

Report | Discovery Research

▼ Note type

Research

 Research Attributes☒ Customer Account

 Participant(s)

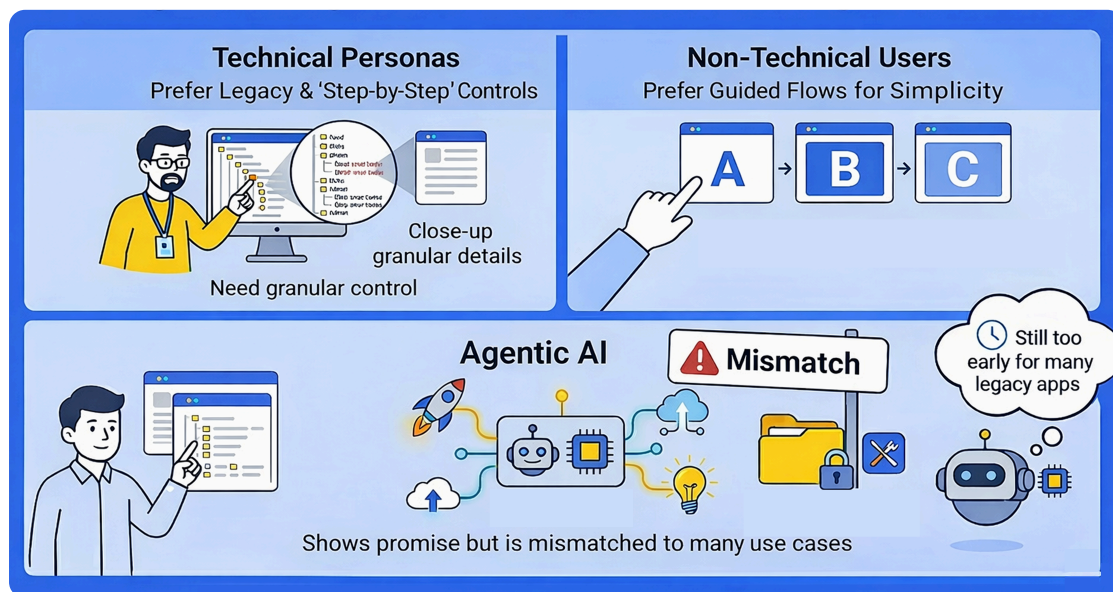
Feedback Source

 Date Conducted Empty

▼ Research Stage

Research Method

Product Domain

☒ Beta / Early Access

Executive Summary

This discovery research aimed to uncover user needs, behaviors, mental models, and pain points in application onboarding, configuration, and management across Saviynt's products (e.g., IGA, PAM). Through internal stakeholder interviews and customer sessions, we identified significant usability challenges, including fragmented workflows, over-reliance on

legacy interfaces for control and flexibility, and under-adoption of newer experiences like AOB and Agentic AI due to incomplete coverage of complex scenarios.

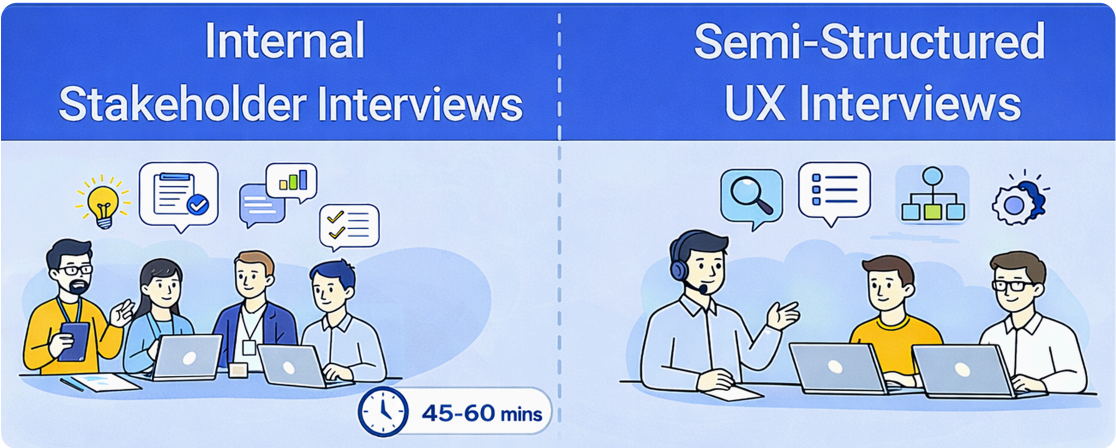
Findings are organized around the three onboarding pathways that exist today, the legacy configuration experience, the Application Onboarding (AOB) wizard, and the emerging agentic AI capability, followed by cross-cutting themes that span all three.

