Migrating from *Matillion ETL to Maia*

A FIELD-ENGINEERING WHITE PAPER

Maia

Contents

Matillion's current **AI Data Automation** platform, **Maia**, comprises three integrated components:

01 **Maia Team**
(the AI agents, including the Migration Agent),

02 **Maia Context Engine**
(business rules and institutional knowledge),

03 **Maia Foundation**
(the enterprise backbone; security, governance, observability, runtime, and scale).

The broader Matillion product suite is the **Maia platform**
Where this paper talks about "the platform you're migrating to", we mean Maia and it's underlying Foundation.

Please note that a large portion of the migration is deterministic and can be done using non-AI tooling, which may be preferable due to security concerns. However, migration is accelerated when the Maia Migration Agent is used.

Executive Summary

The Shift

Matillion is transitioning from Matillion ETL (a legacy, VM-based, customer-operated platform) to Maia, an AI-native, SaaS-based data orchestration platform.

Benefits

Maia delivers a modern, API-first development experience with native Git integration, horizontal autoscaling, and reduced operational overhead. Its AI-native architecture, powered by Maia Team agents, accelerates development and remediation.

Key Considerations

The migration process is largely deterministic, allowing most pipelines to move without manual refactoring. Success hinges on a two-step approach: performing a “lift-and-shift” migration first to ensure parity, followed by modernisation initiatives like event-driven jobs and shared logic refactoring. Organisational readiness, specifically embracing Git-based workflows and new runner operational models, is the primary driver for a successful transition.

01

Introduction

1.1 Elevating your data strategy: *The future with Maia*

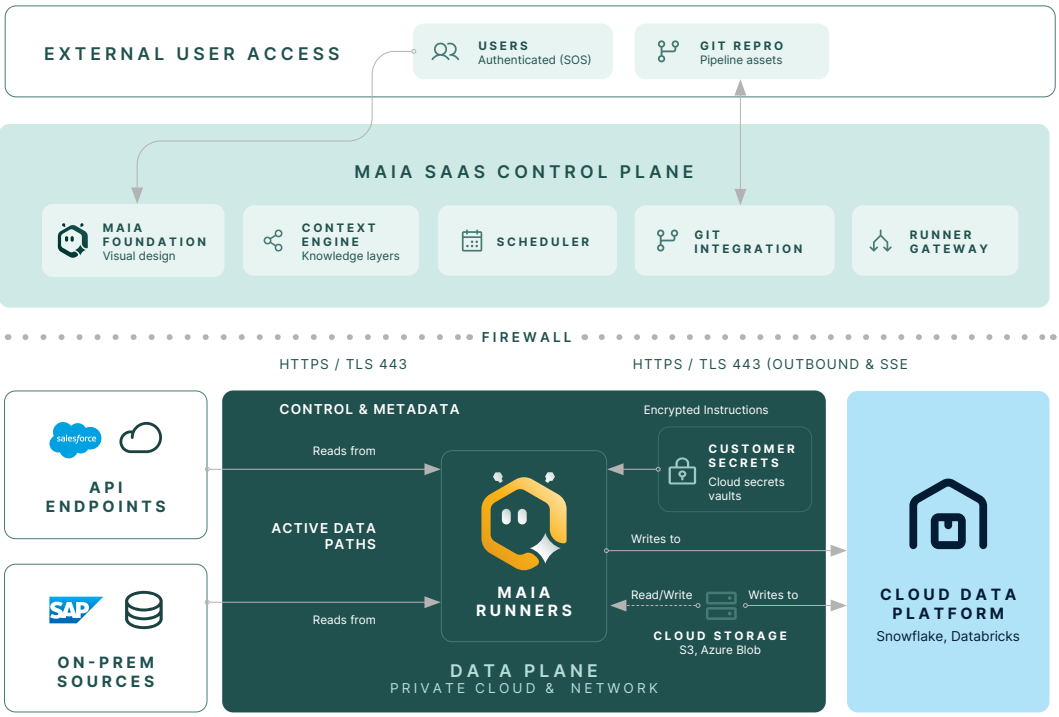
Matillion ETL was designed for an era when dedicated EC2/VM instances per environment were the norm, when data teams owned the operating system and pipelines, and when every customer ran their own copy of the orchestration plane.

That model worked; many of our largest customers still run multi-thousand-job workloads through it every day, but the cost shape, the operational burden, and the velocity ceiling all belong to that earlier era.

1.2 Architectural *comparison*

Matillion ETL single-instance, customer-operated VM: Every concern - UI, scheduling, orchestration, execution, metadata storage - runs in one place, owned and operated by the customer. Patching the VM means a maintenance window. Scaling means a bigger VM. Multi-environment means multiple VMs. There is a single canonical copy of the truth, and it resides on a host the customer is responsible for.

Maia split control plane + Maia runner (Hybrid SaaS shown): The control plane (Designer, Scheduler, Runner Gateway, API Gateway, Git integration, observability) is Matillion's responsibility; patched, scaled, and monitored as SaaS. In Hybrid SaaS, the runner is the only thing inside your network. Sources and the warehouse are accessible from the same network position as the Matillion ETL VM. Multiple runners can run in parallel; capacity grows as more runners are added.



