

AVATO

Technical Architecture & Modernization Approach

White Paper 001

A technical reference for enterprise architects, CTOs, and senior technical consultants



Library
Version
Focus

Avato Technical Library

1.0 — August 2026

Architecture, deployment, operations, and modernization strategy

AVATO

Executive Technical Summary

Organizations rarely struggle because they lack another API, connector, or application. They struggle because critical systems are tightly coupled to one another, to historical data models, to specific protocols, and to implementation decisions that are difficult to change without operational risk.

Avato is a configuration-driven orchestration platform that introduces a stable interoperability layer between systems. It receives messages and events from existing applications, applies security and policy, normalizes representations where useful, validates and transforms information, coordinates downstream services, and records the resulting operational flow. The participating applications do not need to share the same protocol, data model, release cadence, hosting environment, or organizational owner.

This architectural position supports **modernization without disruption**. Rather than making a replacement program the precondition for progress, Avato allows organizations to stabilize current operations, expose durable service boundaries, replace systems incrementally behind those boundaries, and add new channels, partners, analytics, or AI participants without repeatedly rewriting every connected application.

Avato can be deployed as an intermediary for a single integration, as a domain-level orchestration layer, or as a distributed interoperability fabric across business units and organizations. It runs within customer-controlled infrastructure and can be deployed natively on an enterprise Java application server, in virtual machines, in containers, or in clustered environments. Its runtime is built from a stable core and reusable packages, interfaces, components, schemas, transforms, and environment-specific properties.

The central architectural idea is straightforward:

Move change into the layer designed to absorb change.

Application-specific code, database workarounds, and point-to-point mappings accumulate technical debt because every new participant changes the assumptions of existing participants. Avato isolates that variability in a declarative, observable, governed layer whose purpose is to translate, coordinate, and evolve.

Table of Contents

<i>Executive Technical Summary</i>	<i>2</i>
<i>The Modernization Problem</i>	<i>5</i>
<i>Where Avato Fits</i>	<i>6</i>
Typical Infrastructure Elements	7
Avato's Role Within Common Enterprise Infrastructure	7
<i>The Processing Model</i>	<i>8</i>
<i>Representation Independence</i>	<i>9</i>
<i>Configuration Model and Reusable Composition</i>	<i>10</i>
<i>Orchestration Patterns</i>	<i>11</i>
<i>Operations, Observability, and Failure Handling</i>	<i>12</i>
Message Sequences and Correlation	12
Inspection and Replay	12
Validation and Fault Isolation	12
Incident Notification and Undelivered Work	12
Masking and Audit	12
Operational Assurance	13
<i>Deployment and Lifecycle</i>	<i>13</i>
Consistent Runtime Architecture	13
Native Deployment	13
Containerized Deployment	15
Docker Deployment	15
Kubernetes Deployment	16
High Availability	17
Typical Enterprise Environment	19
<i>Security, Trust, and Governance</i>	<i>20</i>
<i>Artificial Intelligence in Enterprise Architecture</i>	<i>21</i>
AI as a Governed Participant	21
Controlled AI Participation	22
<i>From Tactical Integration to Enduring Enterprise Capability</i>	<i>22</i>

<i>Architectural Patterns Proven in Production</i>	24
Pattern 1 — Federating Independent Organizations.....	24
Pattern 2 — Progressive Core System Modernization.....	26
Pattern 3 — Controlled Modernization Cutover	27
Pattern 4 — Policy-Driven Trusted Information Exchange	29
<i>Architectural Principles</i>	30
Configuration-Driven Architecture	30
Representation Independence	30
Deterministic Orchestration and Governance	31
Operational Intelligence and Assurance	31
Independent Evolution	31
Incremental Modernization	31
Federated Enterprise Architecture	31
Accumulating Architectural Capability	31
<i>Conclusion</i>	32

The Modernization Problem

The objective of enterprise modernization is not merely to connect systems, but to enable their independent evolution. Applications, technologies, organizations, and infrastructure should be free to change at different rates without imposing unnecessary change upon one another.

Integration is often described as a connectivity problem: system A must call system B. In long-lived enterprises, the harder problem is coupling. Systems become coupled through payload structures, timing assumptions, database schemas, authentication methods, network locations, error semantics, release schedules, and undocumented operational behaviour.

This coupling creates predictable symptoms:

- a legacy application cannot be replaced because too many consumers depend on its internal contract;
- new digital channels require changes across multiple systems;
- one application's data model becomes an accidental enterprise standard;
- business rules are duplicated in applications, stored procedures, and scripts;
- synchronous dependencies make downstream outages visible to users;
- production incidents are difficult to reconstruct because the full message path is not recorded;
- each modernization initiative adds another temporary adapter that becomes permanent;
- AI initiatives cannot safely access authoritative operational information without bypassing established controls.

A replacement-first strategy attempts to remove coupling by replacing a large part of the ecosystem at once. That may be appropriate in some cases, but it creates a large dependency chain: data conversion, process redesign, interface replacement, organizational change, security approval, cutover, and rollback all become part of one program.

Avato takes a different approach. It reduces coupling before, during, and after replacement. Existing applications continue to operate while stable contracts and orchestrated processes are introduced around them. This gives the enterprise a controlled path from tactical interoperability to strategic modernization.

Where Avato Fits

Avato sits between participants that need to exchange information or coordinate work. It is not limited to a single architectural role.

As an intermediary, Avato receives a request in one protocol or representation and delivers it to another system in the form that system expects.

As a domain orchestration layer, Avato coordinates multiple applications and services to deliver a business outcome, while presenting a stable interface to consumers.

As a distributed interoperability fabric, multiple Avato instances can be positioned near systems, business domains, organizations, or security boundaries. Each instance can enforce local policy while participating in broader shared processes.

As an application platform, Avato can host selected web services, scheduled tasks, workflows, utilities, and user-facing functions where that is the simplest complete solution.