Background and Research Goals

[Link to Discovery Research Plan](#)

Methodology

Mixed methods blending qualitative data for empathy and insight:



Research Activity	Participants	Format	Purpose / Focus Areas
Internal Stakeholder Interviews	6 Solutions Engineers & Technical Architects	Alignment sessions	Validate assumptions, understand technical and organizational constraints, and evaluate existing experiences
Semi-Structured UX Interviews with Customers	3 customers	Workflow-led, semi-structured sessions	Observe natural navigation; assess management workflows, hierarchy, attribute prioritization, user confidence, and gather feedback on new concepts

Key Takeaways

A Blended, Unified Onboarding and Configuration Experience

The current experience is fractured across three onboarding pathways and two configuration modes, each with different strengths and gaps. Users want a single, modern, usable interface that guides them through onboarding and configuration in a coherent flow, while preserving the flexibility of direct JSON editing for complex applications that require it.

The goal is not to eliminate JSON, but to make it one tool within a guided experience rather than the entire experience. A unified approach would blend the AOB wizard's approachability, the assisted flow's scaffolding, and advanced mode's depth into a continuous journey without cliffs or dead ends.

Greater Self-Service for Business Users

Every session surfaced the aspiration to enable application owners non-technical business users to participate meaningfully in onboarding and management without relying on highly technical workflows. Today, every pathway requires JSON knowledge, developer-level terminology, or administrator hand-holding. The opportunity is a tiered model: simplified intake and task routing for business users, with guardrails and approval gates, feeding into expert configuration for administrators. This would reduce the bottleneck on central IT teams while maintaining governance and quality.

Inline Testing and Validation

Across all pathways and configuration modes, users have no way to test as they build. The current model forces users to leave the configuration experience or run the entire process end-to-end before discovering whether something works. The need is for inline, incremental validation at every stage: test a connection without completing the full setup, validate a JSON block before saving, preview import results before committing, simulate provisioning before going live. This single capability, building confidence as you go rather than hoping it all works at the end would fundamentally change the configuration experience.

Auditing and Version Control

For organizations operating in regulated environments, the ability to track changes, compare configurations, roll back to known-good states, and maintain approval workflows is not optional, it is a compliance requirement. The legacy history tab provides a partial foundation, but it lacks diff views, rollback, downloadable snapshots, and approval gates. Newer pathways have no auditing at all. Building comprehensive version control into the unified experience would both serve compliance needs and give all users the confidence to make changes without fear of irreversible mistakes.

Workflow Visibility and Progress Across All Flows

No flow in the current experience, the AOB wizard, the assisted configuration, or advanced mode tells users where they are, what they've completed, or what remains. There are no step counters, no progress bars, no completion indicators, and no reliable save-state feedback. Users navigate blind, unsure whether their work is saved, uncertain how much is left, and unable to jump to a specific step without clicking through everything in between. This is a universal need that cuts across every part of the product: a consistent model for showing progress, indicating completion, and letting users orient themselves at any point in the process.

Consistent Terminology and Meaningful Labeling

The product uses different terms for the same concepts across different screens and pathways. "Policies" and "rules" are interchangeable. "Custom Property 30" conveys nothing about what the field actually holds. "Disconnected" implies file-based onboarding in one context and URL-based in another. These inconsistencies add cognitive load for every user and are especially disorienting for non-experts encountering the product for the first time. A terminology audit and consistent labeling strategy, meaningful defaults, identical names for identical concepts, and clear distinctions where things genuinely differ would reduce confusion across the entire experience.

AOB (Application Onboarding) Experience Feedback

AOB wizard provides a guided, modern initial flow but falters in post-onboarding, leading to hybrid usage and abandonment for management.

Usage Pattern

The wizard is used extensively in sales demos.

The typical flow is: open the marketplace catalog → search for an application → select it → enable features like account/group management and request configuration → and finally, fill in the basics via guided steps. The experience demos well and generates genuine customer enthusiasm.

However, a critical demo behavior has emerged: demo teams deliberately avoid saving or finalizing in the wizard. Completing the wizard and attempting real configuration would expose the sharp transition into fragmented legacy screens. Instead, demos switch to pre-onboarded applications for deeper discussions. This approach is effective for sales but creates an expectations gap that surfaces during implementation.