1.3 Why Maia architecture *is superior*

Maia's architecture is built on three core pillars that provide a fundamental advantage over legacy VM-based systems:

01 Scalability

By decoupling the runners from the control plane, Maia allows capacity to scale horizontally. This amortises operational toil, shrinks failure domains, and ensures that recovery is as fast as a redeploy since runners hold no state.

02 API control

Maia features a first-class, comprehensive REST API for every orchestration concept. This API-first approach makes orchestration programmable, allowing for advanced automation via the Model Context Protocol (MCP) and full integration with enterprise CI/CD pipelines.

03 Version control

With a Git-first model, the repository is the source of truth rather than the orchestration engine. This enables modern software engineering practices like branching, pull requests, and straightforward rollbacks via Git reverts.

Maia separates the orchestration control plane (which Matillion now operates as a SaaS offering, built on the security, governance, observability, and scale guarantees of the Maia Foundation) from the data plane. For most migrating customers, the right choice is Hybrid SaaS: the data plane runs inside the customer's network, where the Matillion ETL VM runs today, so existing source connectivity carries over without changes. Full SaaS, with the data plane running on Matillion-hosted infrastructure, is available to customers whose sources are reachable via public or peered endpoints and who want to further reduce their infrastructure footprint.

The result: less infrastructure for the enterprise data team to manage, a CI/CD-first development workflow that fits modern data engineering practice, an API-first surface that makes orchestration programmable from the outside, an agentic data engineering workforce via Maia Team (including the Migration Agent that does most of the translation work for this very migration), and a scaling model that that frees up the data engineer from infrastructure tuning to deliver business value.

Customers migrate for one or more of the following reasons:

Category shift: ETL tool vs. agentic framework

While Matillion ETL remains a fully supported product, Maia represents a fundamental shift in category, moving from a traditional ETL tool to an agentic data engineering framework. This allows organisations to evolve from manual pipeline building to a paradigm where data engineers act as managers of specialised agents that understand their data architecture, effectively changing the working paradigm of the data organisation.

Cost:

A single Matillion ETL instance per environment, sized for peak concurrency, is an expensive idle. Maia's hybrid runner model lets execution capacity scale up and down with load, while the control plane is a fixed SaaS fee.

Developer velocity:

Native Git, environment promotion, an API surface for CI/CD, and modern observability primitives meet data engineers where they already work.

Operational risk reduction:

No more Tomcat, no more JVM tuning, no more upgrade outages on a single-instance VM. Patching, scaling, and resilience move to a managed runtime.

Strategic platform alignment:

Maia is the platform for new connectors, new transformation primitives, and new AI features for the future Data Engineering team. Staying on Matillion ETL means staying behind that line.

1.4 The next generation of *data orchestration*

Capability	Matillion ETL	Maia
Topology	One or more dedicated VMs per environment, customer-operated	SaaS control plane + Maia runners (customer-hosted in Hybrid SaaS, Matillion-hosted in Full SaaS)
Scaling	Vertical only; manual; capped by the largest VM you'll pay for	Horizontal runners can autoscale within the chosen hosting platform
Source control	Bolted-on Git via the UI, not the native dev model	Git is the source of truth; branches and PRs are first-class
API surface	Limited; UI-first	Comprehensive REST API for every orchestration concept
Upgrades	Customer-owned outage; risk per upgrade	Continuous delivery from Matillion; no customer outage
Connector velocity	Maintenance only	All new connectors land here first
Observability	Tomcat logs + custom scripts	Structured telemetry; natural language querying for business insights

1.5 Strategic Value and *Executive Considerations*

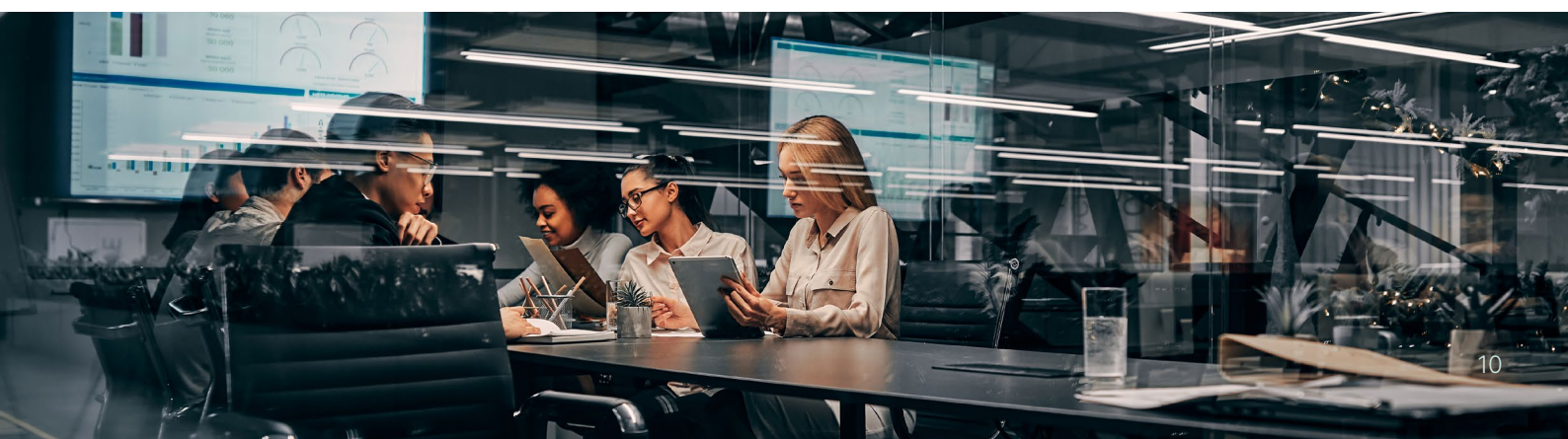
A migration of this kind is not a like-for-like reinstall. It is a one-time opportunity to minimize infrastructure and operational management challenges that have been quietly compounding for years and to land on a platform purpose-built for AI-native, collaborative data engineering at enterprise scale. Executives sponsoring this work should weigh these four strategic pillars.