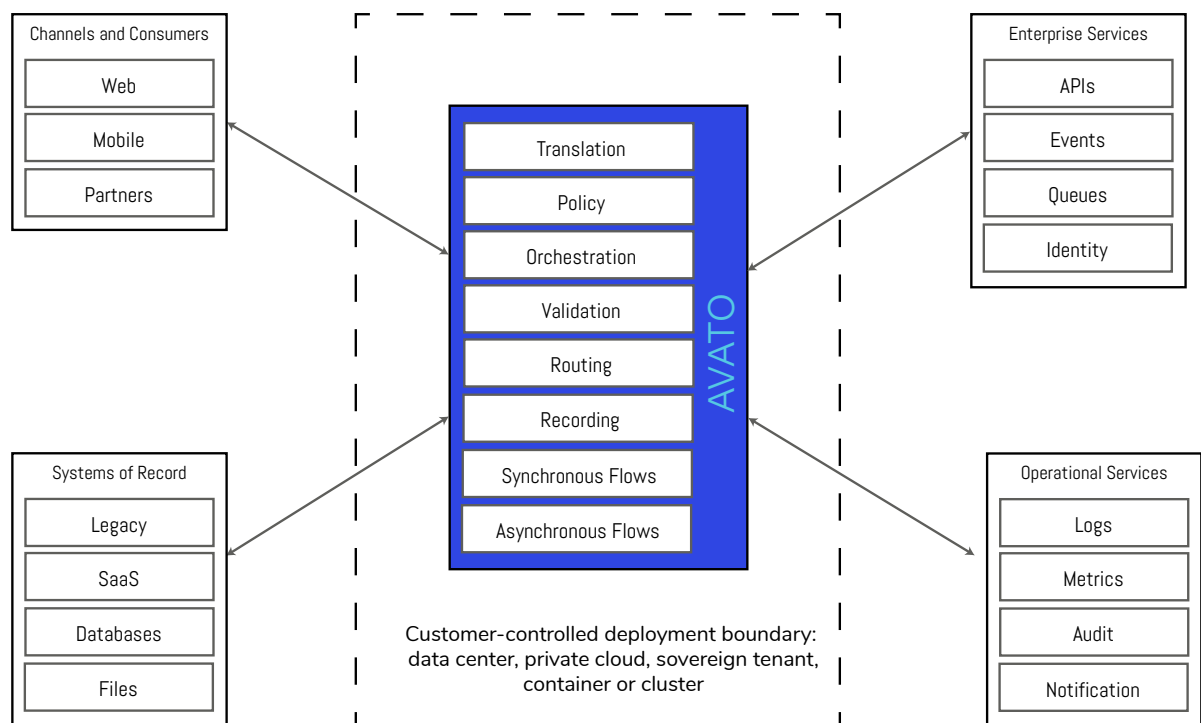


Figure 1 – Avato's architectural position between enterprise participants and operational services

It is not necessary to create a separate application merely to add a small governed capability to an orchestrated process.

The platform is intentionally deployment-independent. A customer may operate it in a data centre, private cloud, sovereign tenant, virtual machine, container environment, or highly available cluster. The architectural model remains the same: applications connect through standards-based endpoints; Avato applies configuration-driven behaviour; operational records are retained under customer control.

Typical Infrastructure Elements

A production deployment commonly includes:

- one or more Avato application instances;
- an enterprise Java runtime such as JBoss EAP or WildFly;
- a relational database for platform and operational data;
- load balancing or ingress services for clustered deployments;
- identity providers and enterprise directories;
- package and configuration repositories integrated with the customer's delivery process
- optional additional messaging infrastructure, complementing Avato's native asynchronous messaging capabilities where enterprise messaging platforms are already deployed¹;
- optional additional customer logging, monitoring, certificate, and key-management services.

Avato does not require an external hyperscale control plane. The customer determines where the runtime, configuration, operational records, and credentials reside.

Avato's Role Within Common Enterprise Infrastructure

One of the most common questions during modernization initiatives is whether Avato replaces existing infrastructure or complements it. In practice, Avato is designed to integrate with and enhance an organization's technology landscape, replacing only those custom integration capabilities that become unnecessary while coexisting with enterprise platforms that continue to provide value.

¹ Avato includes native messaging, persistence, routing, orchestration, monitoring and operational management services. Where organizations have standardized on enterprise messaging platforms such as IBM MQ, Kafka or cloud messaging services, Avato integrates with those technologies as peers rather than requiring them. This allows existing messaging infrastructure to be retained while avoiding unnecessary architectural dependencies.

Existing capability	Avato role
ESB	Replace or augment
API Gateway	Augment / partially replace
Integration code	Replace
Point-to-point interfaces	Replace
Model Gateway	Can become governed gateway
Workflow engine	Sometimes replace
Message broker	Integrates with rather than requires

The Processing Model

Avato processes requests and events through configurable component circuits. A circuit is a sequence or graph of reusable runtime components. Components receive a message, perform a focused responsibility, and pass a result to the next component or endpoint.

A representative synchronous flow is:

1. Receive. An endpoint accepts an HTTP, SOAP, REST, file, queue, scheduled, or other supported interaction.
2. Authenticate and authorize. The platform applies the configured inbound security method and access rules.
3. Record and correlate. The request is assigned or associated with correlation metadata and can be recorded as part of a complete message sequence.
4. Normalize. The input is converted into a structured representation suitable for deterministic processing.
5. Validate. Schemas and rules verify that required structure and content are present before downstream work occurs.
6. Transform and enrich. Declarative transforms adapt the request to the target contract, add contextual data, or construct one or more downstream messages.
7. Route and orchestrate. Rules choose destinations, invoke services, branch, aggregate results, or coordinate tasks and workflows.

8. Adapt the response. Downstream responses and faults are converted into the contract expected by the original consumer.
9. Observe and surface. The complete flow, timing, payload metadata, and faults are available for operational analysis subject to configured masking and retention.

The same model supports asynchronous delivery. A request can be accepted, persisted or queued, delivered later, retried, routed to an alternate destination, or surfaced as an undelivered message for investigation and resubmission.

Representation Independence

Avato does not require all enterprise systems to adopt one external canonical contract. Instead, it provides representation independence: information can be received and delivered in the form best suited to each participant while preserving meaning through a controlled internal processing model.

XML is a particularly effective internal representation for deterministic orchestration because it combines structured data, namespaces, schemas, validation, XPath addressing, and mature transformation standards. Avato uses XSLT and related standards to make transformations explicit, testable, versionable, and independent of application code.

JSON, form data, SOAP, REST, files, database results, and other representations can be converted at system boundaries. The resulting transformations are not hidden inside connector code. They are platform artifacts that can be inspected, tested, packaged, promoted, and reused.

This approach provides several architectural advantages:

- source and target applications remain independent;
- data contracts can evolve without synchronized releases;
- routing and policy can operate on a consistent structured message;
- transformations can be reused across channels and consumers;
- validation failures are detected before corrupt or incomplete data reaches downstream systems;
- the same logical capability can be exposed through multiple protocols or representations.

A canonical model is therefore a processing technique, not an enterprise-wide mandate. Avato can use a shared domain representation where it adds leverage, or perform direct source-to-target transformation where that is simpler.

Configuration Model and Reusable Composition

Avato separates a stable runtime from customer and domain configuration. The practical hierarchy is:

Platform → Profile → Packages → Interfaces → Components → Properties

The platform supplies the runtime, management functions, reusable components, security mechanisms, message recording, scheduling, and operational services.

Profiles package the Avato platform for a particular market, industry, or solution domain. Each profile combines a curated set of reusable packages, its own package marketplace, branding, documentation, and domain-specific capabilities to create a tailored starting point for a particular class of customer. Organizations can then extend the selected profile by installing additional packages, exposing interfaces, composing components, and configuring properties to meet their specific requirements.

Packages group related interfaces, schemas, transforms, configuration, documentation, tests, and dependencies into deployable units. Packages can represent a protocol, enterprise application, business domain, common utility, or complete solution capability.

Interfaces define coherent sets of listening and forwarding endpoints and the circuits that connect them.

Components perform focused functions such as dispatching, forwarding, transformation, validation, logging, recording, database access, file handling, queueing, mailing, delay, scheduling, or command execution.

Properties separate environment-specific values—addresses, credentials, paths, database connections, feature switches, and deployment parameters—from reusable package logic.

This separation is important. Customer solutions do not require customer-specific forks of the product. The same core runtime is assembled with different packages and properties for development, test, production, business units, or customers. Core Java development is normally required only when a genuinely new low-level component or protocol capability is needed. Most solution behaviour is delivered through configuration, schemas, transforms, and component composition.

The result is not “no code” as a slogan. It is a deliberate placement of code at the correct level. Stable runtime concerns belong in reusable components. Variable business and

integration behaviour belongs in declarative artifacts that can change without rebuilding the platform core.