Customers do not use the wizard as a complete onboarding solution. It serves as an initial shell generator, basics are configured in the wizard, then all substantive work happens in legacy. Senior technical users skip it entirely.

Strengths

- **Strong first impression:** The guided, step-by-step approach feels modern and accessible. It creates genuine excitement and contrasts favorably with the legacy experience.
- **Marketplace model:** The marketplace is seen like an app-store (search for an application, select it) resonates. Pre-populated fields for known applications like SAP cover roughly 90% of initial setup needs.
- **Vault configuration:** Configuring credential vaults within the wizard is convenient and preferable to plain password fields.
- **Entitlements grouping:** The wizard's approach to organizing entitlements is cleaner than the legacy equivalent.

Pain Points:

- **The AOB-to-configuration Cliff Is the Most Critical UX Failure**

Critical - Identified Across All Sessions

The AOB wizard delivers an excellent guided experience for initial setup, but once it completes, users are dropped into fragmented configuration screens with no guidance, no progress indicators, and no continuity. This is the single most impactful finding in the entire study.

After the AOB wizard completes, there is no landing page, no indication of what remains to be configured and no bridge into the configuration experience. App owners who complete the AOB flow believe they are done, unaware that substantial follow up configuration is required. This breaks the self service promise and erodes trust.

- **REST and Custom Applications Get a Blank Slate**

The wizard is not useful for REST and custom application types. REST onboarding presents a single blank text box with no assistance, no scaffolding, no examples, no progressive fields. This compares unfavorably to legacy's assisted setup, which at least provides initial JSON structures based on selections.

Competitors auto-pull schema from databases once connected and present options visually. Saviynt's generic flows offer no such intelligence. For disconnected apps, the wizard assumes file-based input but the actual flow leans toward URL/RPA-style interaction, creating a mismatch with the vast majority of enterprise use cases where file-based integration (SFTP, S3, CSV) is the standard.

- **No Progress Tracking, Save-State, or Completion Indicators**

No progress indicators exist within the AOB wizard or the configuration experience. Users cannot see which sections are complete, what remains, or whether their work has been saved. There is no auto-save feedback. The rule builder lacks mid-process saving, navigating away may result in lost work.

Grouped interfaces compound this problem by hiding failures. When mandatory fields deep in a form block progress, users cannot anticipate which fields are required or why. Step-by-step flows with visible pass/fail indicators at each stage are strongly preferred for their transparency and predictability.

- **The Manage Apps Page Lacks Business-Centric Filtering**

The Manage Apps page currently supports filtering by application type, but this is insufficient for large-scale environments. Organizations with hundreds of onboarded applications need to filter and organize by business-relevant dimensions: application owner, geography or custom metadata tags. The current type-only filtering makes it difficult to find,

manage, or report on applications in the way that business stakeholders actually think about them.

- **Validation Errors and Preview Gaps Undermine Trust**

The onboarding wizard sometimes rejects valid inputs, for example, SAP instance number validation that flags legitimate values, forcing users to enter dummy data to proceed.

- **Pre-Amsterdam Applications Are Invisible**

Applications onboarded before the Amsterdam release do not appear in the Manage Apps/Integrations list, only wizard-onboarded applications are shown. Organizations with existing deployments see an incomplete picture of their application landscape, which undermines trust in the new interface as the single source of truth and discourages migration.

Recommendations

- **Extend the wizard into post-onboarding configuration:** Add guided steps for attribute mapping, connection parameters, custom attributes, and basic provisioning rules. Reduce the cliff by covering the most common post-setup tasks within the guided flow.
- **Build a post-wizard landing experience:** After the wizard completes, show a dashboard with completion status for each configuration area, clear next steps, and contextual links. Make it obvious what's done and what remains.
- **Add assisted intelligence for REST and custom apps:** Auto-recommend JSON structures based on URL or schema. Pull schema post-connection for known database types. Provide guided steps for file-based disconnected apps.
- **Implement progress tracking and persistent save:** Add visible progress indicators across all steps. Enable save-at-any-point with clear feedback. Support pausing and resuming with contextual re-entry prompts.
- **Surface all applications in Manage Apps:** Ensure pre-Amsterdam applications appear alongside AOB wizard-onboarded ones for a complete, trustworthy inventory.