First, Risk Management and Migration Effort. The migration itself is largely deterministic and low-risk. The bulk of a typical Matillion ETL estate is translated to Maia by Matillion's Migration Agent without per-object engineering effort. Furthermore, because Matillion ETL remains a fully supported platform, organizations can plan for a phased, safe cutover without the pressure of an immediate platform retirement. Success is found by resourcing the project around the small subset of objects that aren't automatic, rather than the parts that are.

Second, The Strategic Risk of Inaction. While Matillion ETL is stable, staying on a legacy, VM-based architecture creates an unrecoverable disadvantage. Competitors are already shifting to agentic data delivery paradigms that decouple growth from headcount. Organizations that delay this transition risk being trapped behind a velocity ceiling, maintaining infrastructure while peers are utilizing the Maia Team agents to accelerate development and remediation. Moving now is a defensive necessity to ensure the data organization remains a competitive asset.

Third, The Operating Model Evolution. The operating model changes more than the pipelines do. While the deterministic translation ensures familiar looking pipelines, the team's day-to-day will be fundamentally different: pipelines live in Git, environments are promoted via CI/CD pipelines, and runner capacity is observed rather than provisioned. Executive sponsorship is critical here to clear the path for these working norms to evolve. The teams that land fastest are those where leadership invested early in practices like PR review, environment promotion, and runbook ownership.

Fourth, Migration vs. Modernization. "Change as little as possible" is the right migration discipline. During the move, resist the temptation to refactor or restructure; every moving part lengthens the cutover. After migration, however, the architecture-better-suited-to-Maia conversations, event-driven jobs, granular shared logic, and better concurrency, are exactly the right ones to have. Treat these as sequential programs; migration's finish line is modernization's starting line.



02

The *migration process*

The process below is the migration workflow that Matillion Field Engineering runs with customers. Each phase has clear entry and exit criteria; the customer-facing experience is the same whether the estate has 50 or 5,000 pipelines.

Overview

- **Discovery**
- **Maia Runner + Source Connectivity**
- **Project, Env, Credential Config**
- **Workload Identification + Initial Migration**
- **Remediation**
- **Client Review + Cutover**
- **Parity Validation + Matillion ETL Decommissioning**

Each phase has a defined entry criterion and a defined exit criterion. Phases are sequential at the workload level but can overlap across waves in larger engagements (see §3 on planning).

2.1 *Discovery*

Goal:

Understand the customer's Matillion ETL estate with sufficient fidelity to plan the migration: which jobs exist, which connectors and components they use, what is and isn't deterministically translatable, what the long tail looks like, and which business processes the pipelines feed.

What happens:

The Matillion-assigned Solutions Architect runs an automated discovery diagnostic against an export of the customer's Matillion ETL metadata. The diagnostic enumerates jobs, classifies components by translatability, surfaces custom Python and unusual connectors as remediation candidates, and produces both a sizing estimate and a categorised blocker list.

Discovery sign-off, which historically required a multi-week stakeholder review cycle, now typically completes in a single working session; the heavy lifting is automated, and the conversation with the customer is about validating findings rather than producing them.

Exit criterion:

A signed-off Discovery output: estate summary, complexity score, blocker list with owners, and a wave-scoped migration plan if the estate is large enough to phase.

2.2 Infrastructure - *Maia runner, networking & security*

This can require multiple teams to collaborate within the client organisation, which can increase lead time if not managed and organised correctly.