Orchestration Patterns

Avato supports both request-response and event-driven patterns. Common patterns include:

- **Protocol adaptation:** SOAP to REST, JSON to XML, file to service, queue to HTTP, and related combinations.
- **Content-based routing:** choose a destination or processing path using message content, headers, identity, or policy.
- **Aggregation:** call multiple services and combine their responses into a single consumer-oriented result.
- **Multiplexing:** expose one stable interface while routing to different back-end implementations.
- **Asynchronous handoff:** accept work independently of downstream availability and deliver through durable messaging.
- **Retry and recovery:** retry anticipated failures, retain undelivered work, and support controlled resubmission.
- **Scheduled and conditional execution:** run tasks or workflows at configured times or when conditions are satisfied.
- **Human and system notification:** send operational notifications or integrate with collaboration services when defined events occur.
- **Service virtualization:** record, replay, and mock services to reproduce production behaviour during testing and troubleshooting.

The orchestration layer can start small. A single circuit may translate one legacy interface. Over time, the same stable boundary can coordinate more systems, expose more channels, or hide the replacement of a back-end application.

Operations, Observability, and Failure Handling

Interoperability is only useful when it can be operated. Avato treats message intelligence and operational assurance as architectural capabilities rather than afterthoughts.

Message Sequences and Correlation

The Sequencer can record the messages exchanged between the original requester, Avato components, external services, and the final response. A sequence retains ordering and timing so operators can see the complete path of a transaction rather than isolated log lines. Correlation identifiers and origin tracking support search and traceability across components.

Inspection and Replay

Authorized users can inspect request and response payloads, headers, timing, faults, and the components that handled them. Recorded sequences can be exported, imported, replayed, or used to reproduce production scenarios in controlled environments. Replay is particularly valuable when an incident depends on timing, message order, or a specific historical payload.

Validation and Fault Isolation

Validation can reject malformed or incomplete messages before they affect downstream systems. Fault handling can generate the response expected by the caller while separately recording technical detail for operators. Components can log, notify, queue, or route failures according to policy.

Incident Notification and Undelivered Work

Operational packages can send error notifications, aggregate repeated incidents to reduce alert fatigue, and preserve messages that could not be delivered. Operators can inspect and resubmit undelivered messages after the underlying issue is corrected.

Masking and Audit

Logging and recording can be configured to mask sensitive values. Access logs, audit records, user and role controls, and message history support accountability and forensic

investigation. Retention and exposure should be configured to the customer's privacy, security, and regulatory requirements.

Operational Assurance

Automated tests, synthetic transactions, scheduled verification, expected-traffic monitoring, and post-deployment validation can actively prove that critical integration paths remain functional. Monitoring shows what is happening; operational assurance verifies that intended business flows still work.

Deployment and Lifecycle

Avato is designed to integrate into existing enterprise environments rather than dictate them. Its runtime architecture supports multiple deployment models, from lightweight single-instance deployments to highly available clustered environments, while preserving a consistent operational model and configuration approach.

Consistent Runtime Architecture

Regardless of deployment model, the Avato runtime behaves consistently. Interfaces, transformations, orchestration logic, policies and operational capabilities are portable between deployments. Moving from a development workstation to Docker, Kubernetes or a clustered production environment does not require redesigning integrations or changing architectural patterns.

Native Deployment

Unlike many integration platforms that require dedicated infrastructure or proprietary runtime environments, Avato is deployed as a conventional enterprise application, allowing it to be managed using the same operational practices as other business applications.

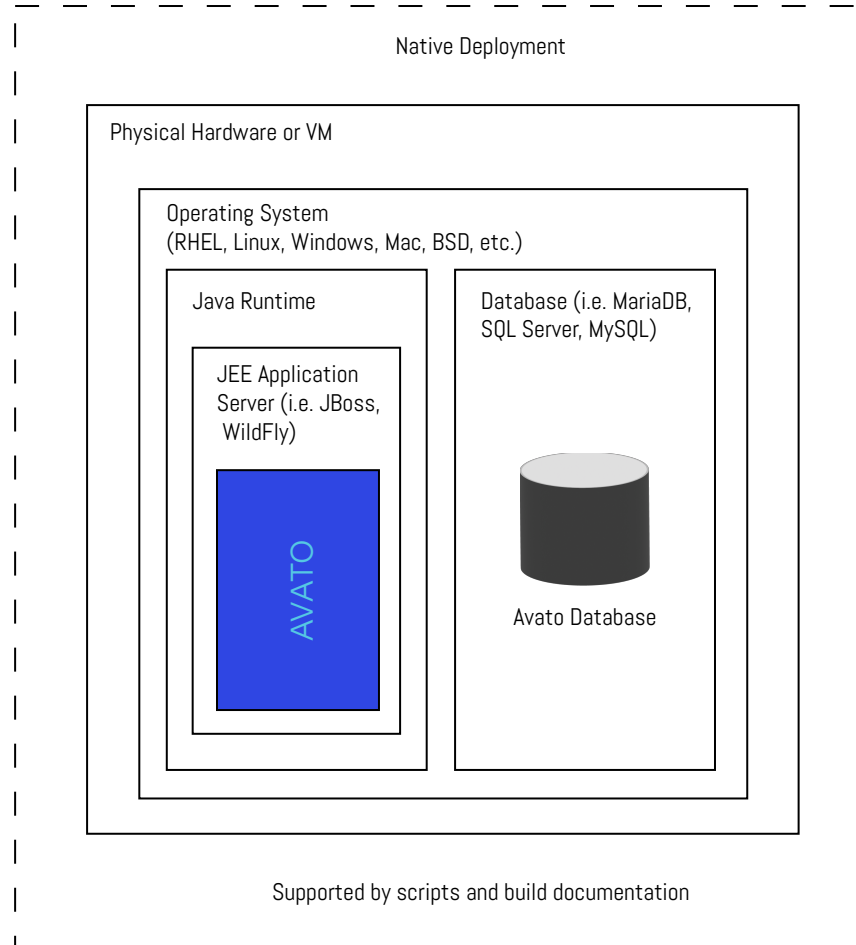


Figure 2 - Native Avato Deployments on physical hardware or virtual machines

For organizations with established enterprise Java infrastructure, or where support for low-level UDP socket or hardware serial port protocols are needed, Avato may be deployed directly within a supported application server environment. This approach allows the platform to integrate naturally with existing operational practices, including enterprise authentication, monitoring, backup, high availability, and database services, without introducing additional infrastructure requirements.

A native deployment typically consists of one or more Avato runtime instances connected to a relational database that stores configuration, operational metadata, message history, and other persistent platform information. Runtime behaviour, interfaces, orchestration logic, and management capabilities remain identical regardless of whether the platform is deployed natively or within containerized environments.

Native deployment is often appropriate where organizations have standardized operational practices, existing middleware infrastructure, or regulatory requirements that favour traditional application server deployments. Because Avato is implemented as a standard

enterprise application rather than a proprietary appliance, it can participate fully within existing operational and governance processes while requiring only the infrastructure necessary to support its intended workload.

Figure 2 above illustrates a typical native deployment.

Containerized Deployment

Docker Deployment

Containerized deployment using Docker provides a lightweight and consistent runtime environment suitable for development, testing, departmental deployments, embedded solutions, and smaller production environments. By packaging the complete Avato runtime together with its dependencies, Docker provides predictable behaviour across development, test, and production while simplifying installation and ongoing maintenance.

Figure 3 illustrates a typical Docker deployment consisting of one or more Avato containers communicating with an external relational database used for persistent configuration, operational data, message recording, and runtime state. The database may be deployed locally or provided as a managed service, depending on organizational standards and operational requirements.

