



EBOOK

Beyond the Toggle

Feature Flags as Your Migration Safety Net



Table of Contents

1. Executive Summary	3
2. What Is a Cloud Migration?	4
3. The Common Challenge: Risk at the Point of Release	5
4. Feature Flags in Cloud Migrations	6
4.1 Replace the Big Bang with Progressive Rollout	6
4.2 Measure What Matters	7
4.3 Test in Production, Safely	8
5. What to Look for in a Feature Flag Platform for Migration	9
5.1 Integration with the Delivery Pipeline	9
5.2 Experimentation and Metrics Built In	9
5.3 Enterprise-Grade Governance	10
5.4 Developer and Operator Friendly	10
6. Conclusion: Migrate with Confidence, Not Risk	11
Referenced Articles & Sources	12

Executive Summary

Enterprise cloud migrations are notoriously complex — high-risk, high-impact efforts that often span legacy systems, monoliths, microservices, new APIs, and UI changes. Too often, these transitions rely on “big bang” releases that tie deployment and release into a single moment of failure. One bad rollout can mean downtime, broken user experiences, or rollback chaos.

Feature flags provide a smarter path. By decoupling deployment from release, they allow teams to ship code without activating it — enabling gradual rollouts, targeted testing, instant rollbacks, and production experimentation. This shift turns risky migrations into measured, reversible changes.

Harness makes this approach enterprise-ready. With integrated **Feature Management and Experimentation**, Harness provides the abstraction layer and observability teams need to migrate safely and strategically. Cloud functionality can be released incrementally, tested against production metrics, and rolled out when it’s proven — not just when it’s shipped.

This paper explores how organizations can use Harness to support cloud migrations of any kind: infrastructure changes, service breakouts, API replacements, or complete re-platforms. Along the way, we’ll look at common failure patterns, modern rollout strategies, and real-world success stories — all pointing toward a simple truth: anything can (and should) be behind a feature flag.

“More than half of all transformations fail to achieve their intended benefits within three years.”

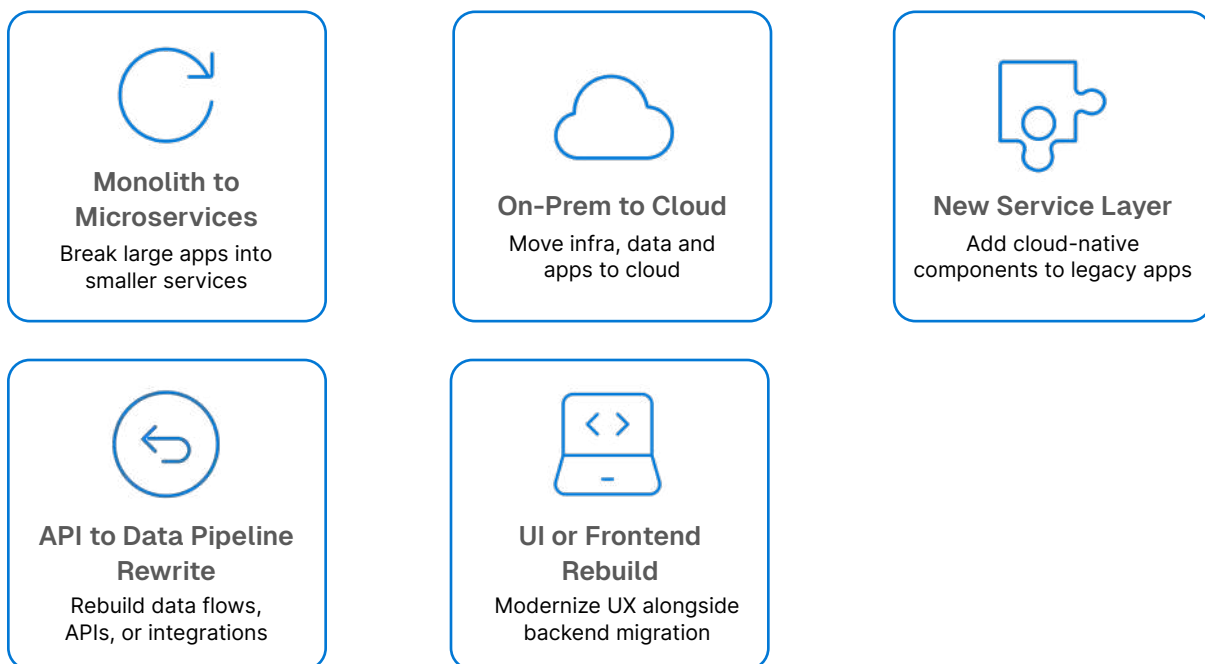
— BCG, [Overcoming Cloud Migration Challenges](#)

2. What Is a Cloud Migration?

“Cloud migration” sounds straightforward, but in practice, it’s rarely a single, linear task. For most enterprises, it’s a layered, multi-phase transformation — often involving **code, infrastructure, data, and architecture changes all at once**.

