ZETALY®

WHITE PAPER

Mainframe to Cloud Migration: Pros, Cons, and What Actually Happens

Every year, hundreds of enterprises debate whether to migrate their mainframe workloads to public cloud. The promise is compelling — lower costs, greater agility, modern developer tooling, and freedom from IBM and Broadcom licensing contracts. The reality is considerably more complex.

This guide is a practical, honest treatment of mainframe-to-cloud migration: the genuine advantages, the real risks, the common failure modes, and the case studies — both successful and cautionary — that every decision-maker should know before committing to a migration program.

We've written this guide without a predetermined conclusion. Some mainframe workloads are excellent cloud migration candidates. Others are not. Most organizations land in a hybrid state for longer than they planned. Understanding where your workloads fit is the first and most important step.

In this guide, we will cover

- Why organizations consider migrating — and what they're actually hoping to achieve
- The real advantages of cloud migration for mainframe workloads
- The real risks and costs that migration advocates understate
- The four migration strategies and when to use each
- The hybrid model: mainframe and cloud working together
- Case studies: what successful migrations actually looked like
- Case studies: what failed migrations actually looked like
- How to decide: a framework for evaluating your workloads
- The role of mainframe FinOps during the migration journey

Also see: **What is Mainframe FinOps? the companion guide covering mainframe cost visibility, optimization, and governance.**
Download at Zetaly.io/MainframeFinops

1- Why Organizations Consider Migrating

The mainframe-to-cloud migration conversation is usually triggered by one of several catalysts:

- **Cost pressure**. IBM and ISV mainframe software costs — particularly MLC — escalate every year. For large enterprises, annual mainframe software spend can range from $10 million to over $100 million. When business leaders see these bills, cloud's pay-per-use model looks attractive by comparison.

- **Talent scarcity.** The COBOL developer population is aging out. The average COBOL developer is in their mid-50s (Computerworld). Finding, hiring, and retaining mainframe skills — z/OS system programmers, COBOL developers, capacity engineers — is increasingly difficult and expensive. Cloud platforms have a vastly larger developer talent pool.

- **Agility limitations.** Mainframe release cycles are slower than cloud-native development. Deploying new features on a mainframe system that processes millions of daily transactions requires extensive testing, coordination, and maintenance windows. Cloud-native architectures with CI/CD pipelines can deploy changes in minutes, not months.

- **Vendor lock-in concern.** A small number of vendors — IBM, Broadcom, and a handful of ISVs — control mainframe software pricing. Organizations that have watched Broadcom raise prices following its acquisition of CA Technologies in 2018 (Broadcom press release) are understandably wary of continued dependence on a platform where they have limited negotiating leverage.

- **Strategic direction.** Many enterprise technology strategies now mandate cloud-first or cloud-preferred approaches. Mainframe can feel inconsistent with that direction, regardless of its economics for specific workloads.

These concerns are legitimate. They are also frequently overstated in migration business cases — and the costs and risks of migration are frequently understated. Both sides of the ledger deserve honest scrutiny.

2- The Real Advantages of Cloud Migration

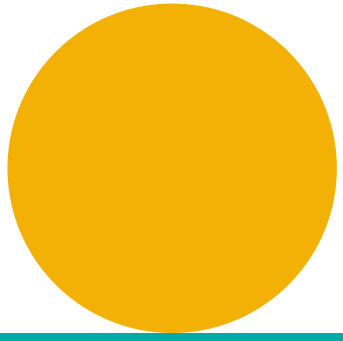
When mainframe migration succeeds, the benefits are real and measurable. The following advantages appear consistently across well-executed migration programs.

Lower compute costs — for the right workloads

Cloud's pay-per-use model genuinely delivers cost savings for workloads with highly variable demand patterns — workloads that spike during peak periods and sit idle otherwise. On mainframe, you pay for capacity whether you use it or not. On cloud, you pay only for what you consume.

> Meliá Hotels migrated its COBOL-based central reservation system from a mainframe to AWS and achieved 60% compute cost savings — «seven figures of savings,» in the words of Meliá's VP of Global IT. The reservation system had highly variable demand (peak periods around holidays and promotions) that made cloud elasticity genuinely valuable. (AWS — Meliá Hotels Case Study)

For high-throughput transaction processing workloads that run near-continuously, the cloud cost advantage is less clear. Mainframe's economics at sustained high utilization can be competitive with cloud, particularly when you account for network egress costs, cloud storage costs, and the operational overhead of cloud-native infrastructure management.