Goal:

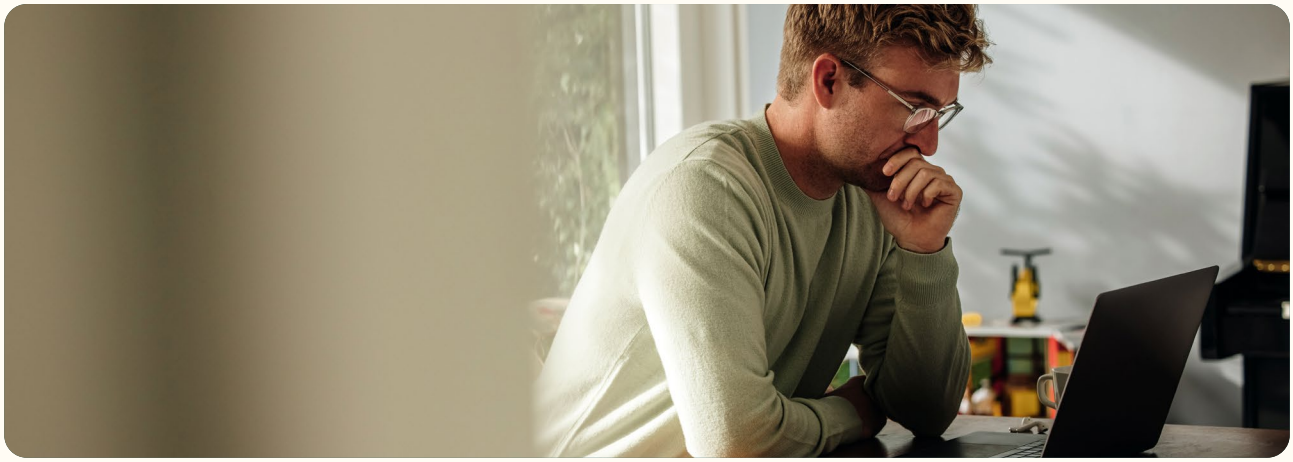
Stand up the execution plane in the customer's network and prove it can reach the data sources the pipelines depend on.

What happens:

This is the phase that Matillion Field Engineering's deployment team owns end-to-end. A Maia runner is deployed in Hybrid SaaS to one of the officially supported customer-hosted deployment targets (AWS Fargate, AWS EKS, Azure Container App/ACI, Azure AKS, Google Kubernetes Engine, or Snowflake Native App) and in Full SaaS to Matillion-hosted infrastructure. Network paths to sources (warehouse, object storage, on-prem systems, SaaS sources behind allowlists) are validated against the same routes Matillion ETL uses today. This is the central parity principle of the deployment phase: the new runner sits exactly where the old VM sat, network-wise. Connectivity behaviour is the one variable this migration cannot afford to introduce alongside everything else changing, and holding it constant is what keeps the rest of the parity story honest.

Exit criterion:

Runner registered to the customer's Maia Foundation tenant, source connectivity smoke-tested for every source class the estate depends on, and the customer's network/security review of the deployment is complete.



2.3 Project - *environment & warehouse configuration*

Goal:

Recreate the Matillion ETL project/environment topology in Maia and load credentials in a format that the migrated jobs can consume without being rewritten.

What happens:

Projects, environments, and warehouse references are configured in Maia Foundation.

Exit criterion:

All target projects/environments exist in Maia Foundation, warehouse credentials are loaded and verified against the live runner, and a verified warehouse connection exists on the runner.

2.4 Workload identification, credential configuration *& initial migration*

Goal:

Take the Discovery output and convert it into running pipelines in the Maia Foundation.

What happens:

Before conversion starts, the estate is broken into workloads, functional groupings of related pipelines (e.g., a business domain or shared-job cluster) rather than migrating pipeline by pipeline. Each workload carries its Type 1/2/3 mix (direct/reconfigure/refactor), estate size, and an effort estimate, and is assigned to a wave. Waves can be sequenced on different criteria depending on what the engagement needs to optimise for: cleanest-first to de-risk early, busiest-schedules-first to validate production load early, biggest-workload-first for common-denominator leverage, or by runtime/execution volume where performance is the primary concern. Most estates are mostly dormant; it's common for only a small fraction of pipelines to carry an active schedule or have been executed recently, so this phase typically also narrows the scope to the active subset, deferring genuinely dormant pipelines rather than migrating everything indiscriminately.

Matillion Field Engineering, supported by the Migration Agent for deterministic translation, converts the in-scope workloads, including credential creation. For partial migrations (waves), only the in-scope wave is converted; the runner's pipeline manifest is configured to expose only those workloads. The Migration agent handles the deterministic majority; the long tail is flagged for the Remediation phase.

Exit criterion:

All in-scope pipelines exist in the target Maia project, are syntactically valid, and execute for each workload. Sign-off is captured per business unit/schedule, so partial progress is auditable as waves complete rather than only at the end of the full migration.

2.5 *Remediation (per workload)*

Goal:

Resolve the long-tail issues that deterministic translation cannot solve.

What happens:

This is where the engagement varies most by customer. Due to the nature of product variations between Matillion ETL and Maia Foundation, and legacy design decisions, there may be a need to utilise the Migration Agent to adjust and adapt workloads; these are tracked as a remediation campaign with clear ownership and a visible status. Matillion Field Engineering triages, prioritises, and either fixes locally or escalates to the Maia engineering team when a platform-level change is required.

Exit criterion:

All differences identified during Discovery, plus any surfaced during Initial Migration, are either resolved or formally accepted by the customer as out of scope.

2.6 Client review *& testing (per workload)*

Goal:

Hand the migrated estate to the customer's data team, and agree on a testing window & validate pipeline accuracy.