Many teams attempt to deliver a new UI, deploy a new service layer, and switch to cloud infrastructure in the same release. This creates tightly coupled dependencies, increases complexity, and amplifies the blast radius of failure.

To make things more concrete, here are the **five most common forms of cloud migration** encountered in the enterprise:



These changes rarely happen in isolation — and that’s exactly the problem. When multiple transformations converge into a single deployment event, risk multiplies. Without safeguards in place, even a small regression can cause cascading outages or user-facing issues. And yet, due to pressure to “just ship,” many teams go live without full validation or fallback paths.

That’s why cloud migration isn’t just a technical project — it’s a **delivery strategy problem**. And solving it requires tools that support **incremental rollout, testing in production, and controlled exposure**. This is where feature flags become essential.

3. The Common Challenge: Risk at the Point of Release

“Make a note of your current KPIs—page load time, CPU usage, memory used percentage, for example—and plot targets for improvement post-migration.”

— Jefferson Frank, Top Reasons Cloud Migrations Fail

Most cloud migrations fail not because of bad architecture or poor engineering — but because of **how and when changes are released**.

Teams are often forced to launch too much at once. A new service goes live, infrastructure flips to the cloud, and a redesigned UI is deployed — all in one coordinated event. There’s no safe way to test in production. No gradual exposure. No rollback strategy beyond redeploying the old stack, which might not even be compatible anymore.

This is the essence of the **“big bang”** problem: every change is tightly coupled, and release becomes an all-or-nothing moment. If something breaks, it breaks big — and users feel it.

Even worse, many teams don’t have a reliable baseline to compare behavior before and after migration. Are things better? Faster? More stable? Can it scale more easily to meet future demand? There’s no visibility into what changed, or what worked.

This challenge isn’t solved with better code — it’s solved with better release mechanics. Feature flags provide a safety net by allowing incremental exposure, rollback without redeploy, and experimentation in production. They help teams answer the most important question during migration: **Are we making things better, or just different?**



4. Feature Flags in Cloud Migrations

Feature flags are a strategic control layer that separates **code deployment** from **feature release**. Instead of flipping the switch for your entire user base, you can target specific segments, roll out incrementally, and monitor behavior before going fully live. This is especially valuable during cloud migrations, where the risk of unknown failure modes is high.

Here's how feature flags solve key migration challenges:

4.1 Replace the Big Bang with Progressive Rollout

Cloud migrations often fail because teams attempt to do too much at once — pushing infrastructure, services, and UI changes live in a single “big bang” event. Feature flags eliminate this risk by enabling **progressive rollout**: the ability to ship changes gradually, expose them selectively, and scale confidence before going full production.

You can ship cloud-based services into production without exposing them to users, route only a fraction of traffic to the new environment, and instantly flip back if anything breaks. This gives your team control over **when, how, and to whom** new functionality is released.

What Is the Strangler-Fig Pattern?

“**Strangler Fig**” is a term coined by software architect Martin Fowler to describe a safe, incremental approach to replacing legacy systems.

Instead of rebuilding everything at once, you build new functionality **alongside** the existing system. Over time, more and more traffic is routed to the new cloud-based components, and legacy code is gradually phased out — like a strangler fig tree that wraps around and eventually replaces its host.

Feature flags make this pattern executable in production, letting teams direct traffic to new services, test functionality safely, and expand adoption at their own pace.

This pattern — enabled by feature flags — allows you to modernize your architecture piece by piece, without incurring the risk of a one-shot cutover.

4.2 Measure What Matters

When you're migrating systems to the cloud, it's not enough to ask “**Did it work?**” The real question is: “**Is it better?**” That means tracking performance, cost, and reliability — and doing it in production, where real behavior emerges.

There are many metrics you want to consider for any given application or migration but we wanted to share a story of one of our own migrations and how we measured (and improved) our own SaaS platform.

