ASCENDION

AAVA™: Agentic COBOL Modernization

A collaborative modernization model with AWS Transform, AWS Kiro, and Ascendion AAVA

This whitepaper is prepared by Ascendion for informational and collaborative engagement purposes. It presents a reference architecture for COBOL modernization in which AWS Transform provides the foundational modernization analysis and conversion layer, AAVA extends the downstream engineering workflow, and AWS Kiro can support spec-driven finishing activities where appropriate.

Descriptions of platform roles reflect Ascendion's current point of view based on hands-on knowledge, solution design work, and engineering analysis. Product capabilities, naming, packaging, and recommended usage may evolve over time and should be validated from time to time.

Abstract

AWS Transform represents a major step forward in AI-assisted COBOL modernization. It brings machine-scale analysis, documentation, decomposition support, and automated code transformation for a problem that has historically required large teams and long timelines. For many enterprises, that foundation changes what is practical to modernize and how quickly modernization can begin.

This whitepaper presents AAVA as a complementary solution that builds on top of that foundation. AAVA is positioned as a downstream enrichment layer that helps enterprises convert modernization output into a fuller delivery package: clearer architecture artefacts, product-ready SDLC documentation, idiomatic Spring Boot implementation patterns, microservice decomposition guidance, governance-aware code generation, and test acceleration.

Where teams adopt AWS Kiro, the combined model can extend further into spec-driven development, scaffolding, build validation, and developer productivity. Together, AWS Transform, AAVA, and Kiro can be viewed as a staged modernization pipeline: Transform for foundational analysis, AAVA for engineering enrichment and enterprise hardening, and Kiro for developer-centric completion and validation workflows.

This document is intended for enterprise architects, engineering leaders, modernization program sponsors, and AWS-aligned delivery teams evaluating how multiple tools can work together in a coherent modernization process.

Contents

1. Introduction 04
2. Enterprise Modernization Landscape 05
3. AWS Transform as the Foundation 06
4. Where AAVA Adds Complementary Value 07
5. The Role of AWS Kiro 08
6. Platform Architecture - The Big Picture 09
7. Modernization Delivery Methodology 10
8. Deep Dive - AAVA Platform Agents 11
9. Conclusion 12
10. Glossary of Terms 13

Introduction

The world still runs a remarkable share of its critical business logic on mainframes. Banking transactions, insurance processing, claims adjudication, public-sector benefit systems, and high-volume batch operations continue to rely on decades of accumulated COBOL and mainframe-era design. These systems are often dependable, but they are also expensive to change, difficult to document, and increasingly hard to staff.

Modernization therefore cannot be treated as a code-conversion exercise alone. Enterprises need a path from legacy source to a target estate that cloud and Java engineering teams can govern, understand, test, extend, and operate. That requires more than syntax transformation. It requires reverse engineering, architecture interpretation, SDLC artefacts, code quality controls, data migration thinking, and disciplined execution.

In that context, AWS Transform is highly significant. It creates the foundation that many enterprises have historically lacked: a scalable way to analyze large COBOL estates, surface dependencies, organize modernization waves, and produce modernization outputs at machine speed. The opportunity for partners is not to compete with that foundation, but to amplify it.

This paper explains how AAVA can help achieve that. AAVA is a complementary layer that starts where the core modernization engine creates momentum and then extends the workflow into enterprise-ready outcomes. The result is a stronger joint story: AWS Transform establishes the modernization baseline, AAVA increases delivery depth and adoption readiness, and Kiro can help teams move faster through spec-driven completion activities.

Enterprise Modernization Landscape

COBOL modernization remains one of the most consequential technology shifts in large enterprises because it sits at the intersection of cost, resilience, agility, talent availability, and time-to-change. Organizations are not modernizing simply because legacy code is old; they are modernizing because the surrounding operating model has become harder to sustain.

Three realities define the landscape. First, the business logic in these systems is mission critical and deeply entangled with batch schedules, file structures, and operational dependencies. Second, the people who understand that logic are retiring faster than most organizations can replenish equivalent expertise. Third, infrastructure modernization alone does not deliver the full business outcome if the application estate remains structurally difficult to evolve.

That is why AI-assisted modernization is increasingly important. It creates leverage in the earliest and most expensive stages of the lifecycle: program comprehension, dependency analysis, domain grouping, documentation, and initial code transformation. The more complete and trustworthy that foundation becomes, the more practical it is for downstream engineering teams to take over and move at speed.