What happens:

The customer's data team reviews the migrated pipelines and reviews data outputs. Customer's data team signs off on pipelines.

Exit criterion:

Customer has signed off and accepted migrated workloads.

2.7 Parity validation *& cutover (per workload)*

Goal:

Confirm, post-cutover, that the migrated estate produces the same business outcomes as Matillion ETL did.

What happens:

The customer's data team reviews the migrated pipelines, runs them in parallel against Matillion ETL for a chosen period (based on the frequency of existing production runs), and reviews the resulting data outputs. Once the parallel run confirms parity within agreed tolerances, schedules are activated in Maia. A short post-cutover window (1–4 weeks, depending on cadence) during which Field Engineering remains engaged to investigate any divergences identified by the customer's data team.

Where outputs are amenable to automated comparison, this parallel run is driven by a test harness rather than manual checking, see §4.5 for how Maia's automation collapses the assurance effort into a capability the customer retains.

Exit criterion:

Customer sign-off; engagement feedback captured.

2.8 Matillion ETL *decommissioning*

Goal:

Decommission the workload and retire the Matillion ETL infrastructure to enable cost savings and reduce maintenance effort for the customer.

What happens:

The customer signs off, the Matillion ETL VMs are decommissioned, the licence is terminated, and the engagement is closed with a feedback record that feeds back into Matillion's product roadmap.

Exit criterion:

Matillion ETL infrastructure removed.

03

How to plan your ***migration to Maia***

3.1 *The principle*

Change as little as possible in your migration. The single largest predictor of migration delay is scope creep, "while we're in there, let's also…" restructure the project topology, re-platform the warehouse, refactor that long-running pipeline, adopt a new naming convention, or break up the monolithic shared job.

These are all good ideas, just not now. They land better after migration, on the platform that actually supports them, rather than during it, when they introduce variables that dilute the parity story.

The discipline: the migrated estate should produce the same outputs from the same inputs as the Matillion ETL estate did, no more and no less. Optimisation comes after.

3.2 *Resources*

A typical mid-sized engagement (multiple workloads, mainstream connectors, no exotic Python) involves:

- **One Project Owner** to set the project rhythm, and control the sign-off and completion of milestones
- **One or two customer-side technical owners** who can speak to the pipelines, the warehouse, and the network.
- **A customer-side executive sponsor** who can unblock cross-team work (network changes, security reviews, schedule windows).
- **A customer-side data team,** of any size, that will own the migrated estate after cutover and should be engaged in Client Review.
- **Pre-Migration Maia Team Assets generated by the Matillion team** to help with any required refactoring or reworking of pipelines.

Larger or more complex engagements add additional remediation effort, wave-based scoping (see below), and sometimes Maia engineering escalation for platform-level remediation work.

3.3 *Timing*

For a mid-sized engagement (multiple workloads, mainstream connectors).

Roughly speaking:

- **Discovery** is the fastest phase under the current toolchain; typically, it involves a single working session once context files are produced.
- **Initial migration** is the largest single block of work, with most of the volume handled deterministically by the Migration Agent, while the long tail drives human effort.
- **Testing and parity validation** are predominantly customer-led and bounded by the cadence of the slowest pipeline (often a month-end close).
- **Deployment** is bounded by the customer's network and security review cadence, which is calendar-driven and not amenable to acceleration by adding people.

The longest pole in most engagements is rarely the Matillion upgrade effort; it is the customer-side calendar (network reviews, parallel-run windows, business-owner sign-off cycles). Plan accordingly.

For larger estates (>1,000 jobs, multiple business domains, exotic connectors), workload-based scoping is the right answer: identify the smallest coherent slice of the estate that can migrate independently (one business domain, one schedule cadence, one source class), migrate it end-to-end, and use the learnings to size the next wave.

3.4 *Preparation steps*

Before Discovery formally kicks off, the customer can shorten the engagement by:

01 Inventorying their Matillion ETL estate themselves

A clean export with the customer's own understanding of which jobs are in use vs deprecated is the highest-leverage preparation activity.

02 Identifying out-of-scope workloads

Every job you do not migrate does not have to be validated. If 30% of the estate is dead code, prune it in Matillion ETL first.

03 Standing up the Maia Foundation tenant

Tenant creation, SSO configuration, and initial admin user provisioning; all of this can happen in parallel with Discovery.

04 Briefing the network/security team

The Runner deployment requires an outbound HTTPS connection to the Matillion control plane and reuse of existing inbound paths to data sources. Surfacing this to the security team early reduces the calendar lag during deployment.

05 Identifying the data team's owners

The people who will own the migrated estate need to be in Client Review; ideally, they are involved from Discovery onward.

06 Agree on what success looks like

Pin down the definition of done and the exit criteria for Matillion ETL before Discovery; the parity, performance, and cost thresholds that demonstrate a workload has been migrated; and the conditions under which Matillion ETL can be switched off.