Because the runtime itself remains unchanged, interfaces, transformations, orchestration logic, monitoring, and operational procedures developed within a Docker environment are directly portable to larger enterprise deployments. Docker therefore provides a convenient entry point for organizations adopting Avato while preserving a straightforward migration path as deployment requirements evolve.

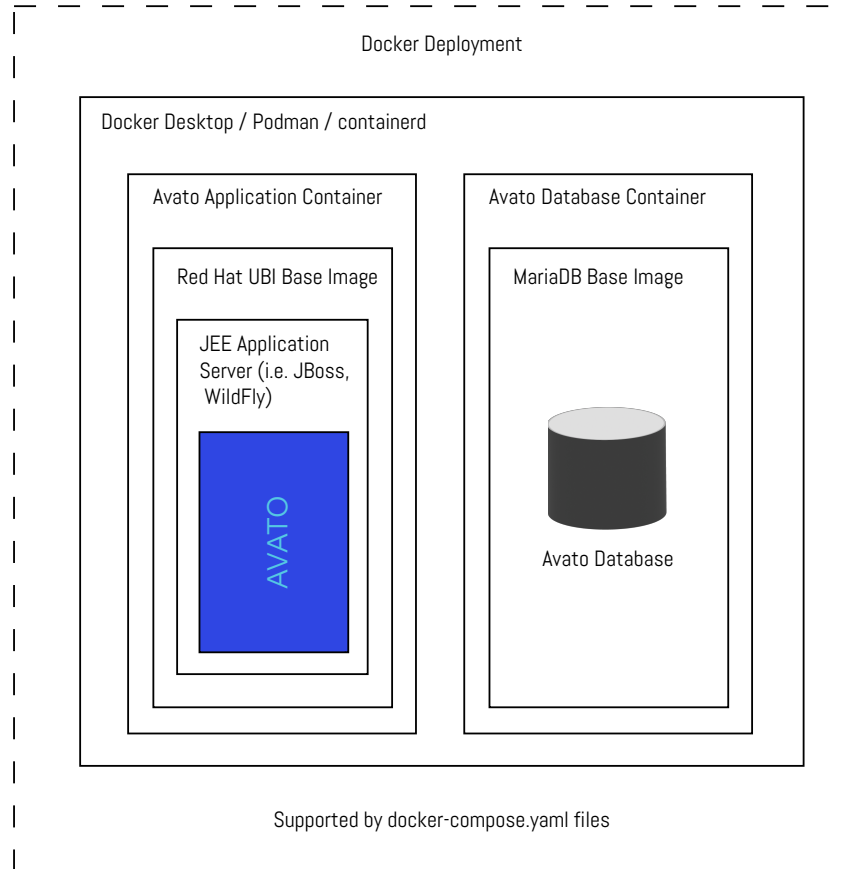


Figure 3 - Docker Deployment Model

Kubernetes Deployment

For enterprise-scale environments, Avato may be deployed within Kubernetes or compatible container orchestration platforms. Kubernetes provides automated scheduling, health monitoring, rolling upgrades, scaling, and self-healing capabilities while allowing multiple Avato runtime instances to operate as a coordinated service.

As shown in Figure 4, individual runtime instances remain stateless wherever practical, with durable configuration, operational metadata, and recorded message information maintained within shared persistent storage. This architecture enables runtime instances to be added, removed, or replaced without affecting the logical behaviour of the integration platform.

Deploying within Kubernetes allows Avato to participate naturally within cloud-native operational environments while remaining architecturally identical to smaller deployments. The integration assets, configuration, interfaces, and operational tooling remain consistent regardless of the underlying orchestration platform, allowing organizations to evolve their infrastructure independently of their integration architecture.

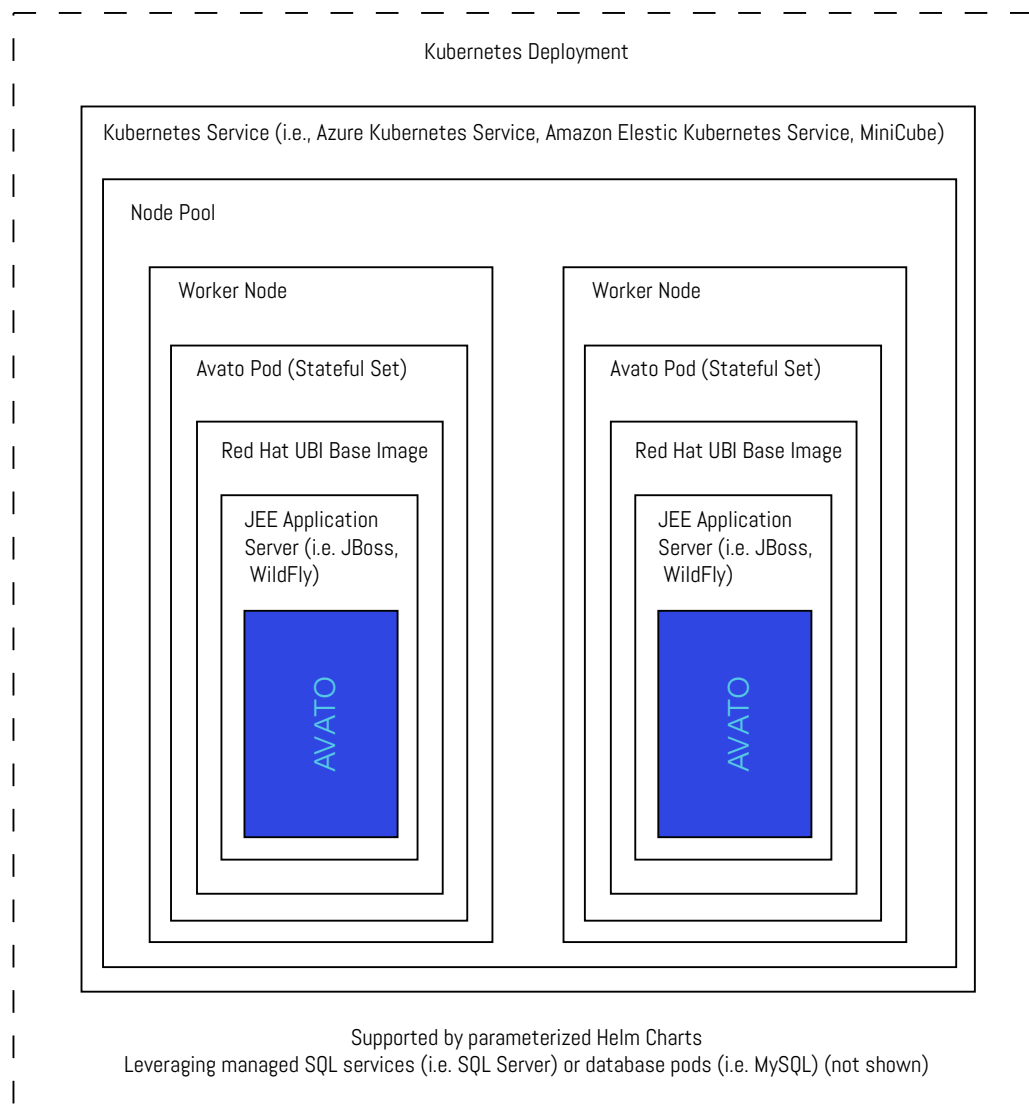


Figure 4 - Kubernetes Deployment Model

High Availability

Enterprise integration platforms frequently become critical operational infrastructure, supporting business processes that cannot tolerate extended outages or single points of failure. For these environments, Avato supports highly available deployments consisting of multiple runtime instances operating as a coordinated cluster behind standard network load balancing infrastructure.