Modern developer tooling and faster release cycles

Cloud-native development using modern languages, containerized microservices, and CI/CD pipelines enables deployment velocity that is difficult to match on mainframe. Organizations that migrate to cloud and adopt DevOps practices consistently report dramatic improvements in deployment frequency and lead time.

> A consumer goods company that migrated 25 IBM mainframe applications to AWS using NTT DATA's modernization platform reduced manual deployment activity by 80%, eliminated deployment defects entirely (100% reduction), and converted weekly releases to daily and daily releases to hourly. (NTT DATA — Consumer Goods Company AWS Migration Case Study)

Access to cloud-native services

Cloud platforms provide immediate access to managed AI/ML services, data analytics platforms, serverless computing, and a rich ecosystem of SaaS integrations that are difficult or impossible to access from mainframe environments. For organizations building new customer-facing products, this ecosystem advantage is often the most strategically compelling argument for migration.

> Commonwealth Bank of Australia, which completed the migration of its core banking platform to AWS in 2025, cited AI capability as a central driver: «We're now making around 55 million AI-driven decisions every day across more than 2,000 models built on 157 billion data points.» The proximity of core banking data to the bank's AI infrastructure on the same cloud platform was a key design principle. (Commonwealth Bank)

Reduced vendor concentration risk

Migrating off mainframe reduces dependence on IBM and Broadcom, whose pricing power over mainframe customers is substantial. Following Broadcom's acquisition of VMware and the pricing changes that followed, many IT leaders have reassessed their exposure to any single vendor's licensing decisions. Cloud pricing, while not immune to increases, is subject to significantly more competitive pressure.

Talent market access

Cloud skills are widely available. The talent pool for AWS, Azure, and Google Cloud architects, DevOps engineers, and cloud-native developers is an order of magnitude larger than the mainframe talent pool. Organizations that successfully migrate gain access to a much broader hiring market and can build internal teams more sustainably.

3- The Real Risks and Costs Migration Advocates Understate

The business cases for mainframe migration often look compelling on paper and fall short in practice. The following risks and costs are consistently underestimated in pre-migration planning.

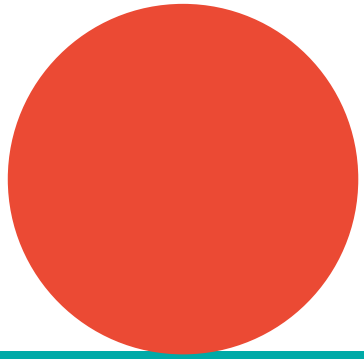
Migration projects take far longer and cost far more than projected

The single most reliable finding from mainframe migration case studies is that timelines and budgets are almost always exceeded — often dramatically.

> TSB Bank's 2018 core banking migration was planned for two years. It slipped repeatedly. When it finally went live on April 22, 2018, it failed catastrophically within hours. The UK's Financial Conduct Authority investigated and found that TSB's timeline had been set «based on very little information» before requirements were fully understood — a timeline described internally as «deliberately very ambitious» to act as a «forcing mechanism.» Three years of planning, 85 specialized subcontractors, multiple board-level reviews, and third-party audits could not prevent a failure that cost TSB £62 million in fines, over £32 million in customer compensation, and its CEO's job. (FCA Final Notice — TSB Bank, 2022)

The technical root cause was not incompetence. It was the inherent complexity of mainframe systems: decades of accumulated business logic, regulatory requirements, edge cases, and institutional knowledge embedded in COBOL, Assembler, Easytrieve, PL/1, and dozens of integration points with external systems. This complexity is invisible in migration planning and devastating in execution.

Even successful migrations demonstrate this pattern. Meliá Hotels set an «ambitious goal» to complete their migration in two years — a goal they hit only with intensive AWS support. Commonwealth Bank's 18-month migration required 54 testing crews and an international partner team co-located in Australia to overcome coordination challenges. These are the success stories.



The «last mile» problem: exotic technologies derail late-stage projects

Most mainframe applications contain a core that can be migrated with standard tools — CICS, COBOL, Db2 — but also a periphery of unusual technologies: Assembler, Easytrieve, CA Ideal, PL/1, IMS DB, IDMS, or VSAM files with complex access patterns. These «exotic» components are often underdocumented, poorly understood, and have no direct cloud equivalent.

Migration projects that proceed by focusing on the core and deferring the exotic components routinely discover late-stage blockers that invalidate earlier timeline estimates.

This «last mile problem» is one of the most common causes of cost overruns and project abandonment.