3.5 Testing *data outcomes*

Parity validation is the part of the migration that customers consistently underestimate. The mechanism is straightforward:

- **Run both estates in parallel where feasible:** the recommended approach is at minimum one full business cycle of the slowest-cadence pipeline (often a month-end close). This is the most rigorous way to validate parity, but it isn't free: for customers with long or infrequent cycles, a full parallel run can mean weeks of duplicated infrastructure and effort. Where the cost is prohibitive, a shorter window combined with targeted historical-data comparison can still build confidence, though with a corresponding increase in residual risk carried into cutover.

- **Compare data outputs** at a granularity meaningful to the downstream consumer; row counts and totals are necessary but not sufficient; sample-row comparisons across all important dimensions capture subtle divergences. Automate as much as possible via Maia.

- **Investigate every divergence:** Most are explainable (Matillion ETL had a bug, or a connector behaves differently in a way that is now actually correct). Some require remediation. Either way, document them.

- **Get explicit sign-off** from the downstream business owner before cutover, not just the data team.

Automating parity on Maia. Because Maia pipelines are versioned artefacts driven by an API, parity validation does not have to be the manual, spreadsheet-driven slog it is on legacy platforms. It sets up a test harness that runs both estates on the agreed schedule, lands their outputs in a common location, and applies the comparison rules: row counts, control totals, and sampled-row checks across the dimensions that matter, automatically surfacing only the divergences that need a human. The customer's job shifts from running comparisons to adjudicating exceptions.

The leverage comes from the guardrails being defined up front: the success criteria, the tolerance bands, the scenarios that must pass, and the condition that ends the testing window. These are the same definition-of-done criteria agreed upon before Discovery (§3.4). Settle them before the parallel run starts, and have the harness report continuously how close the estate is to exit, rather than leaving it to a subjective end-of-engagement judgement, which would collapse this phase from weeks of manual checking to a largely unattended run.

The harness, guardrails, and comparison scenarios built to validate the migration are not discarded at cutover; they become the regression-testing scaffold for every pipeline the data team builds afterwards. The assurance effort that legacy migrations treat as pure cost becomes a reusable capability the customer keeps: validation and assurance you automate once and re-run on demand, instead of repeating by hand every release.

3.6 Using the Maia Team and Maia Team assets *to accelerate any remediations*

Some remediation work requires a platform refactor, connector configuration change, or approach refactor (a new connector behaviour, an addition to the Python library, or an extension to the API surface). The engaged Matillion team can work with Maia Team assets generated from the migration pre-work on the customer's behalf or with the customer's technical owner(s) to apply remediation. The customer and the Matillion team should be aligned on who works with the Maia Team, scopes the change, and tracks it against the migration timeline.

Customers can accelerate this by being specific about which business outcome is at risk if the platform change does not land, and when; those two pieces of information drive prioritisation and help align Matillion-side and customer-side stakeholders.

3.7 Networking *during migration*

The Runner should run in the same network "location" as the Matillion ETL VM it is replacing. Same VPC, same subnet class, same route table, same egress posture. The reason: every source connectivity behaviour that Matillion ETL exhibits today is a function of where it sits in the network; if the Runner sits somewhere different, parity testing is fighting two variables at once (the migration and the network change).

You could decide to switch from Hybrid SaaS (customer-hosted runner) to Full SaaS (Matillion-hosted runner) during the migration. Field Engineering's recommendation is: don't, unless the cost/control trade-off has already been decided independently. It adds a variable to the parity story. Migrate first, then re-platform the runner if there is a business reason to do so.

04

Considerations

This section is for the technical owners who will operate the migrated estate.

4.1 Understanding *the Maia runner*

The Maia runner is the only Matillion-shipped component that lives in the customer's network under the Hybrid SaaS deployment model. It is a containerised, stateless workload that:

Communicates outbound-only with the Runner Gateway (the control-plane endpoint). The runner initiates an HTTPS connection, requests work, and receives responses. It never accepts inbound connections, which is what makes the security review far simpler than Matillion ETL's.

Pulls pipeline definitions from the project's Git repository to determine which pipeline and subsequent tasks to execute, and when (based on configured schedules or manual executions).

Executes against the customer's sources and warehouse using credentials retrieved from the Maia Foundation secret manager.

Reports the execution state back to the control plane.

Runners are designed to scale horizontally. Multiple runners attached to the same tenant can share work issued by the control plane based on project and environment configurations; capacity grows by adding or scaling runners, not by scaling a single runner vertically.

Two deployment models:

01 Maia Hybrid SaaS (customer-hosted runner):

The runner lives on one of the officially supported hosting targets: AWS Fargate, Azure Container App/ACI, Google Kubernetes Engine or Snowflake Native App. The customer owns the operational surface (deployment, scaling policy, observability hooks). This is the model that maps most naturally onto a Matillion ETL-replacement footprint when the warehouse, Matillion ETL VM, and sources live in AWS, Azure, Google Cloud, Snowflake, or the customer's on-prem network spaces.

---

02 Maia Full SaaS (Matillion-hosted runner)

The runner lives in Matillion-hosted infrastructure adjacent to the customer's tenant. Matillion owns the operational surface; the customer trades some control for a smaller infrastructure footprint. Suitable when the data sources are themselves Matillion-reachable SaaS services, or when the customer explicitly wants to outsource runner operations.