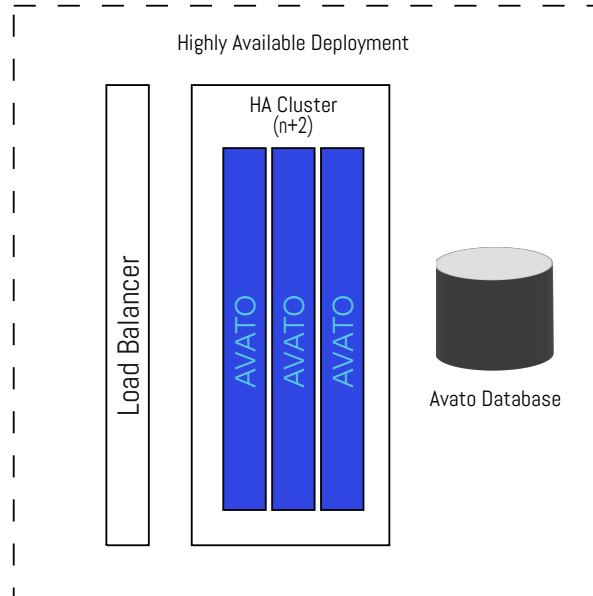


Figure 5 - High Availability Deployment Model

Each runtime instance executes the same deployed interfaces, orchestration logic, transformations, and operational policies, allowing work to be distributed across the cluster while presenting a single logical integration platform to connected systems. Persistent configuration, operational metadata, message recording, and other shared runtime information are maintained within a common relational database, ensuring consistent behaviour regardless of which runtime instance processes a particular transaction.

Because the platform is designed to operate as multiple cooperating runtime instances rather than a single monolithic server, individual instances may be restarted, upgraded, or replaced with minimal disruption to normal operations. Planned maintenance and infrastructure changes can therefore be performed incrementally, while unplanned failures are isolated through standard enterprise clustering and failover mechanisms.

Figure 5 above illustrates a typical highly available deployment within a single operational environment. Larger enterprise deployments may extend these same principles across multiple data centres or cloud regions as part of broader disaster recovery and business continuity strategies.

High availability addresses runtime resilience, while the deployment and promotion model described in the following section provides resilience across the software delivery lifecycle.

Typical Enterprise Environment

While Avato supports a variety of deployment models, enterprise implementations typically follow a consistent operational lifecycle that separates development, testing, production, and disaster recovery environments. This separation enables new integrations, enhancements, and operational changes to be developed, validated, and promoted through established governance processes before entering production.

Packages containing interfaces, transformations, orchestration logic, policies, and supporting configuration are developed and tested in lower environments before being promoted through quality assurance and into production. Because the same runtime architecture is used throughout each stage of the lifecycle, the behaviour of an integration remains consistent as it progresses from development through production deployment, reducing the operational risks commonly associated with environment-specific implementations.

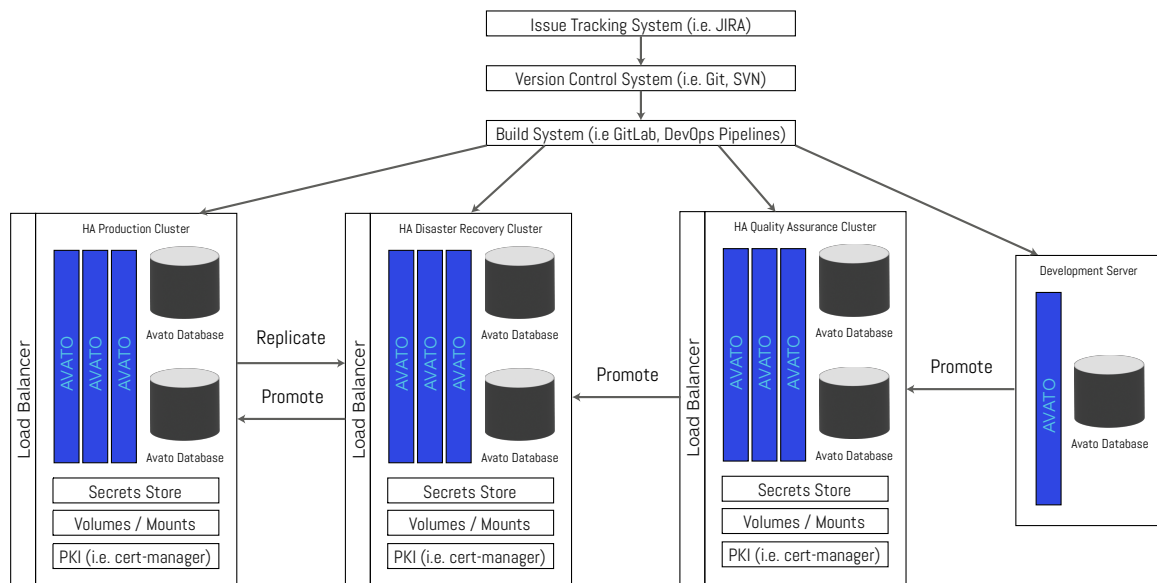


Figure 6 - Typical Enterprise Deployment Model

Production environments are commonly complemented by geographically separate disaster recovery environments, allowing organizations to align Avato with existing business continuity strategies and recovery objectives. Database replication, configuration management, certificate management, monitoring, and backup processes are typically

integrated with existing enterprise operational services rather than being managed independently by the platform.

Figure 6 above illustrates a representative enterprise deployment, demonstrating how development, quality assurance, production, and disaster recovery environments participate in a common operational lifecycle while remaining integrated with enterprise infrastructure services. The exact topology varies according to organizational requirements, but the underlying deployment principles remain consistent across organizations of different sizes and levels of operational maturity.

Security, Trust, and Governance

Avato is designed for environments where integration crosses security and organizational boundaries. Security is applied at the intermediary layer so that legacy systems do not each need to implement every modern control directly.

Supported patterns include:

- TLS and certificate-based server communication;
- configurable inbound authentication methods;
- basic and digest authentication where required by legacy endpoints;
- OAuth 2.0 integration and delegated access patterns;
- LDAP and Active Directory integration;
- role-based access to platform functions;
- IP, method, and endpoint restrictions;
- encrypted configuration values and integration with external key-management services where configured;
- audit logging and privacy-oriented log masking;
- schema validation and protections against common XML and web injection risks.

Customer-controlled deployment is a governance capability as well as an infrastructure choice. The customer determines data residency, network boundaries, identity integration, retention, and administrative access. In federated scenarios, each organization can retain local authority while exchanging only the information and decisions required by the shared process.

Artificial Intelligence in Enterprise Architecture

Enterprise architectures increasingly incorporate artificial intelligence to classify information, extract meaning, generate content, and assist decision-making. These capabilities introduce probabilistic reasoning into environments that have traditionally depended upon deterministic processes, policy enforcement, and authoritative systems of record.

Rather than replacing enterprise integration, AI increases the need for a governed architectural layer capable of providing trusted information, enforcing policy, controlling access to enterprise capabilities, and maintaining complete operational traceability. Within this architecture, AI becomes another participant in enterprise workflows rather than the workflow itself.

AI as a Governed Participant

Large language models and autonomous agents perform probabilistic inference. They benefit from authoritative context, bounded capabilities, deterministic execution, and comprehensive auditability. Avato provides the architectural layer through which models and agents can interact safely with enterprise systems without requiring unrestricted access to the underlying applications.