- **Add contextual guidance:** Replace technical tooltip text with plain-language explanations answering “What does this do?” and “Why would I enable this?”
- **Introduce a Form Builder for Customizable Onboarding Screens:** A significant wish across sessions is the ability to customize onboarding screens through a form builder. This would allow administrators to rename field labels and hide irrelevant options for specific application types or organizational contexts, add custom metadata, reusable templates according to organizational needs
- **Redesign the manage apps page:** Shift filtering from type-based to business-centric dimensions like Filter and group by application owner, business unit, geography, or compliance classification

Legacy Experience Feedback

Legacy workflows (connectors/endpoints/security systems with JSON configs) are the default for post-onboarding and complex setups, preferred by technical users for control but criticized for clutter and inefficiency.

Usage Pattern

In demos, legacy screens are shown selectively, typically with pre-configured applications like Active Directory to avoid exposing raw JSON complexity. JSON-heavy and advanced configuration screens are avoided in sales contexts because they appear overwhelming and undermine the simplicity narrative.

In implementation and day-to-day operations, legacy is the default. Implementation teams work through Excel templates and one-by-one configuration for reliability. Enterprise customers operate in a hybrid mode: they may start in the new AOB wizard to create a basic shell, then immediately switch to legacy for real configuration. Senior developers skip the wizard entirely, building and testing JSON externally in tools like Postman before pasting it into Saviynt.

Strengths

Despite its age, the legacy experience has capabilities that users actively value and that newer workflows do not provide:

- **Assisted scaffolding:** For known protocols, legacy provides progressive fields based on selections (e.g., POST body, authentication type). This guided scaffolding is especially valued for REST connectors, where it provides an initial JSON structure that can be manually refined.
- **Pre-import preview:** The ability to see sample data before committing an import prevents garbage data from repeated test imports
- **History tab:** Field-level change tracking by date and user has proven essential for audit compliance in regulated environments. No equivalent exists in the new experience.
- **Simulate provisioning:** Resolving provisioning values before committing allows validation without production risk. This capability exists for only a subset of connectors and is wanted universally.
- **Failure visibility:** Step-by-step legacy flows show exactly where failures occur. This transparency is the primary reason implementation teams prefer legacy over grouped or wizard-based interfaces, which can hide errors behind mandatory field blocks.
- **Pagination and column mapping:** Functional at roughly 80–90% coverage, sufficient for most needs, though REST pagination lacks offset and subtype options.

Pain Points:

- **JSON Configuration Dominates the Experience**

JSON editing is the central activity in legacy configuration and the single biggest pain point. The editors are small text areas with no syntax highlighting, no validation, no auto-formatting, and no error feedback. Missed commas or curly braces cause silent failures that are extremely difficult to diagnose.

Customer (Kiewit Corp): *"99% of my configuration time is spent in JSON. It's an unavoidable evil with the current tooling."*

A common workaround has emerged: copy JSON to an external editor or Postman, make changes, then paste it back. This workflow introduces copy-paste errors, creates security exposure from sensitive data in unsanctioned tools, and adds unnecessary friction. Multiple separate JSON blocks exist for different operations (create account, update account, enable, disable), which feel redundant and are confusing to navigate between.

- **Testing and Troubleshooting Are Blind**

Testing capabilities are inadequate and opaque. The “test connection” feature requires a fully completed setup to function, preventing incremental validation. There is a single monolithic “Save & Test” action that spawns background jobs with unreliable results. There is no visibility into what Saviynt sends or receives during API calls, when something fails, there is no way to diagnose whether the issue is in the configuration, the network, or the target application.

Queries that work perfectly in external tools fail silently in Saviynt with no telemetry to explain the discrepancy. There is strong demand for per-section testing: ping the server, test authentication only, validate a search filter, confirm object filter returns, each as an independent, incremental check.

- **Clutter Accumulates Without Cleanup Tools**

There is no ability to delete or archive test applications, resulting in a cluttered environment that makes it harder to find production configurations. Documentation and sample configurations are outdated. The history tab, while valuable, lacks diff views, users can see that a field changed on a given date but cannot easily compare before-and-after states or roll back to a previous configuration.

- **Requirements gathering happens entirely outside the product**

Implementation teams send Notepad/ Excel templates about user counts, host names, use cases, connected/disconnected details before onboarding begins. This creates ambiguity and back & forth. There is no product mechanism for intake or requirements gathering.

