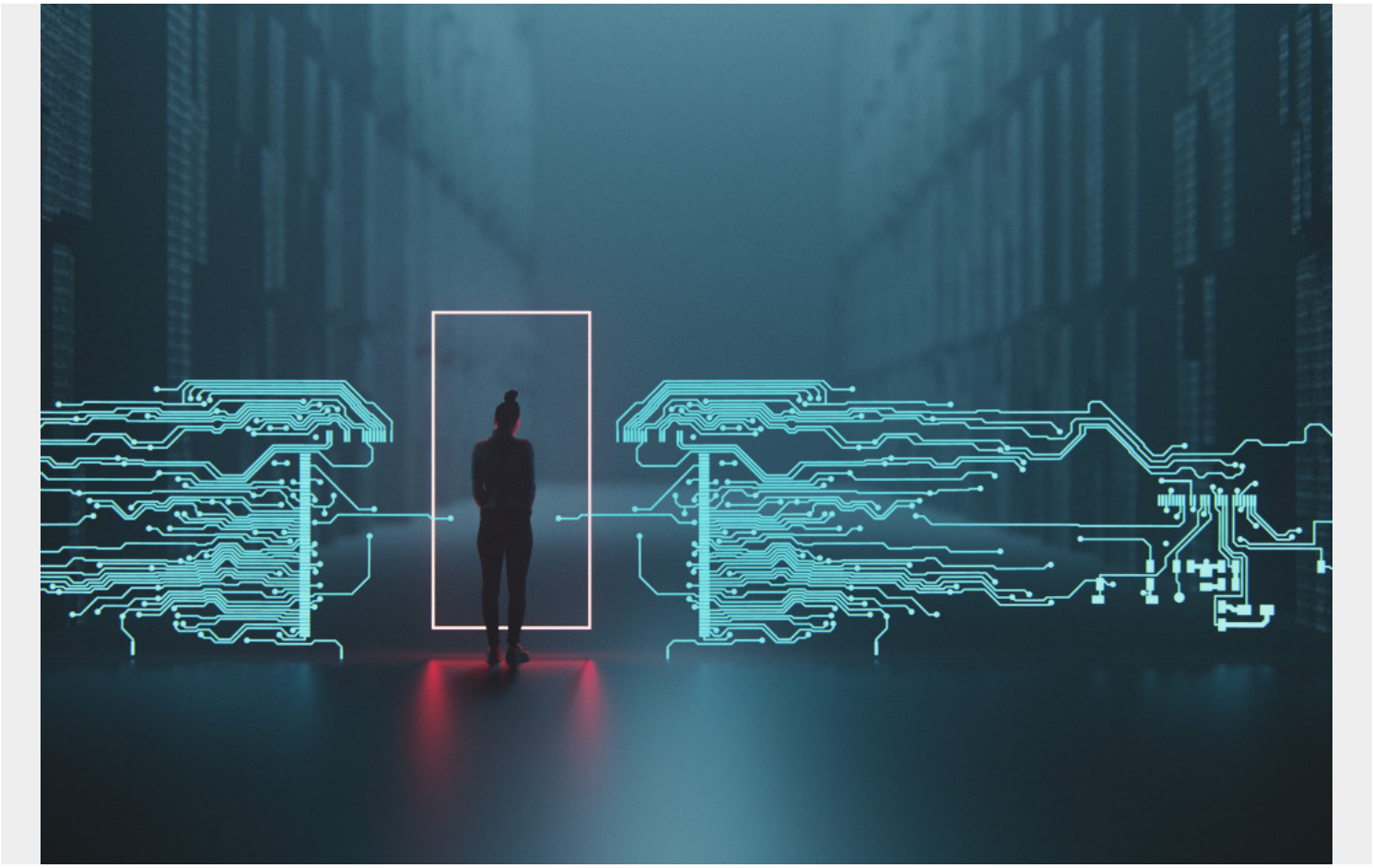
12 BEST PRACTICES FOR IMPLEMENTING APPLICATION WORKFLOW ORCHESTRATION



The most effective way to implement application workflow orchestration is to treat workflows like software: version-controlled, modular, well-documented, and built for failure recovery from the start. The 12 best practices below provide a practical implementation framework for teams adopting platforms like Control-M to orchestrate data and application workflows at scale.