COBOL is harder to replace than it looks

COBOL code written in the 1970s and 1980s routinely encodes business rules that exist nowhere else — not in documentation, not in the heads of anyone still employed at the organization. When that code is refactored or re-platformed, subtle behavioral differences can emerge that are only discovered under production load conditions.

The FCA's investigation of TSB found that the Proteo4UK Platform required 221 applications, many of which had complex interdependencies involving COBOL, CICS, VSAM, Db2, IMS, and multiple batch job chains. Even with years of testing, the behavior of the live system under real customer load was different from what was tested — with catastrophic consequences.

Dual-running costs escalate during transition

During migration, organizations run both environments simultaneously — mainframe production and cloud build-out. This dual-running period can last years, and during it, costs are elevated compared to either steady-state scenario. Mainframe costs don't decrease during migration; they continue at near-full levels until workloads are fully cutover and decommissioned. Organizations that underestimate the dual-running period — or that achieve partial migration but cannot complete the final cutover — can end up paying both mainframe and cloud costs indefinitely, with neither environment fully optimized.

Compliance and regulatory requirements add complexity in regulated industries

Financial services, healthcare, insurance, and government organizations face regulatory requirements that significantly complicate cloud migration. Data residency rules, audit trail requirements, disaster recovery standards, and operational resilience regulations (such as DORA in the EU) all require careful architectural design and regulatory pre-approval for cloud migrations. Commonwealth Bank worked «closely with the Australian Prudential Regulation Authority (APRA)» to ensure their migration met regulatory requirements, including maintaining all banking data within Australian-based AWS data centers. This regulatory engagement was a multi-year process running in parallel with the technical migration. (Commonwealth Bank)

4- The Four Migration Strategies

Mainframe workloads can be moved to cloud in four fundamentally different ways. Each has different risk profiles, costs, and outcomes. Most large migrations use a combination of strategies across different applications.

Strategy 1: Rehosting («Lift and Shift»)

Rehosting moves the mainframe application to a cloud environment with minimal code changes, using emulation or compatibility layers that replicate the mainframe execution environment (CICS runtime, JCL batch execution, VSAM file handling) on cloud infrastructure.

When it works: Applications with stable business logic, high complexity, and a primary goal of reducing infrastructure costs. Rehosting is the fastest path and preserves existing code.

Limitations: The application continues to run on COBOL and the mainframe execution model — you gain cloud infrastructure cost benefits but not cloud-native development velocity or ecosystem access. Technical debt is deferred, not eliminated.

Example: NTT DATA's consumer goods company migration used UniKix, a CICS emulation platform, to rehost 25 mainframe applications on AWS. The project delivered 65% infrastructure cost reduction while preserving existing COBOL code — and completed ahead of schedule.

Strategy 2: Replatforming

Replatforming makes targeted code modifications to adapt the application to cloud infrastructure — changing database connectivity, replacing mainframe-specific file I/O with cloud storage APIs — without fundamentally redesigning the application architecture.

When it works: Applications with well-understood code where specific mainframe dependencies (VSAM files, JES2 batch) can be replaced with cloud equivalents without a full rewrite. A middle path between rehosting and refactoring.

Limitations: Requires deeper code analysis and carries more migration risk than rehosting. Behavioral differences between the original and modified code must be thoroughly tested.

Strategy 3: Refactoring (Re-architecting)

Refactoring redesigns the application for cloud-native architecture — decomposing monolithic applications into microservices, rewriting business logic in modern languages, and adopting cloud-native data storage and messaging patterns.

When it works: Applications slated for significant functional enhancement, new digital product development, or where the cloud-native architecture delivers genuine business value (elasticity, developer velocity, ecosystem access). The highest-ROI strategy when executed well.

Limitations: The highest risk, highest cost, and longest timeline strategy. Requires deep understanding of existing business logic to avoid introducing behavioral regressions. The TSB failure was fundamentally a refactoring project — the new Proteo4UK platform was not a copy of the old system, it was a redesign. When refactoring fails, the consequences are severe.

Example: Meliá Hotels refactored its COBOL-based reservation system into microservices on AWS EKS, achieving 60% compute cost savings and 75% improvement in time to market. (AWS — Meliá Hotels Case Study)

Strategy 4: Retain and Optimize

Not migrating is also a strategy — and for some workloads, the right one. High-throughput transaction processing, deeply regulated record-keeping systems, and workloads where the cost of migration risk outweighs the benefit of cloud are legitimate candidates for staying on mainframe with active FinOps-based cost optimization.