The choice between them is a customer policy decision and depends on where the data lives. From the pipeline's perspective, the two models are equivalent. Note that the officially supported managed-container surfaces are Fargate, Container App / ACI, Google Kubernetes Engine and Snowflake Native App; bare-EC2 and self-managed Kubernetes are not on the supported runner-target list.

4.2 Queue / Runner Gateway: *old vs new runner*

Matillion ETL pushed the work to its embedded scheduler and executed it in line. Maia uses a pull model: the control plane prepares work, and the runner requests it via the Runner Gateway over an outbound persistent HTTPS connection.

The implications:

- The control plane never needs an inbound connection to the runner: Network posture is outbound-only from the runner. Security reviews are dramatically easier than those for Matillion ETL, where inbound paths to the VM were typically required for the UI.
- Runner restarts are non-events: A restarting runner stops pulling for a moment; Any pending work waits on the control plane and is picked up when the runner returns.
- Backpressure is observable: Work depth is a first-class metric. When work accumulates faster than runners can drain it, you need more runner capacity, and that signal is visible, not inferred from “the VM looks busy”.
- Ordering is per pipeline, not global: Strict global ordering across all work is not promised. This is a non-issue for the overwhelming majority of pipelines, but it is worth knowing about.

4.3 Memory usage: *Python and Bash*

Matillion ETL ran Python and Bash as child processes within the JVM; memory was bounded by the VM’s RAM and the JVM’s tuning. The runner runs them in containers; the container’s limit bounds memory.

The practical implication is that pipelines with memory-intensive Python or Bash scripts will encounter container limits in Maia, resulting in an Out-of-Memory (OOM) error, in which the kernel terminates the container to protect the host. The failure is visible and bounded, a meaningful improvement over a VM under pressure, where a single memory-hungry job could destabilise everything running alongside it, but it does require remediation of the pipelines in question.

For customers with significant Python or Bash workloads, the recommended path is the **Script Pushdown component** paired with the **shared script runner:** a lightweight, separately sized interpreter that runs alongside the Maia runner and executes scripts via SSH, entirely outside the runner’s own memory envelope. The script runner is independently sized (small through xlarge, up to 32 GiB), so script memory allocation is decoupled from runner sizing. Field Engineering assesses Python and Bash usage during Discovery and recommends whether a script runner is warranted and how to size it.

4.4 Concurrency *model changes*

Matillion ETL ran pipelines as JVM threads within a single process. Maia runs pipelines as work items, distributed across runners and across processes within a runner.

The practical implications:

- **Thread-local state doesn't carry across pipeline steps, as some Matillion ETL jobs assume.** Where it matters, the Migration Agent translates the patterns; where it doesn't translate cleanly, remediation handles it.
- **Highly sequential pipelines feel network latency more.** Every step that round-trips to the control plane (for state or the next instruction) incurs that latency. A pipeline of 10,000 trivial sequential steps will be slower in Maia than in Matillion ETL, despite much higher throughput. The fix is structural: fewer, larger steps, and is exactly the kind of post-migration optimisation §1.3 referred to.
- **Concurrent pipelines are first-class, but third-party systems may not be.** Where Matillion ETL's scheduling constraints naturally throttled concurrency, Maia's distributed execution model removes that ceiling. For pipelines that read from SaaS sources, on-premises databases, or external APIs, this is a change to plan for: systems that were never stressed under Matillion ETL's serialised execution may now receive a volume of simultaneous requests they were not configured to handle. Rate limits, connection pool exhaustion, and API throttling are the most common symptoms. Field Engineering surfaces source-system concurrency limits during Discovery and recommends runner configurations and pipeline-level throttling controls where needed.

4.5 Scaling: *manual vs autoscaling*

Matillion ETL scaling by VM size: stop the VM, resize it, then start the VM. Manual, downtime-incurring, capped at the largest instance type you'll pay for.

Maia scales by runner count. Customers can run the native autoscaling provided by the chosen hosting target (Fargate task scaling, Azure Container App scale rules, Snowflake Native App scaling) based on load signals, or use manual scaling (a fixed runner count sized for peak). For most customers, a manual with a generous peak count is the right starting point; autoscaling is a second-quarter optimisation.

4.6 Performance: *round-trip latency*

Tightly sequential workloads: pipelines composed of many small steps, where each step requires confirmation of the previous step's completion from the control plane; feel the round-trip latency to the SaaS control plane. This is the single most common performance surprise.

The mitigation is architectural: consolidate small sequential steps into fewer, larger steps where the logic permits. Field Engineering flags this pattern during Discovery and diagnoses it during parity validation.

4.7 Runner specs *vs VM-based execution*

A Matillion ETL VM had GBs of RAM available to a single job by default. A runner container has a configured memory limit per pipeline instance. Workloads that assumed "infinite" RAM will need to be sized.

OOMing on the runner is recoverable (the pipeline fails, surfaces the OOM, and can be retried with a larger configured size). OOMing a Matillion ETL VM often brought down everything running on it. The new failure mode is better but different, and the team should be familiar with it.