Companies across many industries have embraced application workflow orchestration as a way to drive digital modernization forward. From streamlining targeted advertising to orchestrating complex data pipelines, the right workflow orchestration for hybrid cloud is the foundation. Here are 12 best practices to follow on your orchestration journey.

> ***Application workflow orchestration*** makes sure data and application workflows are carried out in the correct sequence and at the correct time to ensure the successful delivery of a business service.

If you're ready to start your application workflow orchestration journey, here are 12 best practices to follow.

What are the best practices for implementing application workflow orchestration?

Successful application workflow orchestration implementation depends on twelve principles—spanning workflow design, governance, visibility, and resilience. Here is each one in detail.

1. Support an “as-code” approach

Storing and managing workflows in a text or code-like format is the foundation of a modern, maintainable orchestration practice—because version control is mandatory regardless of how workflows are authored.

Whether workflows are created through a graphical interface or written directly in code, the platform must allow them to be stored in a versionable format. This enables modern deployment pipelines, peer review, and rollback capability—the same standards applied to application code

2. Think in microservices

Monolithic workflows create the same maintenance and scalability problems as monolithic applications. Design workflows as modular, loosely coupled microservices that can be connected, reused, and recombined independently.

Service (Flow) A:	Service B:	Service C:
Do something1, emit “something1 done”	Wait for “Service A”	DO NOT run while something2
Emit “something2 running,” Do something2,	done	is running
Emit “something2 done”	Do BThing, emit	Wait for BThing done
Emit “Service A” done	BThing done	Etc.

This structure makes dependencies explicit, simplifies debugging, and allows individual services to be updated without affecting the rest of the flow.

3. Don’t reinvent the wheel

If a function is used more than once, build it once as a reusable workflow class that can be instantiated as many times as needed and maintained in a single place.

Rather than creating multiple versions of the same service to handle variations, use variables or parameters to accommodate the differences. This keeps the codebase lean and prevents divergence between copies over time.

4. Process lineage

Effective problem analysis in complex workflows requires the ability to trace the exact sequence of processing that brought a flow to its current state—not just the data lineage, but the process lineage.

Without process lineage, diagnosing failures in pub/sub or "launch-and-forget" trigger patterns is extremely difficult. The ability to reconstruct what ran, when, and in what order is a mandatory requirement for any orchestration platform managing business-critical workflows.

5. Make the work visible

Workflow relationships should be visible in real time—including sensors and watchers that are waiting for events that have not yet occurred.

The most valuable visualization scenarios are often the quiet ones: everything appears normal, but nothing is running. A clear line of sight between a waiting trigger and the downstream process that was never started—because the expected event never fired—can surface problems that would otherwise go undetected until an SLA is breached.

6. Codify SLAs

The best way to define a non-event as an error is to set an explicit service level expectation—so that missing a deadline is treated as a failure, not a gap in monitoring.

At its most basic, an unmet SLA is identified as an error. For example: a file is expected between 4 PM and 6 PM, takes 15 minutes to cleanse and 30 minutes to process, setting the SLA at 6:45 PM. If processing hasn't completed by then, the error is surfaced at 6:45 PM.

A more sophisticated approach uses trending data from previous executions to predict SLA breaches early. If the cleanse step hasn't finished by 6:15 PM, the system can generate alerts immediately—maximizing the time available to intervene. Adding "slack time" indicators gives operators a live view of how many minutes remain before the SLA is breached.

7. Categorize

Tagging workflow objects with meaningful values—covering relationships, ownership, environment, and business context—makes large orchestration environments manageable and auditable.