When it works: Workloads where mainframe's throughput/cost economics are genuinely competitive, where migration risk is high relative to benefit, or where the timeline to complete migration safely extends beyond the organization's planning horizon.

The right question: Not «can we migrate this?» but «should we migrate this, given the cost, risk, timeline, and benefit compared to optimizing it in place?»

5- The Hybrid Model: Mainframe and Cloud Working Together

For most large enterprises, the end state is not «mainframe» or «cloud» — it is both. The hybrid model, where mainframe handles the workloads it does best while cloud handles the rest, is the operational reality for the majority of organizations running IBM Z today. It is also, for many, the right long-term architecture — not merely a transitional phase.

Understanding how to design, operate, and optimize a hybrid mainframe-cloud environment is arguably more important than understanding how to migrate fully to cloud, because most organizations will live in the hybrid state for years or decades.

What the Hybrid Model Looks Like in Practice

In a hybrid architecture, the mainframe and cloud environments are connected but distinct. Workloads are placed on the platform best suited to their characteristics:

Stays on mainframe:

- High-volume, high-throughput transaction processing (core banking ledgers, payment clearing, insurance policy systems)
- Regulated record-keeping systems with deep auditability requirements
- Batch processing workloads that run near-continuously at high utilization
- Systems where migration risk is high relative to benefit, or where the COBOL/Assembler codebase is too complex to safely refactor

Moves to cloud:

- Customer-facing digital channels (mobile apps, web banking, online portals)
- Analytics, reporting, and data science workloads
- New product development and digital innovation
- Workloads with highly variable demand that benefit from cloud elasticity
- AI/ML inference and training pipelines
- DevOps tooling and non-production environments

The integration layer — APIs, event streams, and data pipelines connecting both environments — is where the hybrid model lives or dies. A well-designed integration layer allows cloud-native applications to consume mainframe transaction data in near real time. A poorly designed one creates latency, consistency problems, and security risk.

The Mainframe as a Transaction Engine, Cloud as the Experience Layer

The most common and successful hybrid pattern positions the mainframe as a high-throughput transaction and record-of-truth engine, while cloud handles the customer experience, analytics, and innovation layers above it.

In this model:

- A customer's mobile banking app runs on cloud-native infrastructure
- The actual account ledger, payment processing, and core transaction logic runs on mainframe
- APIs (typically REST or event-driven) expose mainframe capabilities to cloud applications
- Data is replicated or streamed to cloud analytics platforms for reporting and AI

This architecture is already the operating model for many of the world's largest banks, insurers, and retailers — even those that have never formally announced a «hybrid strategy.» They have cloud-native front ends and mainframe back ends, connected by an integration layer.

IBM's Hybrid Strategy: Cloud-Native on Z

IBM has explicitly embraced the hybrid model with several z/OS and LinuxONE capabilities designed to run cloud-native workloads on mainframe infrastructure alongside traditional workloads:

- **IBM z/OS Connect** enables REST API exposure of mainframe CICS and IMS transactions to cloud consumers — the mainframe becomes an API-accessible service
- **IBM Wazi** provides cloud-native development environments for z/OS, allowing developers to build and test mainframe applications using modern DevOps tooling
- **LinuxONE** runs Linux workloads — including containerized cloud-native applications — on mainframe hardware, combining mainframe reliability and security with open-source software stacks
- **IBM Z and Cloud Modernization Stack** provides tooling to integrate z/OS workloads with Red Hat OpenShift and cloud-native platforms

These capabilities reflect IBM's strategic positioning: the mainframe is not separate from cloud, it is a node in the hybrid cloud. (IBM — IBM Z Hybrid Cloud)

The Real Challenges of Running Hybrid

Hybrid is not free of complexity. Organizations that operate hybrid mainframe-cloud environments consistently face three categories of challenge:

1. **Cost visibility across two completely different billing models.** Mainframe costs are driven by MSU consumption under IBM pricing contracts. Cloud costs are driven by instance hours, storage, and egress fees under public cloud pricing. Very few organizations have unified cost visibility across both. Without it, hybrid total cost of ownership is poorly understood and difficult to optimize.

2. **Operational complexity.** Two platforms mean two sets of tools, two teams with different skills, two sets of monitoring and alerting, and two incident response playbooks. Integration points between mainframe and cloud are frequent sources of latency, availability, and data consistency issues.