Within this pattern:

- Avato exposes stable, purpose-specific services rather than direct access to enterprise systems.
- Identity, authorization, and policy determine which operations and information are available.
- Information is collected, transformed, and presented in the representation required by the model.
- Responses may be validated, enriched, correlated, routed for human approval, or translated into deterministic downstream actions.
- Requests, responses, decisions, and resulting system activity remain observable and auditable throughout the process.

Avato is not positioned as the AI model itself. Instead, it provides the deterministic integration, governance, and operational control upon which enterprise AI solutions can be safely constructed.

Controlled AI Participation

While some organizations may choose to expose enterprise capabilities directly to autonomous agents through tool-oriented interfaces, many regulated and mission-critical environments require a more constrained adoption model. In these environments, AI is introduced as one bounded service within a deterministic workflow rather than as an autonomous controller of the workflow.

Avato orchestrates the surrounding business process, determining when an AI capability should be invoked, what information should be provided, how its response should be interpreted, and what actions may follow. The platform remains responsible for identity, policy enforcement, routing, validation, exception handling, and access to downstream systems, while the AI service contributes only the specific inference for which it has been invoked.

This separation of probabilistic intelligence from deterministic execution allows organizations to incorporate AI capabilities without relinquishing operational control. As governance requirements evolve, the same architectural approach can accommodate increasingly autonomous interaction patterns while preserving enterprise oversight, security, and auditability.

From Tactical Integration to Enduring Enterprise Capability

Most integration initiatives begin by solving an immediate business problem: connecting systems, exposing services, supporting modernization, or enabling new digital capabilities. While these projects deliver immediate value, they also create an opportunity to build lasting architectural capability.

Avato is designed so that each implementation contributes reusable enterprise assets rather than isolated project deliverables. Interfaces, transformations, orchestration logic, policies, operational patterns, and supporting components are packaged as modular capabilities that can be extended and reused across future initiatives. As additional systems and business processes are introduced, organizations build on these assets instead of recreating them.

This changes the economics of modernization. Rather than repeatedly rebuilding integrations as applications and technologies evolve, organizations continue investing in an architectural layer designed to absorb change. Existing systems can be upgraded, replaced, migrated, or retired independently while the surrounding interoperability capabilities continue to evolve.

Over time, the platform becomes more than an integration solution—it becomes an enduring enterprise capability. Each project leaves behind reusable architectural knowledge that accelerates the next initiative, reduces implementation effort, lowers operational risk, and increases organizational adaptability.

This reflects a fundamental principle of enterprise architecture: every modernization initiative should leave the enterprise with greater capability than it had before. By treating interoperability assets as reusable enterprise capabilities rather than disposable project artefacts, organizations establish a modernization foundation that continues to deliver value long after the original project has been completed.

Architectural Patterns Proven in Production

The architectural principles described throughout this paper have been applied across large-scale modernization programmes in financial services, healthcare, government, and other regulated industries. While each organization faced unique operational and technical constraints, several recurring architectural patterns emerged. Rather than representing industry-specific solutions, these patterns illustrate common approaches to reducing coupling, enabling independent evolution, and delivering modernization incrementally while preserving operational continuity.

Pattern 1 — Federating Independent Organizations

Architectural challenge

Large enterprises and public-sector organizations frequently comprise multiple autonomous organizations that must cooperate operationally while retaining independent governance, technology choices, and operational practices. Replacing existing applications with a single enterprise platform is often impractical due to cost, operational risk, or organizational independence.

Architectural approach

Rather than attempting to standardize applications, each participating organization retained its existing clinical, inventory management, and procurement systems while Avato provided the integration layer responsible for information exchange, orchestration, transformation, reliable delivery, validation, and operational visibility.

Canonical representations isolated each organization's internal data models from those of its partners, allowing systems to evolve independently while maintaining consistent inter-organizational communication. Operational policies, routing logic, exception handling, and monitoring were centralized within the integration layer rather than embedded within individual applications.

Outcome

The resulting architecture enabled multiple independent health authorities to participate in a coordinated provincial supply chain without requiring replacement of their existing systems or introducing tight coupling between organizations. Existing investments were preserved while new capabilities—including automated ordering, inventory visibility, workflow coordination, and operational reporting—were introduced incrementally through the integration layer.

More importantly, the architecture established a durable integration foundation capable of supporting future modernization initiatives without requiring participating organizations to adopt common applications or synchronize technology lifecycles.

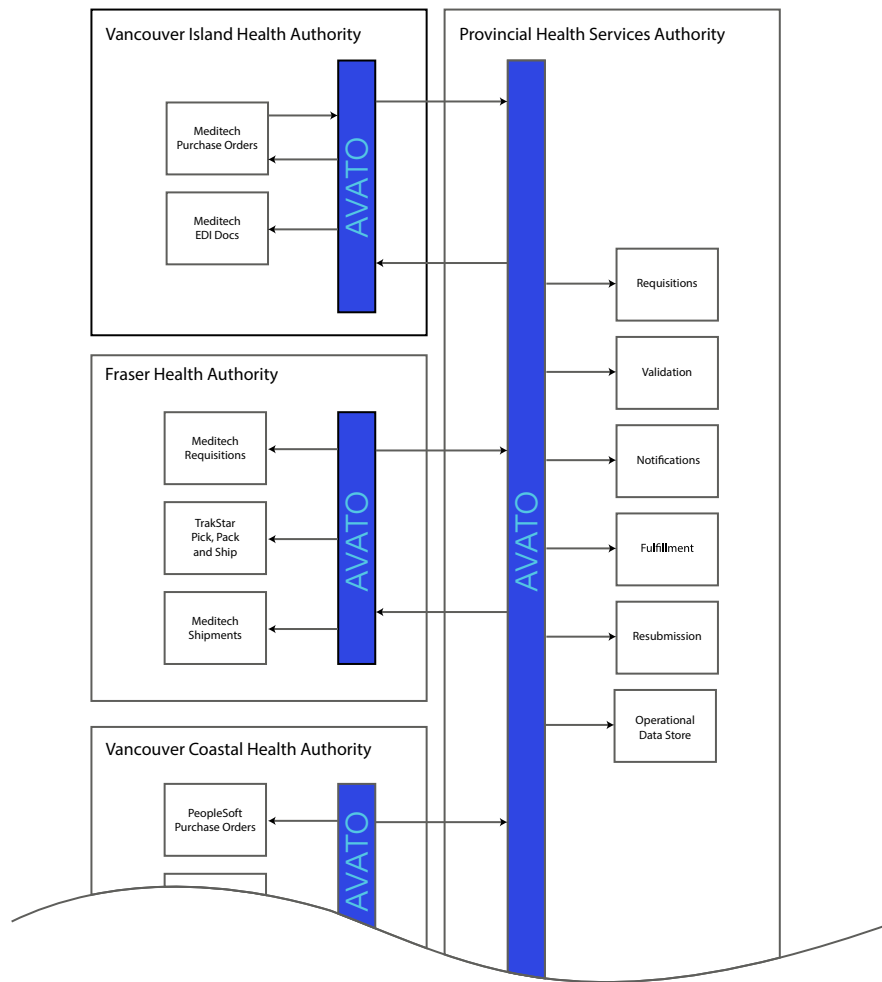


Figure 7 - Inter-organizational Hospital Supply Orchestration at BC Health Authorities

Pattern 2 — Progressive Core System Modernization

Architectural challenge

Core business systems often represent decades of accumulated investment, operational knowledge, and business logic. Replacing these systems outright introduces significant technical, operational, and organizational risk, while maintaining them indefinitely can limit an organization's ability to adopt new technologies and business capabilities.