As workflows multiply across teams and environments, ad hoc naming and loose categorization make it increasingly difficult to identify what owns what, what depends on what, and what the business impact of a failure might be. Consistent tagging from the start avoids this problem at scale.

8. Use coding conventions

Naming conventions for workflows carry the same importance as naming conventions in application code—ambiguous names compound quickly in multi-user environments.

A workflow called "MyDataPipeLine" may be clear when built in isolation, but becomes confusing the moment others need to maintain, debug, or extend it. Use descriptive, qualified names—similar to naming an API endpoint "CreditCardValidation" rather than simply "Validate"—so that purpose and scope are apparent without needing to open the workflow definition.

9. Think of others

Even workflows that currently have a single owner will eventually need to be modified, debugged, or handed off—and comments and documentation make that transition significantly faster.

Include descriptions on all workflows. For complex ones, add supporting documentation. Critically, update documentation in the same commit or change event as the workflow itself—documentation that drifts from the actual implementation is often worse than no documentation at all.

10. Keep track

For workflows supporting important applications, comprehensive audit data is non-negotiable: who built the workflow, who ran it, who paused or killed it and why, whether it succeeded or failed, when and how it was fixed.

Ensure the orchestration platform captures the full history of every workflow execution. This audit trail supports compliance, accelerates incident response, and provides the context needed to distinguish a new failure pattern from a recurring known issue.

11. Prepare for the worst

Workflows will fail. The question is whether the data needed to diagnose and fix the failure is available when it happens.

Collect and retain execution data long enough to enable meaningful comparison between new failures and previous failures or successful runs. This directly supports the "keep track" requirement and speeds root cause analysis—especially for intermittent failures that are difficult to reproduce.

12. Harness intelligent self-healing

The most resilient orchestration implementations define not just what constitutes a failure, but what automated recovery action should follow each specific failure type.

Look for platforms that allow flexible, context-aware error definitions. Code that exits with a zero exit code but emits catastrophic error messages is a real scenario—the orchestration layer should be able to distinguish it from a genuine failure. Configurable automated recovery actions, tuned to each specific situation, reduce manual intervention and minimize downstream impact when things go wrong.

Frequently asked questions

What is application workflow orchestration?

Application workflow orchestration is the automated coordination of data and application workflows to ensure they run in the correct sequence and at the correct time, delivering a complete business service reliably. Platforms like Control-M manage the dependencies, scheduling, monitoring, and error handling across all workflows in an orchestrated environment.

What is the difference between workflow orchestration and workflow automation?

Workflow automation refers to executing individual tasks without manual intervention. Workflow orchestration goes further—it coordinates multiple automated tasks across systems, managing dependencies, sequencing, error recovery, and SLAs to ensure end-to-end business processes complete successfully. Orchestration requires automation as a foundation but adds governance and visibility across the full workflow lifecycle.

Why is an "as-code" approach important for workflow orchestration?

Treating workflows as code enables version control, peer review, automated testing, and repeatable deployments—the same engineering discipline applied to application development. Without an as-code approach, workflow definitions exist only in proprietary tool interfaces, making them difficult to

audit, roll back, or integrate into CI/CD pipelines.

How do you prevent SLA breaches in workflow orchestration?

Preventing SLA breaches requires codifying service level expectations directly in the orchestration platform so that late or missing executions are treated as errors. Advanced platforms use historical execution data to predict breaches before they occur, generating early alerts that give operators maximum time to intervene. Slack time indicators show how many minutes remain before a breach—enabling proactive rather than reactive responses.

What should I look for in an application workflow orchestration platform?

Key capabilities to evaluate include as-code workflow authoring and version control support, modular workflow design with reusable components, process lineage tracking, real-time visualization of workflow status and dependencies, configurable SLA management with predictive alerting, comprehensive audit logging, and intelligent self-healing with configurable automated recovery actions. Control-M from BMC covers all of these capabilities across hybrid and multi-cloud environments.

The views and opinions expressed in this post are those of the author and do not necessarily reflect the official position of BMC.