Recommendations

- **Modernize the JSON editor:** Replace raw text areas with a code editor offering syntax highlighting, real-time validation, error detection, auto-formatting, and bracket matching. Provide pre-populated templates for known connector types.
- Eliminate the need for external tools like Postman.
- **Simplify navigation:** Move to top navigation with a sequential flow (connection → import → provisioning). Consolidate redundant screens and connection details into a single view.
- **Build in-product intake templates:** Replace external Excel workflows with guided, in-UI requirement gathering (user counts, hostnames, use cases, connectivity details) that feeds directly into onboarding.
- **Enhance history and auditing:** Add diff views, rollback capability, and downloadable configuration snapshots to the history tab.
- **Enable cleanup:** Allow deletion or archiving of test applications.

Agentic AI Experience Feedback

Agentic AI generates excitement as a future-proof innovation but remains early-stage and misaligned with prevalent use cases.

Usage Pattern

Agentic AI appears in nearly every sales demo, it is positioned as the innovation story, proof that Saviynt is investing in reducing the friction that has historically made identity security onboarding painful. The flow walks through URL input, prompting, browser recording, and auto-generated JSON.

For existing customers, the AOB wizard is used to prove GA capability; agentic AI is reserved for the vision conversation.

No one uses agentic AI for production onboarding. It functions exclusively as a demo and sales tool at present.

Strengths:

- **Universal excitement:** Every session reported positive reactions. The plain-English, ChatGPT-like interaction model resonates immediately

and differentiates the product from competitors.

SE from Saviynt: *"No customers reject agentic AI, there's universal interest. But it's too early for deep adoption."*

- **Connector building from API docs:** Building connectors on-the-fly from documentation eliminates the recurring customer question of whether a specific vendor connector exists. It shifts the paradigm from a fixed marketplace to an open-ended capability.
- **Competitive positioning:** Agentic AI is the primary innovation narrative in sales. For customers tired of manual JSON configuration, the contrast between old and new is immediately compelling.
- **Prompting flow design:** The interaction model for gathering information is well-designed as a foundation to build on.

Pain Points:

- Not truly conversational: mostly form-filling with limited chat; no always-on guidance.
- The agentic flow focuses on URL-based, RPA-style onboarding (web walkthroughs, browser recording), but the vast majority of enterprise disconnected apps use file-based integration: CSV over SFTP or S3. Major enterprises across financial services, healthcare, manufacturing, and transportation all reported file-based as their standard. URL-based RPA success stories are rare exceptions, not the norm.
The naming compounds the problem: the "Disconnected" label in the UI implies file-based onboarding, but the actual flow prompts for URLs. This mismatch creates false expectations and reduces the feature's perceived relevance for the most common use case.
- No built-in testing or validation: connection tests missing post-setup; errors hidden until provisioning jobs fail; troubleshooting logs unhelpful.
- Progress/resume gaps: no smart "welcome back" prompts or pending-task dashboard; no labeling to distinguish agentic-built apps.

Recommendations

- **Prioritize file-based disconnected app support:** Extend the flow to handle SFTP/S3 file locations, auto-build schema from uploaded files, and support periodic imports. This addresses the majority of real enterprise needs.
- **Make it truly conversational:** Enable always-available chat for questions, guidance, and navigation throughout the flow. Support natural language queries like “What does this field mean?” and “Take me to account configuration.”
- **Add post-setup testing and validation:** Implement connection tests, input validation, mapping verification, and clear error reporting with diagnostic suggestions.

Configuration Experience Feedback

Regardless of whether an application was onboarded through Legacy, AOB wizard or agentic AI, all users land in the same configuration experience for post onboarding-setup.

Usage Pattern

Users frequently switch between modes, and in many cases assisted mode only covers initial setup before users must move to advanced. The two modes share the same navigation structure and visual framework.

In demos, assisted mode is shown selectively. Solutions Engineers use it for straightforward connectors like Active Directory, where pre-configured attributes make it the easiest path to demonstrate depth. For REST/database/SOAP, it is used as the entry point before pivoting. Assisted mode is rarely demoed for advanced scenarios, it looks too limited for technical audiences who will immediately ask “what if” questions.

In real-world use, assisted mode serves as a starting scaffold. Users complete the guided steps to generate initial JSON, then switch to advanced mode for refinement. Implementation teams use it to build a baseline, particularly for REST connectors where the progressive fields help structure the first API call. However, once the initial setup is done, users do not return to assisted mode, all subsequent work happens in

advanced. For disconnected or complex multi-connection apps, assisted mode is skipped entirely.