4.8 System limits *in the Maia Foundation*

The Maia Foundation has tenant-level limits, including concurrent pipelines, pipelines per runner, schedule density, and API rate limits. These are generous for typical workloads, but very large estates can exceed them. Field Engineering sizes against these limits during Discovery; customers approaching them will be flagged, and the limit-extension conversation happens proactively, not at cutover.

4.9 *Connectors*

Connectivity is a constantly shifting world. Source APIs change, authentication models change, third-party SaaS deprecates endpoints, and warehouse vendors release new client libraries. Even a perfect deterministic translation of a Matillion ETL job's connector configuration can produce a different result in Maia if the upstream connector has been updated in the meantime. This is not a Maia-vs-Matillion ETL issue per se; it is a fact of connector life.

Field Engineering surfaces these as parity divergences during the parallel run and treats them as remediation work; sometimes a configuration tweak, sometimes a connector-team escalation, occasionally a customer-side data-source change. Plan for non-zero connector-driven remediation in any estate that uses third-party SaaS services.

05

Roadmap

What is Matillion doing to make the migration easier?

Active investment areas:

01 Migration Agent coverage continues to widen

Every engagement surfaces components or patterns that aren't yet handled automatically. Those become inputs to the Migration Agent itself, so the next customer's migration starts from a higher deterministic baseline than the last.

02 Public API expansion

Operational and remediation tasks that today require Matillion-side access are being migrated onto the public API surface, so customer teams can self-serve more of the workflow over time.

03 Wave-aware tooling for large estates

Larger migrations are driving investment in wave-scoped Discovery, Remediation, and reporting; so a multi-thousand-job estate can be migrated in coherent slices without losing visibility of the whole.

04 Observability and triage

Cross-product diagnostic tooling for Matillion ETL and Maia continues to grow, shortening the time-to-diagnosis for the most common remediation patterns and supporting the "is this a real divergence or a process artefact?" question that dominates parity validation.

05 Connector parity work

The Maia connector roadmap is directly informed by the long tail surfaced in real migrations.

The strategic direction is unambiguous: the migration experience is being made progressively more deterministic, more self-serve, and less dependent on bespoke human effort.

06

FAQ

What we have learned from migrations so far?

Q: How long does a typical migration take?

A: A mid-sized estate (a few hundred jobs, mainstream sources) is typically a 6-12 week elapsed engagement. The longest pole is rarely a Matillion effort; it is the customer-side calendar (network reviews, parallel-run windows, business-owner sign-off cycles).

Q: How much of my Matillion ETL estate will translate automatically?

A: For a typical estate, the majority of objects translate deterministically through the Migration Agent. The specific per-estate number is surfaced during Discovery, so you'll see your own figure before committing to a scope. The long tail is custom Python, exotic connectors, and architectural patterns that are worth revisiting anyway.

Q: Will my pipelines run faster on Maia?

A: For most workloads, comparable or somewhat faster, especially under concurrency. For highly sequential pipelines composed of many small steps, you may see worse single-pipeline wall-clock time due to control-plane round-trip times; the fix is to consolidate small sequential steps, which is exactly the kind of optimization Maia rewards.

Q: Can I migrate in waves?

A: Yes, and for large estates, this is the recommended approach. The toolchain has explicit wave-scoped support, and Field Engineering will scope waves with you during Discovery.

Q: Do I need to change my warehouse?

A: No. The warehouse is treated as an external dependency; Maia runs the same SQL against it via the same connector. Do not re-platform the warehouse during the migration; that is a separate program.

Q: What if I have very heavy custom Python?

A: This is the most common remediation tail. Maia runners support Python with a curated library set; libraries outside the set are handled on a case-by-case basis (custom runner image, code refactoring to use available libraries, or platform requests to add the library). Field Engineering flags this during Discovery.

Q: Will my schedules carry over?

A: Yes, schedules are one of the most deterministically translated object classes. Migration Agent recreates them in Maia Foundation; the customer activates them at cutover.

Q: What happens to my Matillion ETL audit history?

A: Matillion ETL execution history stays in Matillion ETL until decommissioning. If audit retention is a compliance requirement, capture it before decommissioning and store it in your normal compliance retention store; it does not migrate into Maia.

Q: Can we run Matillion ETL and Maia at the same time during migration?

A: Matillion ETL execution history stays in Matillion ETL until decommissioning. If audit retention is a compliance requirement, capture it before decommissioning and store it in your normal compliance retention store; it does not migrate into Maia.

Q: What if parity validation fails?

A: Investigate, remediate, and re-validate. Cutover does not happen until parity is signed off. The most common causes of failed parity are connector behaviour drift and topology choices that didn't translate cleanly; both are addressable through remediation.

Q: What does post-migration support look like?

A: A short post-cutover window (1–4 weeks) of Field Engineering engagement to triage any divergences the customer's data team finds. After that, the customer owns the estate and uses Matillion's standard support channels.

Q: Who do we contact if we want to start?

A: Your Matillion Customer Success or Account contact. They will engage Field Engineering, which will run Discovery as the first concrete step.