3. **Data gravity and latency.** The mainframe holds decades of transaction history and is the system of record for the most critical business data. Moving that data to cloud for analytics is possible but introduces latency, replication lag, and synchronization challenges. Keeping data on mainframe while running analytics on cloud requires careful architectural design.

Hybrid FinOps: Managing Costs Across Both Platforms

One of the least-discussed challenges of hybrid mainframe-cloud is financial: how do you manage costs across two fundamentally different billing models, for two different audiences (mainframe ops vs. cloud ops), with two different sets of tools?

The answer is a unified FinOps practice that spans both platforms:

- **Mainframe side:** real-time MLC visibility, LPAR capacity optimization, and contract compliance tracking (as covered in the companion What is Mainframe FinOps? guide)
- **Cloud side:** standard cloud FinOps — rightsizing, reserved instance management, waste elimination via cloud cost management tools
- **Unified view:** cost allocation by business unit, application, or product across both environments — so that when a product team asks «what does our service cost to run?», the answer includes both the mainframe transaction processing and the cloud application layers

Organizations that achieve unified hybrid cost visibility make better architectural decisions — they can evaluate whether a workload is genuinely cheaper on cloud than on mainframe with accurate, apples-to-apples data, rather than comparing a fully-loaded mainframe cost against an incomplete cloud estimate.

6- Case Studies: What Successful Migrations Actually Looked Like

Meliá Hotels International — COBOL Reservation System to AWS

Workload: Central reservation system (CRS) built in COBOL on mainframe. Managed room inventory, bookings, loyalty programs, and tour operator contracts for 380+ hotels across four continents.

Why migrate: Prohibitive and growing per-transaction costs. Regular 4-hour maintenance windows causing online sales disruption. Inability to rapidly develop new digital booking features. Competitors were building cloud-native systems at lower cost.

How they did it: Partnered with AWS Enterprise Support and AWS Professional Services. Completed AWS Migration Readiness Assessment. Split monolithic CRS into independent microservices. Migrated 6 TB of core data to Amazon Aurora. Ran production workloads on Amazon EKS. Used AWS Spot Instances for cost optimization.

Timeline: 2 years — «ambitious» by their own description, achieved only with intensive AWS support.

Results: 60% compute cost reduction («seven figures of savings»). 75% improvement in time to market for new features. 99.99% application availability (vs. regular downtime previously). Response time reduced from 234ms to 160ms. (AWS — Meliá Hotels Case Study)

Key success factors: Well-defined, relatively self-contained workload. Clear business case with quantifiable cost pressure. Full commitment and executive sponsorship. Partnership with specialist migration expertise.

Commonwealth Bank of Australia — Full Core Banking Migration to AWS

Workload: Entire core banking platform — ledgers for deposits, loans, and transaction processing — supporting 16 million customers and 40% of payment flows across the Australian economy.

Why migrate: Aging mainframe infrastructure limiting AI and data capability development. Strategy to operate core banking and data platforms in the same cloud environment. Goal of enabling AI-driven personalization at scale.

How they did it: 18-month migration with partners SAP, SAP Fioneer, Accenture, AWS, and Red Hat. 54 separate testing crews. Offshore partner teams co-located in Australia. Kept configurations to vendor-supported standards to minimize customization risk. Engineered sub-two-minute failover capability. Executed final cutover in under 8 hours.

Results: Core banking platform now on AWS. Payment updates that took 6 seconds now update in real time. Making 55 million AI-driven decisions daily. Cost outcomes described as «neutral or slightly better» than legacy i nfrastructure. First major Australian retail bank to fully migrate core banking to public cloud. (Commonwealth Bank)

Key success factors: Multi-year preparation including extensive regulatory engagement with APRA. World-class partner ecosystem. Explicit FinOps modelling of cloud cost optimization. Strong internal governance and one unified team structure replacing eight siloed teams.

Consumer Goods Company — 25 Mainframe Applications to AWS

Workload: 25 IBM mainframe applications including corporate billing and reimbursement, processing over $8 billion in revenue per year.

Why migrate: Business-level decision to exit the data center. Needed to close mainframe as the last blocker to complete data center exit.

How they did it: Partnered with NTT DATA using their UniKix rehosting platform. Comprehensive discovery and target architecture design. Migrated CICS, Batch/JCL, COBOL, CA Ideal, PL/1, and Datacom applications. Single vendor approach for rehosting technology, migration services, testing, and support. Introduced DevOps processes post-migration.