Strengths:

Assisted configuration:

- **Progressive scaffolding:** Form fields that build on each other based on prior selections provide a starting JSON structure. For REST connectors, selecting authentication type, HTTP method, and endpoint URL generates a working foundation that can be refined. This is genuinely valued as a starting point.
- **Walkthrough plus editor switch:** For REST and SOAP connectors, assisted mode offers a guided walkthrough along with the ability to switch to full editor, which is appreciated for entry level guidance while preserving expert access.

Advanced configuration:

- **Full control and flexibility:** Advanced mode provides complete access to every configuration parameter. For technical users, this is essential, there is no black box. Every JSON field, every mapping, every rule is visible and editable.
- **Rule designer:** The drag-and-drop rule designer supports logic like random/auto-increment and is better than legacy text-only methods. When shown in demos, it is generally well-perceived.

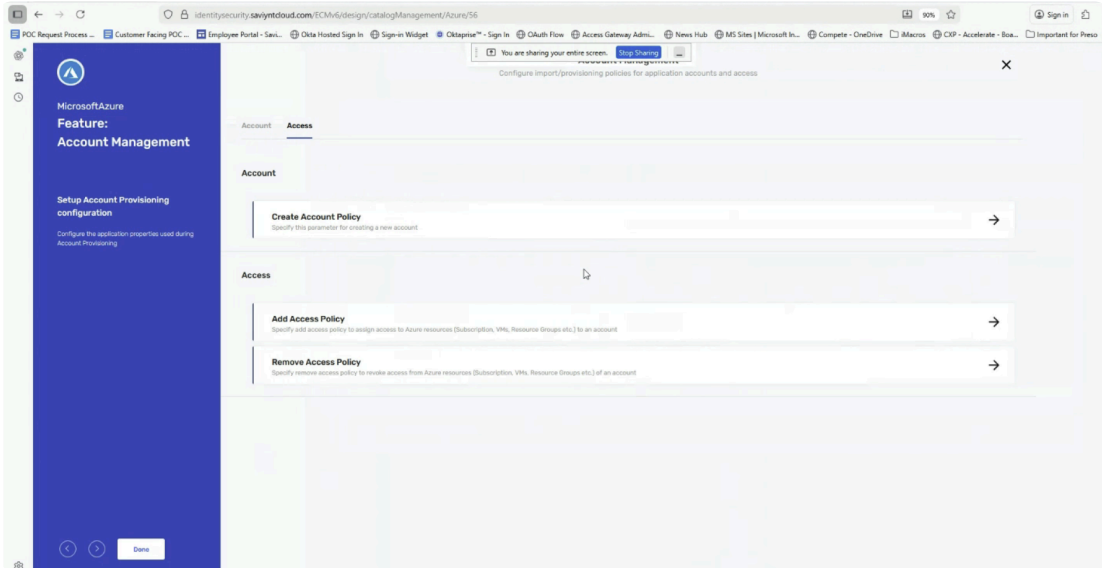
Pain Points:

Assisted configuration:

- **Assisted Coverage Is Limited and Inconsistent Across Connector Types**

Assisted mode exists only for certain connector types, and its depth varies significantly. Active Directory is well-covered. REST, database, and SOAP connectors have basic assisted flows but lack consistent handling of multi-valued fields, and the scaffolding often omits common configuration elements like test connection parameters and pagination options. For disconnected and generic protocol apps, assisted mode may offer nothing at all, users encounter a blank slate with no guidance.

- **Overwhelming Visual layout**



(Snapshot of heavy left side navigation - mention of “too much blue”)

The configuration interface has heavy left-side navigation with a dense blue visual treatment that dominates the screen. There are too many visual elements tabs, sub-tabs, buttons, collapsible sections without a clear hierarchy or flow. The volume of navigation options creates cognitive overload rather than guiding users through a logical sequence. This visual density is one of the most commonly cited pain points and has been described as “blue overload.”