The challenge is therefore not simply replacing a system, but enabling new capabilities to be introduced while existing systems continue to operate safely and predictably throughout the modernization programme.

Architectural approach

Rather than coupling new applications directly to the existing core banking platform, Avato established a stable integration layer that isolated applications from the implementation details of individual systems. Canonical business representations, transformation logic, interface contracts, and orchestration policies were defined independently of the core application's internal data structures and communication mechanisms.

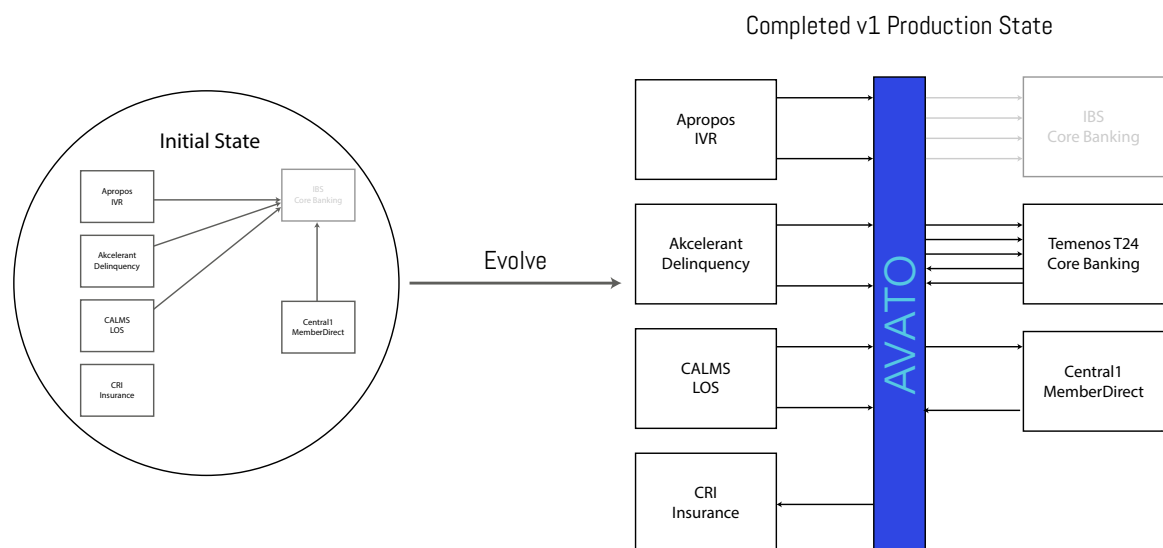


Figure 8 - Core Banking Modernization Without Disruption

As modernization progressed, these reusable architectural assets became the preferred integration point for new projects. Additional channels, services, and business capabilities were connected through the existing integration layer, allowing successive initiatives to build upon established interfaces and business representations rather than creating new point-to-point integrations.

Because the integration layer remained stable while endpoint systems evolved, individual applications could be enhanced, replaced, or introduced independently without requiring widespread changes across the surrounding enterprise architecture.

Outcome

The modernization programme evolved incrementally rather than through a single large-scale replacement initiative. Each successive project expanded the organization's reusable architectural capability while reducing the effort required for future integrations.

Over time, the integration layer became a strategic enterprise asset rather than a project-specific solution. Canonical business models, transformation logic, operational policies, and interface definitions were reused across multiple modernization initiatives, allowing the organization to continue evolving its technology landscape without repeatedly rebuilding the surrounding integration architecture.

This pattern demonstrates that modernization need not be treated as a sequence of isolated replacement projects. By investing in durable architectural assets that survive individual technology changes, organizations establish an enduring modernization foundation capable of supporting successive generations of applications and business capabilities.

Pattern 3 — Controlled Modernization Cutover

Architectural challenge

Large-scale modernization programmes eventually reach a point where an existing operational system must be replaced or a new capability introduced into live production. Traditional cutover approaches often require synchronized application changes, extended outage windows, complex rollback procedures, and significant operational risk, particularly where numerous dependent systems must transition simultaneously.

The architectural challenge is to introduce new capabilities while minimizing disruption to business operations and preserving the ability to respond quickly if unexpected issues arise.

Architectural approach

Because Avato had already become the organization's established integration layer through earlier modernization initiatives, the contact centre application was introduced alongside the existing operational environment rather than replacing it outright. Existing interfaces, canonical business representations, orchestration logic, and operational policies were reused, allowing both legacy and replacement systems to coexist while the surrounding enterprise architecture remained unchanged.

The integration layer isolated dependent systems from the implementation details of the contact centre application, enabling the new platform to be introduced incrementally without requiring widespread changes across consuming applications. Operational routing, transformation, monitoring, and exception handling continued to be managed centrally, allowing the transition to be planned and executed as a controlled operational event rather than a wholesale architectural change.

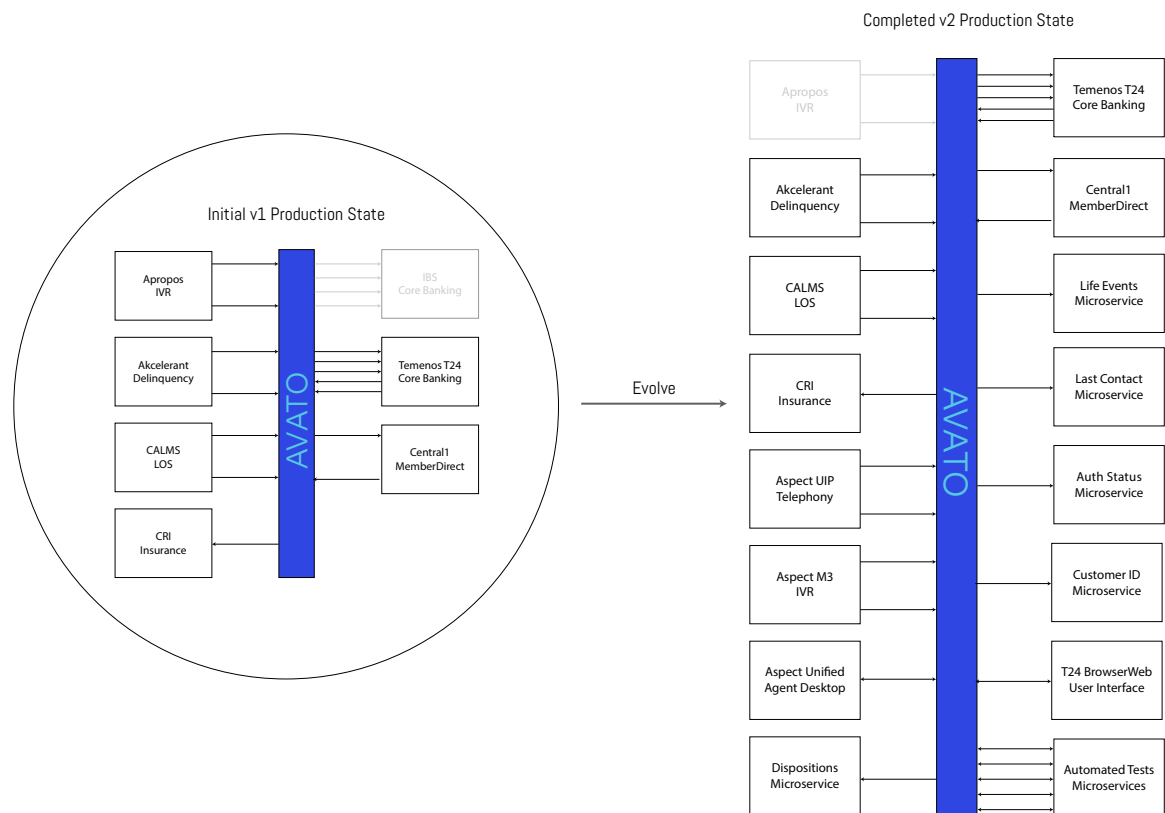


Figure 9 - Contact Center and Telephone Banking Modernization Building on Prior Works

Outcome

The production cutover was completed with only a brief interruption to customer services, demonstrating how an established integration architecture can substantially reduce the operational risk associated with major system replacement.