AWS Transform as the Foundation

AWS Transform should be viewed as the foundational modernization engine in the combined approach described in this paper. Its primary contribution is to industrialize the reverse engineering and initial transformation stages that have historically slowed or blocked mainframe programs.

For COBOL modernization, AWS Transform can provide high-value outputs such as code and dependency analysis, business logic documentation, domain-oriented decomposition support, and modernization wave planning. It also creates a path for automated transformation of legacy assets into modern target technologies while preserving the business intent embedded in the original system.

Equally important, AWS Transform establishes an excellent starting point for partner-led enrichment. Once dependency graphs, documentation, program relationships, and initial transformed artefacts exist in a structured form, specialist platforms such as AAVA can operate on much stronger inputs than raw COBOL alone. That is the heart of the force-multiplier message.

In other words, the combined model begins by accepting AWS Transform as the acceleration engine and then asking how additional tooling can help enterprises get more value from that acceleration.

Where AAVA Adds Complementary Value

AAVA's role is best described as downstream engineering enrichment. It takes the momentum created by AWS Transform and extends it into the artefacts, code conventions, architecture patterns, and governance controls that enterprise delivery teams typically need before they can treat the modernization output as program ready.

Review Role	AWS Transform Contribution	AAVA Complementary Extension	Outcome
Architecture and portability	AWS Transform establishes the modernization baseline through analysis, decomposition guidance, and transformed output that accelerates the move away from legacy operating models.	AAVA can extend that baseline by shaping the target into clean Spring Boot structures, decomposition-aware services, shared utility modules, and deployment patterns aligned to the enterprise landing zone.	A stronger bridge from modernization output to maintainable cloud-native application architecture.
SDLC artefacts	AWS Transform provides the foundational technical understanding of the codebase, which is the hardest prerequisite for producing downstream artefacts at scale.	AAVA can convert that understanding into epics, user stories, acceptance criteria, design documents, API contracts, and implementation-ready technical documentation.	Better alignment between engineering, architecture, product ownership, and governance stakeholders.
Code quality and maintainability	AWS Transform greatly reduces the effort required to reach a modern language target and gives teams a working modernization starting point.	AAVA can further refactor for idiomatic Java and Spring practices, clearer naming, shared patterns, and engineering-standard enforcement.	A codebase that is easier for Java-native teams to read, review, extend, and support.
Data and integration	AWS Transform contributes essential insight into data structures, flows, and system relationships that are necessary for downstream modernization decisions.	AAVA can interpret those insights into domain-oriented entities, integration contracts, migration-ready schemas, and service-facing data models.	A cleaner path from legacy data representations to operationally usable modern application models.

Enterprise modernization is naturally multi-stage, and different tools can contribute strength at different layers. In that model, AAVA acts as a force-multiplier by helping enterprises operationalize and productionize the value that AWS Transform unlocks early in the lifecycle.

The Role of AWS Kiro

AWS Kiro can play an important role as the developer-facing execution environment within the broader COBOL modernization workflow, particularly for organizations that adopt a spec-driven engineering model. In this architecture, Kiro is not positioned as the source of modernization discovery or transformation logic. Instead, it becomes the environment where the outputs of modernization intelligence are translated into structured engineering work and ultimately into running software.

Modernization initiatives often generate many artefacts: reconstructed architecture views, program dependency maps, derived functional specifications, interface definitions, test cases, and partially generated application components. Platforms such as AAVA can produce many of these structured outputs during the discovery and preparation phases. Kiro provides the environment in which these artefacts can be organized, interpreted, and carried forward into implementation.

Within this workflow, Kiro can support several important developer activities. It enables teams to align implementation work with structured specifications derived from legacy systems, ensuring that the intent captured during discovery and transformation phases is preserved during development. Developers can use these specifications as the foundation for generating service scaffolding, refining application structure, completing partially generated code, and implementing integration logic required for the modernized system.

Kiro can also support verification and validation activities. As modernization artefacts are translated into working components, Kiro provides a structured context for implementing validation steps such as functional verification, specification alignment, and developer-driven refinement of generated code. This helps ensure that the resulting applications remain consistent with both the original business logic encoded in COBOL programs and the architecture patterns expected in modern cloud-native systems.

From an architectural perspective, Kiro therefore acts as the spec-driven execution layer of the modernization pipeline. While AWS Transform performs the automated transformation of COBOL logic and AAVA enriches the process through discovery and modernization intelligence, Kiro helps developers operationalize these outputs within day-to-day engineering workflows. The result is a modernization process that not only converts legacy systems but also integrates them into a structured, maintainable development lifecycle aligned with modern cloud engineering practices.