- **The Assisted-to-Advanced Cliff Mirrors the AOB-to-configuration Cliff**

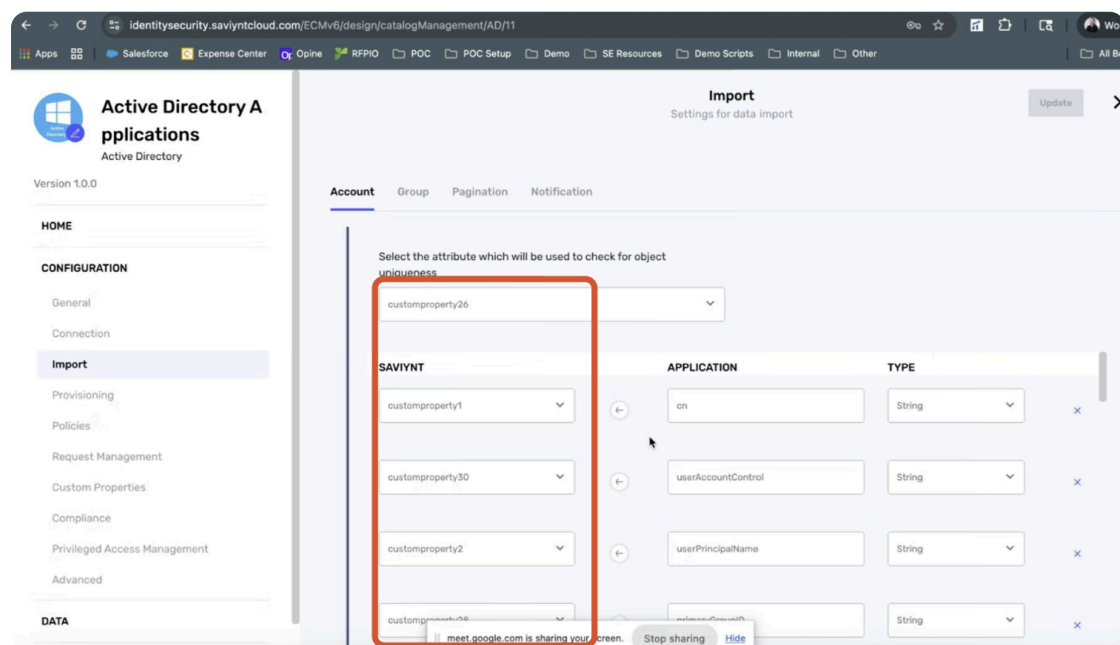
The transition from assisted to advanced mode is abrupt. Users complete the guided steps, then find themselves in the full advanced view with no indication of what was set up, what remains, or where to go next. There are no post-assisted prompts like “Next: configure attributes here.” The assisted experience creates the same cliff pattern as the AOB wizard: an initial guided flow that ends without a bridge into the remaining work. Users, especially app owners who were promised a self-service experience don’t know to return to attribute mapping or advanced config screens after assisted mode ends.

- **No Workflow Visibility or Step Navigation**

The assisted configuration flow provides no indication of where the user is in the process. There is no step counter or progress bar showing how many steps exist, which steps have been completed, and how many remain. Users move forward and backward using arrows, but there is no way to jump directly to a specific step, reviewing a prior step requires clicking through every intermediate step sequentially. This makes it difficult to orient within longer flows, plan time for completion, or quickly revisit a specific field without losing context on the current step.

Advanced Configurations:

- **Attribute Mapping lacks context and reference**



(Snapshot of attributes like “Custom Property 1” and “Custom Property 30” shown under the Import tab in Advanced Configuration, which makes it difficult to determine what these fields represent.)

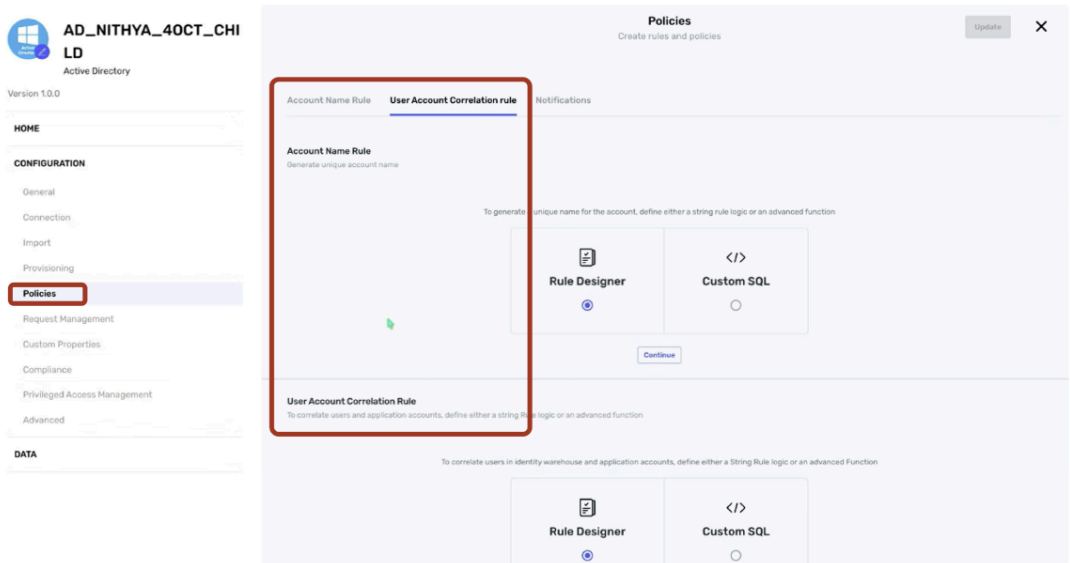
Labels like “Custom Property 30” provide no meaning. The ability to relabel custom properties exists but is hidden in a separate admin section that most users never discover.