Results: 25% reduction in operational support costs. 65% reduction in ongoing infrastructure costs. Data center exited ahead of schedule. 80% reduction in manual deployment activity. 100% reduction in deployment defects. Weekly releases now run daily; daily releases now run hourly. (NTT DATA — Consumer Goods Migration Case Study)

Key success factors: Clear goal (exit data center). Use of proven rehosting platform to minimize code risk. Single vendor accountability. Post-migration investment in DevOps to capture agility benefits.

7- Case Studies: What Failed Migrations Actually Looked Like

TSB Bank — The £62 Million Core Banking Migration Failure [TBC: reconcile with FCA Final Notice — £48.65m combined fine]

Workload: Core banking system supporting 5.2 million UK retail banking customers, including current accounts, loans, mortgages, and all digital banking channels.

Why migrate: TSB was divested from Lloyds Banking Group and needed to exit the Lloyds IT platform. Its new parent, Sabadell, had a modern banking platform (Proteo4UK) to migrate to.

What happened: In April 2018, after nearly three years of planning and development, 85 subcontractors, multiple board-level reviews, and third-party audits, TSB executed the cutover. Within hours, catastrophic failures emerged. Customers were locked out of accounts. Digital banking collapsed. Branch systems failed. The disruption lasted weeks.
The UK's Financial Conduct Authority (FCA) conducted a comprehensive investigation, publishing a 155-page report in December 2022. Key findings:

- TSB's timeline was set «based on very little information» before requirements were fully defined — described internally as «deliberately very ambitious» to act as a «forcing mechanism»
- The Proteo4UK Platform required 221 applications with complex interdependencies in COBOL, CICS, VSAM, Db2, IMS, IDMS, Assembler, Easytrieve, PL/1, and dozens of external integration points
- Testing was compressed to meet the public deadline, not expanded when complexity was discovered
- The «big bang» cutover approach with no rollback option meant that when it failed, there was no recovery path

Financial consequences: £62 million in regulatory fines [TBC: FCA Final Notice 2022 records the combined FCA+PRA fine as £48.65m] (FCA and PRA). Over £32 million in customer compensation. 225,492 customer complaints. CEO resignation. Hundreds of millions in remediation costs.

The systemic lesson: «You cannot know in advance how long a mainframe migration will take or what it will cost.» The FCA report found that TSB had «prepared a plan in circumstances where it had not yet finished defining its requirements.» Setting public timelines before understanding the full scope of work is not planning — it is wishful thinking that creates pressure to cut corners in testing. (FCA Final Notice — TSB Bank, 2022; DSS Blog — The TSB Mainframe Migration Disaster)

The «Belgian Insurer» — Unnamed Cautionary Example

In reporting on lessons from the TSB failure, analyst Steven Dickens of Futurum Group described a conversation with the CIO of a Belgian insurer that «completely decommissioned its 6,000 MIPS mainframe following a divestiture from a parent company.» The CIO «repeatedly used the word 'nightmare' during the discussion.» No further details are publicly available, but the characterization aligns with the pattern seen in TSB and elsewhere: legacy complexity that is invisible in planning reveals itself under execution. (FutureCIO — Lessons from Failed Mainframe Migration)

Common Failure Pattern: The Healthcare Provider and the National Retailer

mLogica, a mainframe modernization vendor, describes two instructive anonymized failures: A global healthcare provider proceeded with cloud migration without a comprehensive assessment of legacy system complexity and interdependencies. The result was unexpected service outages, regulatory scrutiny, and customer disruption. The specific failure mode: undiscovered complexity in component interdependencies that only manifested under production conditions.
A national retailer migrated systems without evaluating cloud readiness upfront. Systems that were not optimized for the new environment experienced performance issues, extensive rework, and delayed timelines — and the original migration business case was never recovered. (mLogica — Avoiding Costly Mainframe Modernization Failures)

8- How to Decide: A Framework for Evaluating Your Workloads

The decision to migrate, rehost, replatform, refactor, or retain a mainframe workload should be made at the application level, not the platform level. Different applications on the same mainframe may warrant different strategies.
The following framework provides a starting structure. It is not a formula — it is a set of questions that should be answered before committing to a migration strategy.

Step 1: Understand what the workload actually does and costs

Before any migration decision, you need accurate answers to:

- What business function does this application perform?
- How much does it cost to run on mainframe — by month, by transaction, by business unit?
- What is the throughput profile: sustained high utilization, or highly variable with peaks?
- How many lines of code? What languages (COBOL, Assembler, PL/1, others)?
- What are the external dependencies — other mainframe applications, downstream systems, batch jobs?
- When was the code last meaningfully documented?

