a single system or vendor. Contract-driven specifications enable interoperability between execution systems and observability backends through CloudEvents, OpenTelemetry, OpenLineage, OCI specifications, and Kubernetes extensions, which together create an ecosystem of cloud-agnostic,

interoperable workflow declarations, execution state, event-driven integration, lineage, and artifact uploads. Using these standardization primitives, organizations have observed improved reliability of their systems, increased productivity of their operations and developers, improved observability for multi-dimensional correlations between distributed traces, and standardized declarative governance enforcement. By addressing issues specific to the cloud-native ecosystem, such as the complexity of microservices, infrastructure as code, polyglot data, and distributed transactions, the multi-dimensional interoperability framework successfully applies thorough observability to reduce mean time to resolution (MTTR) and prevent customer-facing issues. Service reliability patterns, similar to service level objectives and error budgets, help engineering teams find the right balance between feature velocity and stabilization work that can sustainably maintain arbitrary availability targets. Performance numbers include high throughput and low latency for cluster-wide benchmark and observation stacks, achieving thousands of events per second with minimal resource consumption. In the contract-first architectural pattern, workflow engines are seen as integration buses that propagate and instantiate batch compute engines, stream processing engines, machine learning pipelines, and infrastructure provisioning with stable, versioned interfaces to avoid vendor lock-in, while supporting evolution of the overall cloud-native ecosystem. Organizations adopting interoperability standards can survive and flourish as the cloud-native space becomes ever more complex by breaking monolithic platforms into replaceable components that communicate through standard interfaces instead of proprietary APIs.

References

[1] Olawepo Olayemi, "Cloud event specification (a first look)," Medium, 2022. [Online]. Available: https://sejuba.medium.com/cloud-event-specification-d804f7f00f39.
[2] Jay Livens, "What is OpenTelemetry? An open-source standard for logs, metrics, and traces," Dynatrace, 2025. [Online]. Available: https://www.dynatrace.com/news/blog/what-is-opentelemetry/
[3] Lakshmi Priyanka Pillati, "Enterprise Cloud Infrastructure Automation and Platform Engineering for Multi-Cloud Global Systems," Journal of Information Systems Engineering and Management, 2025. [Online]. Available: https://jisem-journal.com/index.php/journal/article/view/12545/5821
[4] Haochen Pan et al., "Octopus: Experiences with a Hybrid Event-Driven Architecture for Distributed Scientific Computing," arXiv, 2024. [Online]. Available: https://arxiv.org/pdf/2407.11432
[5] Rahul Reddy Bandhela. (2022). Advancing Banking Systems with Federated Learning and a Fuzzy-Based Blockchain Framework for Secure and Efficient Transactions. Journal of Informatics Education and Research, 2(2)
[6] Chetan Bansal et al., "DeCaf: Diagnosing and Triaging Performance Issues in Large-Scale Cloud Services," arXiv, 2020. [Online]. Available:https://arxiv.org/pdf/1910.05339
[7] Valerie Silverthorne et al., "Cloud Native 2024: Approaching a Decade of Code, Cloud, and Change," Cloud Native Computing Foundation and The Linux Foundation. [Online]. Available: https://www.cncf.io/wp-content/uploads/2025/04/cncf_annual_survey24_031225a.pdf.
[8] Sridhar Nelloru, "HOLISTIC OBSERVABILITY: ALIGNING METRICS, LOGS, AND TRACES IN CLOUDNATIVE SYSTEMS," International Journal of Computer Engineering and Technology (IJCET), 2025. [Online]. Available: https://iaeme.com/MasterAdmin/Journal_uploads/IJCET/VOLUME_16_ISSUE_1/IJCET_16_01_025.pdf[8] Vinoth Punniyamoorthy et al., "An SLO Driven and Cost-Aware Autoscaling Framework for Kubernetes," arXiv, 2025. Available: https://arxiv.org/pdf/2512.23415
[9] Jay Judkowitz, Mark Carter, "SRE fundamentals: SLIs, SLAs and SLOs," Google Cloud Blog, Jul. 2018. [Online]. Available: https://cloud.google.com/blog/products/devops-sre/sre-fundamentals-slis-slas-and-slos
[10] ANKUSH CHOUDHARY, "Cloud-Native Monitoring Solutions: Implementing Observability with Prometheus, Grafana, and OpenTelemetry 2025," 2025. [Online]. Available: https://www.johal.in/cloud-native-monitoring-solutions-implementing-observability-with-prometheus-grafana-and-opentelemetry-2025/