For public-facing usage, specific product claims should be validated against the current Kiro feature set and approved AWS messaging. The durable architectural message remains consistent: Kiro can serve as the developer workspace that helps teams translate modernization-related intent into working implementations within a specification-driven development model.

Platform Architecture – The Big Picture

The combined AWS Transform + AAVA + Kiro architecture is best understood as a layered pipeline in which each platform has a distinct role and does not need to over extend beyond its natural strengths.

Stage	Primary Platform	Role	Representative Outputs
1	AWS Transform	Foundational analysis and transformation	Dependency maps, business logic documentation, domain grouping, wave planning, transformed assets
2	AAVA	Engineering enrichment and governance	Stories, design documents, decomposition plans, idiomatic code structure, utility modules, tests
3	AWS Kiro	Spec-driven finishing and developer flow	Scaffolded implementation tasks, validation support, build-oriented completion workflow
4	Delivery team	Human review and productionization	Approved artefacts, release decisions, operational hardening, deployment

This division of responsibility creates the ecosystem to deliver the best-in-class output for the end customers. AWS Transform remains the anchor of the modernization motion. AAVA increases the practical delivery value of that motion. Kiro supports developer productivity and validation. Human stakeholders remain accountable for sign-off at every critical boundary.

Modernization Delivery Methodology

A combined platform only delivers value when it is embedded in a delivery methodology that large enterprises can trust. The recommended model is a supervised agentic workflow with explicit human review checkpoints.

Review role	Review point	Responsibility
Mainframe expert	After AWS Transform outputs	Confirms that dependency, documentation, and data interpretations are directionally correct before downstream enrichment begins.
Product owner	After story generation	Validates that the inferred backlog represents the intended business operations and is usable for planning.
Lead architect	After design generation	Approves architecture decisions, service boundaries, API contracts, and target-state design principles.
Senior engineer	After code and test generation	Checks conformance with engineering standards and validates readiness for build, integration, and release.

This human-in-the-loop model supports a careful message externally as well: the platforms accelerate the work, but accountable modernization decisions still remain with the enterprise delivery team.

Deep Dive on AAVA Platform Agents

A1 - COBOL Relationship Mapper

Consumes source materials and AWS Transform outputs to create a structured relationship map for downstream reasoning.

A2 - Decomposition and Migration Planner

Uses relationship intelligence to propose bounded contexts, migration slices, and strangler-style sequencing.

A3 - User Story and Technical Document Creator

Transforms modernization intelligence into epics, features, user stories, and technical summaries that business and engineering stakeholders can review.

A4 - Design Document Creator

Produces architecture artefacts covering service boundaries, APIs, entities, deployment topology, and target-state design choices.

A5 - Modernization Code Generator

Generates Spring Boot-oriented implementation assets aligned to the approved design and engineering conventions.