Without this data, migration planning is speculation. Mainframe FinOps tooling (such as ZCC) can provide per-workload cost data; application portfolio analysis tools can map code complexity and dependencies.

Step 2: Assess migration risk

Risk factor	Lower risk	Higher risk
Code age	Written post-1990, well documented	Written pre-1980, undocumented
Languages	Standard COBOL, Db2	Assembler, Easytrieve, CA Ideal, PL/1
External dependencies	Few, well-defined	Many, poorly documented
Data complexity	Relational Db2, standard schema	VSAM, IMS DB, IDMS, complex access patterns
Business logic documentation	Complete functional specs	Logic exists only in code
Regulatory requirements	Low or well-understood	High, complex, subject to regulatory review
Failure tolerance	Can tolerate disruption during cutover	Zero-downtime requirement

Step 3: Assess migration benefit

Benefit factor	Higher benefit	Lower benefit
Demand variability	High peaks, significant idle time	Near-constant high throughput
Cost delta	Cloud unit economics significantly better	Mainframe economics competitive at volume
Development velocity need	Rapid feature development required	Stable, infrequently changed system
Cloud ecosystem value	AI/ML, analytics, SaaS integration high value	Self-contained processing, no cloud ecosystem benefit
Talent situation	Mainframe skills unavailable, cloud skills accessible	Mainframe team stable and skilled

Step 4: Make the workload-level decision

Risk/Benefit profile	Likely strategy
Low risk, high benefit	Refactor — invest in cloud-native redesign
Low risk, moderate benefit	Replatform or rehost — migrate but don't redesign
High risk, high benefit	Rehost first, then refactor incrementally — don't attempt big-bang refactoring
High risk, low benefit	Retain and optimize — mainframe FinOps, defer migration
Any profile, short timeline	Be very cautious — compressed timelines are the primary cause of migration failure

Step 5: Plan for reality, not the best case

Every mainframe migration plan should be stress-tested against the following questions:

- What happens if this takes twice as long as planned? Can the organization absorb the dual-running costs?
- What is the rollback plan if the cutover fails? Is a rollback technically possible?
- Have we identified all the «exotic» components, or are we assuming they'll be manageable?
- Is the projected cost savings based on eliminating all mainframe costs, or does it account for residual dependencies?
- What does regulatory engagement look like — and have we started it?

9- The Role of Mainframe FinOps During the Migration Journey

Whether you ultimately migrate some, all, or none of your mainframe workloads to cloud, mainframe FinOps serves an essential function during the decision-making and execution period.

Before migration: Build the cost baseline

You cannot make a credible migration business case without knowing what your mainframe workloads actually cost — per application, per business unit, per transaction. Monthly License Charges alone don't give you that picture; you need cost allocation at the workload level. Mainframe FinOps tooling provides this baseline. It is the financial foundation of any migration analysis: the «as-is» cost that migration savings must be compared against.

During migration: Fund the journey and prevent dual-paying

The dual-running period is where mainframe migration budgets are most vulnerable. Active mainframe cost management during this period serves two purposes:

- **Fund the cloud build-out.** MLC savings generated through optimization free up budget that can be redirected to cloud migration investment — making the migration more self-funding.
- **Catch residual costs.** As workloads migrate, mainframe costs should decrease. Without FinOps monitoring, organizations frequently discover that mainframe costs haven't decreased despite nominal migration — because residual dependencies (batch jobs, data feeds, integration layers) kept the mainframe running near full capacity. Cost visibility makes these situations visible before they become budget surprises.

After partial migration: Manage the hybrid state

Most organizations end up in a hybrid mainframe-and-cloud state for longer than planned. FinOps governance across both environments — tracking what each workload costs on each platform — is essential for making informed decisions about what to migrate next, what to optimize in place, and when the mainframe can finally be decommissioned. The organizations that manage hybrid state well do so with clear cost visibility on both sides of the equation. The ones that struggle are typically those that have cloud cost management working but mainframe costs still treated as a fixed, opaque overhead.

10- Frequently Asked Questionselines

How long does a mainframe-to-cloud migration typically take?

Timelines vary significantly by workload complexity and strategy. Simple rehosting of a single well-documented application can complete in months. Full core banking migrations — like Commonwealth Bank's — take 18 months to three years with significant resources. Complex refactoring projects routinely take longer than planned. Planning for twice your initial estimate is not pessimism; it is calibration based on the available evidence.