More significantly, the successful transition was not the result of a carefully scripted one-time migration alone. It reflected years of investment in reusable architectural assets that had progressively decoupled enterprise systems from one another. Because the surrounding architecture remained stable, replacing a mission-critical application became an incremental operational change rather than a disruptive enterprise-wide integration project.

This pattern illustrates an important principle of enterprise modernization: organizations reduce implementation risk not only through careful project planning, but by establishing an architecture in which individual systems can evolve independently over time.

Pattern 4 — Policy-Driven Trusted Information Exchange

Architectural challenge

Organizations increasingly exchange information across organizational, regulatory, and security boundaries. These exchanges often require more than protocol conversion or data mapping; they demand consistent application of business policy, privacy controls, data validation, auditing, and information governance while preserving the integrity and traceability of the exchanged information.

The challenge is to establish trusted information exchange without embedding complex governance logic within every participating application.

Architectural approach

Avato provided a policy-driven integration layer responsible for validating, transforming, enriching, and governing information as it moved between heterogeneous systems. Business rules defining data quality, privacy, masking, representation, and routing were implemented within the integration architecture rather than distributed across individual applications, allowing governance to be applied consistently regardless of the source or destination systems.

Information from multiple independent systems was normalized into common business representations before policy was applied, enabling consistent treatment of sensitive information despite differences in application design, data structures, and implementation technologies. The same architectural approach supported validation, transformation, audit, exception management, and reliable delivery as integral parts of the information exchange process rather than as isolated application-specific functions.

Outcome

The resulting architecture enabled organizations to exchange regulated information with confidence while preserving privacy, data quality, and operational traceability. Governance policies became reusable architectural assets rather than project-specific implementations, allowing new information exchanges to inherit established security, validation, and operational controls instead of recreating them.

This pattern demonstrates that enterprise integration is not simply concerned with connectivity. As organizations modernize and collaborate across increasingly complex

ecosystems, the integration layer becomes a trusted point of policy enforcement, information governance, and operational assurance that continues to evolve independently of the participating applications.

The recurring theme across these patterns is not integration itself, but the deliberate reduction of coupling between systems, organizations, technologies, and successive generations of enterprise architecture. By establishing a stable modernization layer that accumulates reusable architectural capability over time, organizations gain the freedom to evolve their technology landscape incrementally while preserving operational continuity, governance, and trust.

Architectural Principles

Enterprise integration platforms are frequently evaluated by the number of protocols they support, the breadth of available connectors, or the features provided by their runtime environments. While these capabilities remain important, they do not by themselves determine whether an architecture will continue to support an enterprise as its applications, infrastructure, and business requirements evolve.

The architectural approach described throughout this paper is founded on a different set of priorities. Rather than optimizing individual integrations, Avato is designed to establish stable architectural boundaries that reduce coupling, preserve reusable enterprise knowledge, and enable independent evolution over time. The following principles distinguish this approach.

Configuration-Driven Architecture

Business behaviour is expressed through declarative configuration, reusable components, orchestration, transformation rules, and policy rather than application-specific implementation wherever practical. This allows solutions to evolve incrementally while remaining understandable, governable, and reusable across successive initiatives.

Representation Independence

Applications communicate through stable business representations rather than proprietary internal data structures. Separating business semantics from application implementation reduces the impact of technology change and enables systems to be replaced or modernized without requiring widespread changes across the surrounding enterprise architecture.

Deterministic Orchestration and Governance

Enterprise workflows remain deterministic, observable, and policy-driven, even when incorporating external services such as artificial intelligence. Identity, authorization, routing, validation, audit, and operational control remain explicit architectural responsibilities rather than implicit characteristics of individual applications or external platforms.

Operational Intelligence and Assurance

Integration is treated as an operational capability rather than a transport mechanism. Recording, correlation, monitoring, replay, auditing, and diagnostic capabilities provide the visibility required to operate business-critical integration environments while supporting governance, compliance, and continuous improvement.

Independent Evolution

Deployment infrastructure, messaging technologies, cloud platforms, and connected applications are treated as replaceable implementation choices rather than architectural dependencies. Organizations remain free to evolve their infrastructure and technology landscape without redesigning the surrounding integration architecture.

Incremental Modernization

Modernization is approached as a continuous architectural process rather than a single transformation event. Existing systems may continue to deliver business value while new capabilities are introduced incrementally, allowing operational risk to be reduced without constraining future evolution.

Federated Enterprise Architecture

Independent organizations and autonomous business domains can collaborate through shared architectural services without requiring common applications, synchronized technology lifecycles, or centralized ownership. The integration layer enables cooperation while preserving the independence of participating systems and organizations.

Accumulating Architectural Capability

Every modernization initiative contributes reusable architectural assets that simplify those which follow. Interfaces, canonical business representations, transformation logic, orchestration policies, operational practices, and governance become enduring enterprise capabilities rather than disposable project artefacts. As organizations undertake successive modernization programmes, this architectural foundation grows in both scope and value, reducing implementation effort, lowering operational risk, and accelerating future change.

Together, these principles reflect a broader philosophy of enterprise architecture. Conventional integration projects often optimize the first successful connection between systems. Avato instead seeks to optimize the continuing evolution of the enterprise itself. By reducing coupling, preserving architectural knowledge, and establishing stable boundaries

between independently evolving systems, organizations gain the freedom to modernize incrementally while maintaining operational continuity, governance, and trust.

Conclusion

Enterprise modernization is often described as a technology challenge, yet the greatest difficulties rarely arise from technology itself. They arise from the growing interdependence of systems, organizations, and business processes that have evolved over many years. Replacing one application frequently affects many others, increasing the cost, complexity, and risk of every subsequent change.

Avato addresses this challenge by establishing a stable architectural layer between independently evolving systems. Rather than embedding knowledge within individual applications or creating increasingly complex point-to-point integrations, business representations, orchestration, governance, operational policy, and integration logic become reusable enterprise assets that survive successive generations of technology.

As organizations undertake additional modernization initiatives, these assets continue to grow in value. New capabilities are built upon an existing architectural foundation rather than replacing it, allowing systems, infrastructure, deployment models, and emerging technologies—including artificial intelligence—to be introduced incrementally while preserving operational continuity.

The production patterns presented in this paper demonstrate that this architectural approach has been successfully applied across large-scale modernization programmes involving healthcare, financial services, and other complex enterprise environments. Although each organization faced different technical and operational challenges, the underlying architectural principles remained consistent: reduce unnecessary coupling, establish stable architectural boundaries, preserve reusable enterprise knowledge, and enable continuous modernization through incremental change.

Ultimately, the purpose of enterprise integration is not simply to connect systems. It is to create the architectural conditions that allow an enterprise to evolve with confidence.