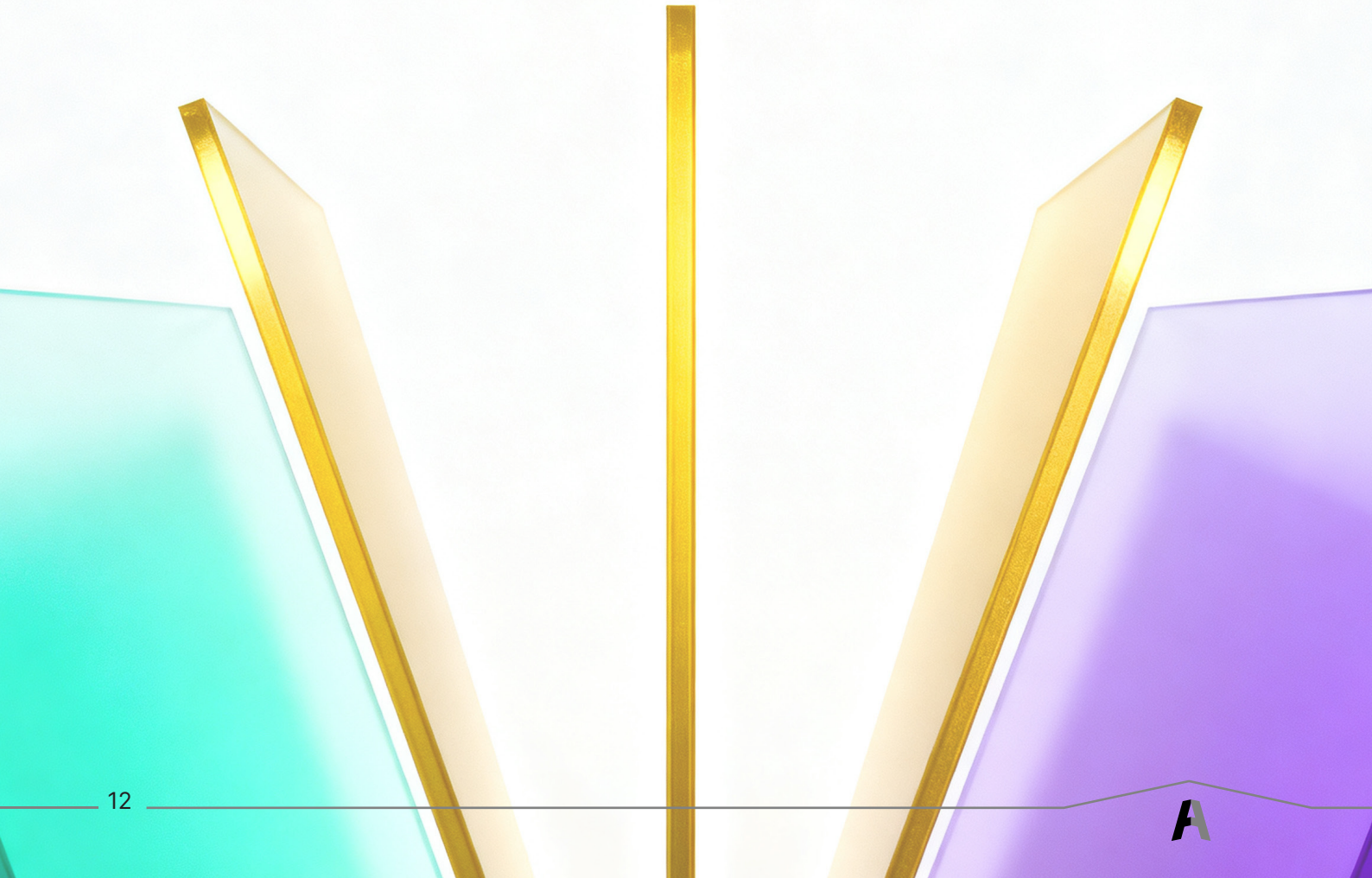
A6 - Unit Test Generator

Produces unit and integration-oriented test assets so the implementation can enter build validation with stronger quality signals.

Conclusion

AWS Transform is the modernization catalyst. It provides the foundational analysis and transformation capability that makes large-scale COBOL modernization materially more achievable. AAVA complements that foundation by extending the output into enterprise-ready artefacts, architecture interpretation, maintainability improvements, and governance-aware delivery assets. Where teams adopt AWS Kiro, the workflow can continue into a spec-driven finishing environment that helps developers move from structured intent to validated implementation.

Positioned this way, AAVA amplifies AWS Transform - taking the acceleration that AWS provides and helping enterprises convert it into a broader, more consumable modernization outcome.



Glossary of Terms

Term	Definition
AAVA	Ascendion AI-Assisted Value Accelerator, a multi-agent platform used to enrich and operationalize modernization workflows.
AWS Transform	Amazon Web Services modernization platform used in this paper as the foundational analysis and transformation layer.
AWS Kiro	An AWS developer experience environment positioned here as a possible spec-driven finishing layer in the delivery workflow.
Bounded context	A domain-driven design concept used to define coherent business boundaries for service decomposition.
COBOL	Common Business-Oriented Language, widely used in enterprise mainframe systems.
HITL	Human-in-the-loop, a workflow pattern in which people review and approve AI-generated outputs at critical checkpoints.
JCL	Job Control Language used in IBM mainframe environments to manage batch execution.
JPA	Java Persistence API, commonly used in Spring-based applications for persistence mapping.
SDLC	Software Development Lifecycle.
Spring Boot	A Java framework for building standalone, production-grade applications.
Strangler Fig pattern	An incremental modernization approach in which new services progressively replace portions of a legacy system.
VSAM	Virtual Storage Access Method, an IBM mainframe data storage approach commonly considered during modernization.

Authors:

Goutam Mukherjee
Sr. Director – Platform Engineering
goutam.mukherjee@ascendion.com

Saniya Kapur
Associate Engineer
saniya.kapur@ascendion.com