- **Rule building is improved but incomplete**

The drag and drop rule designer is better than legacy methods, but it lacks mid-process saving, navigating away risks losing work.

- **Inconsistent terminology (“policies” vs “rules”)**

Terminology is inconsistent across the interface, with interchangeable use of “policies” and “rules” which creates confusion.



- **Update / Save behavior is unpredictable**

There is no clear indication of when data is auto-saved versus when manual saving is required. Updating configuration may or may not affect live connectors, the impact of changes is unclear. There is no partial save capability, no confirmation prompts to prevent accidental overwrites, and no ability to pause and resume a configuration session. Certificate changes often require reboots, adding operational friction to what should be a routine update.

- **Preview, Validation, and Feedback Gaps**

There are no JSON preview or edit capabilities within the wizard flow and no pre-import sample previews. Access query configuration appears in multiple locations with unclear effects. Dynamic attributes and action strings use opaque syntax with unclear hide/show logic. Entitlements are inaccessible until after the first import, creating a circular problem for new applications.

Cross-cutting themes and opportunities

The following themes emerged consistently across all three pathways. They represent systemic issues that require coordinated investment beyond any single experience.

Self-Service for Business Users Remains Unrealized

Enabling application owners, non-technical business users to onboard and manage their own applications is a stated goal and a recurring aspiration. Today, it is not achievable through any pathway. Legacy requires JSON expertise. The AOB wizard drops users into technical configuration screens, and the assisted flow within those screens drops them further into advanced mode. Agentic AI assumes developer-level knowledge. Admin teams handle everything because the UI overwhelms non-technical users.

The desired model is a delegated intake workflow: app owners submit basic information through simplified forms, which are reviewed and approved before administrators complete the technical configuration. This would shift repeatable, low-complexity onboarding off central teams while maintaining governance through gates and approval steps.

Testing and Validation Are Inadequate Everywhere

No pathway provides adequate testing. Legacy's test connection requires full setup. The wizard has no testing mechanism. Agentic AI has no post-setup validation. Across all pathways, there is no visibility into API requests and responses, no per-section testing, and no clear error diagnostics. This drives extended implementation cycles, forces reliance on external tools, and means errors are often discovered only in production.

Auditing, Version Control, and Compliance Are Underserved

The legacy history tab provides basic change tracking that has proven essential for audit compliance, but it lacks diff views, rollback, and downloadable snapshots. Newer pathways have no auditing capabilities at all. There are no approval workflows for configuration changes, no telemetry on operational details (e.g., password rotation timestamps), and no way to compare current state against a known-good baseline.

Design direction: Implement version control across all pathways: diff views, rollback to previous versions, downloadable snapshots, approval workflows, and operational telemetry. Ensure these capabilities exist in the new experience so that migrating from legacy does not mean losing compliance-critical features.

Discovery and Application Context Are Fragmented

Application context is scattered across multiple external systems: CMDBs track assets, GRC tools hold compliance classifications, cloud providers manage infrastructure. Onboarding teams lack a unified view of what an application is, where it runs, who owns it, and what compliance requirements apply. A single application may span many endpoints across production, QA, and test environments without correlation, discovery provides an incomplete picture.

Requirements gathering relies on external Excel templates asking about user counts, hostnames, use cases, and connectivity. There is no in-product mechanism for this intake, and the external process introduces ambiguity and back-and-forth.

Terminology and Labeling Are Inconsistent

"Policies" and "rules" are used interchangeably.

"Custom Property" labels are meaningless.

"Disconnected" implies file-based but leads to URL-based flows.

The wizard, post-wizard configuration, and agentic AI each use slightly different terms for the same concepts. This inconsistency adds cognitive load for all users and is especially disorienting for non-experts encountering the product for the first time.