How much does mainframe-to-cloud migration cost?

Migration costs vary enormously depending on workload complexity, strategy chosen, and whether the organization uses specialist migration partners. Simple rehosting projects can cost millions; full core banking refactoring projects cost hundreds of millions when you include partner fees, dual-running costs, testing, and staff time. TSB's failed migration and its consequences cost the organization well over £100 million all-in. Migration business cases should include full lifecycle costs, not just consultant fees.

Can all mainframe workloads be migrated to cloud?

Technically, almost any workload can be migrated given sufficient time and investment. The more relevant question is whether migration is economically justified for a specific workload. High-throughput, low-latency transaction processing workloads where mainframe economics are competitive, deeply embedded COBOL systems with poor documentation, and systems where migration risk is high relative to benefit are often better candidates for ongoing optimization than for migration.

Is it possible to migrate in phases rather than all at once?

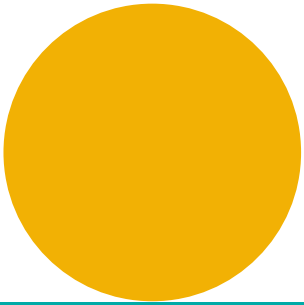
Yes — and phased migration is strongly recommended over «big bang» approaches. The TSB failure was in part a consequence of a single cutover with no rollback option. Phased migration allows organizations to learn from early workloads before tackling the most critical ones, maintains rollback capability, and allows dual-running costs to be managed progressively.

What should we do about mainframe costs while planning a migration?

Implement mainframe FinOps. The planning and preparation phase for mainframe migration typically takes longer than organizations expect — and during that period, mainframe costs continue to run. Optimization during this phase funds the migration and builds the workload cost data needed to make informed migration prioritization decisions. Waiting until migration is complete to address mainframe costs means leaving significant savings on the table.

What is the most common reason mainframe migrations fail?

Based on the available case studies, the most common cause is underestimating complexity — particularly the «exotic» components (non-standard languages, complex batch dependencies, undocumented integrations) that sit at the periphery of the main application. These are discovered late in the project, after timelines and budgets are set, and they derail schedules. Comprehensive upfront assessment — including automated code analysis — is the primary mitigation.



11. References and Further Reading

Migration Case Studies

- Commonwealth Bank — Core Banking Cloud Migration Case Study (March 2026)
- AWS — Meliá Hotels International Mainframe Migration Case Study
- NTT DATA — Consumer Goods Company AWS Mainframe Migration
- FutureCIO — Lessons from Failed Mainframe Migration (TSB analysis)
- DSS Blog — The TSB Mainframe Migration Disaster
- FCA / PRA — Final Notice: TSB Bank plc (December 2022)

Migration Risk and Challenges

- mLogica — Avoiding Costly Mainframe Modernization Failures
- ModLogix — Mainframe to Cloud Migration Challenges
- Forbes Technology Council — How Do Mainframes Fit in the Cloud Era?
- Candela Data — Mainframe to Cloud Migration Challenges (Australia)

Industry Research

- Forrester — The State of Mainframe, Global, 2025
- Mordor Intelligence — Mainframe Market Size, Growth & Outlook (2026)

IBM — Hybrid Cloud

- IBM — IBM Z Hybrid Cloud
- IBM — z/OS Connect
- IBM — LinuxONE

Zetaly Resources

- What is Mainframe FinOps? (companion guide)
- Free ZAC Simulation — Mainframe Savings Potential Assessment
- Maximizing Savings: Transitioning from CMP to TFP
- How to Optimize Your TFP Environment for Maximum Efficiency

Before You Commit to a Migration Program

If you are actively evaluating mainframe migration, the most valuable thing you can do before finalizing your business case is to establish accurate per-workload cost data. Without knowing what each application costs to run on mainframe, you cannot build a credible comparison against cloud alternatives.

Zetaly's ZCC platform provides that cost baseline — real-time visibility into mainframe software costs at the LPAR, application, and business unit level. It is the financial foundation any migration analysis needs, and it generates immediate value regardless of whether migration proceeds.

Explore ZAC for Automated LPAR Capacity Optimization and MLC reduction

Get a free ZAC Simulation
a personalized report showing your potential savings based on your actual environment data.
No commitment required

Explore ZSI for Application-Level Cost Insight

Explore ZSI to See What Each Application Costs

CONTACT US

zetaly.io

contact@zetaly.io

Zetaly/In

+33 (0)2 22 06 88 60

Document 10011 RevA 20260525