At **Split**, we used feature flags to do exactly that during a recent pipeline migration. Our original AWS S3–inbound data pipeline was batch-based and built on Spark. It worked — but as our data volumes grew, so did the latency and cost. We needed to modernize.

We built a new pipeline using streaming architecture, copy jobs, and efficient ingestion via Databricks Autoloader. But instead of flipping the switch and hoping for the best, we ran both pipelines in production — side-by-side — with a **feature flag routing data traffic** between them.

This allowed us to **compare key metrics in real time**:

Metric	Legacy Pipeline	New Pipeline (Behind Flag)
Processing Time	Spark batch jobs; longer latency	Streaming; significantly faster load intervals
Compute Cost (DBU)	High and scaling with volume	Lower and more stable due to architecture = Reduced Support Calls
Customer Visibility	Minimal status detail	Enhanced status files with real-time metadata = Improved NPS
Rollback Plan	Manual, risky	Feature flag fallback to legacy pipeline in less than a second

The feature flag wasn't just a gate — it was an **instrumentation point**. We could track improvements as they happened, with full confidence that we could reverse course at any time.

Once the new pipeline consistently outperformed the legacy path, we scaled traffic gradually until we were confident enough to retire the old flow. **No big bang. No surprises. Just data.**

4.3 Test in Production, Safely

Dev and staging environments can't match production traffic, data volume, or edge cases. Feature flags give teams a way to test cloud services under real conditions — but with guardrails.

You can:

- Limit exposure to internal users or test accounts.
- Use segmentation to isolate high-value or low-risk users.
- Kill the flag immediately if problems arise.



Harness Advantage

Harness Feature Management makes all of this easier with:

- [Automate associated database changes with DB DevOps](#)
- [CI/CD and GitOps integration](#)
- [Enable FinOps to tightly observe cloud cost impact pre- and post-migration](#)
- [Integrate chaos testing](#) into the release to test the unknown
- [Extend experimentation to infrastructure changes](#)
- Enterprise-grade governance and audit trails

Together, these features turn flags from a dev hack into a first-class part of your migration strategy.

5. What to Look for in a Feature Flag Platform for Migration

Not all feature flag tools are created equal — and during a high-risk cloud migration, the wrong limitations can become blockers. A mature, enterprise-ready system needs to do more than just toggle features. It should provide **end-to-end support for safe rollout, observability, and governance** at scale.

5.1 Integration with the Delivery Pipeline

A flagging system should work **in concert with your CI/CD pipeline**, not around it. Look for the ability to:

- Trigger deployments conditionally based on flag state.
- Use flags to gate release stages or roll back automatically.
- Align feature rollouts with observability and performance checks.

5.2 Experimentation and Metrics Built In

Feature flags should support **more than just on/off switches**. Teams should be able to:

- Attach key performance metrics (latency, error rate, conversions) to any flag.
- Run A/B tests or phased rollouts with measurable impact.
- Make data-driven decisions about when to promote or revert a change.

5.3 Enterprise-Grade Governance

At enterprise scale, **auditing and access control** matter just as much as functionality. Look for:

- Role-based permissions on who can edit or toggle flags.
- Approval workflows for production changes.
- Detailed audit logs to track who changed what — and when.

5.4 Developer and Operator Friendly

Finally, a good system should support **modern workflows and tools**, including:

- SDKs in common languages
- GitOps integration and CLI support
- A clear, usable UI for targeting, segmentation, and flag cleanup

Flags shouldn't feel like a bolt-on. They should be part of the way your team builds, tests, and ships software — especially during something as sensitive as a cloud migration.



6. Conclusion: Migrate with Confidence, Not Risk

Cloud migration doesn't have to be a high-stakes gamble. The real risk isn't in the architecture — it's in the release. When teams deploy cloud infrastructure, new services, and modernized UIs all at once, they create a single point of failure.

Feature flags eliminate that risk. By decoupling deployment from release, they give you control over rollout, visibility into impact, and the ability to reverse course without disruption. They turn migrations into manageable, measurable steps — and allow teams to adapt in real time as complexity unfolds.

But to make that possible, your flagging system needs to be more than a toggle. It needs to integrate with your delivery pipeline, support real-time metrics, enforce governance, and fit the way your team builds and ships software.

Whether you're lifting a monolith, replacing APIs, or gradually replatforming your entire stack — **anything can (and should) be behind a feature flag.**

Next Steps

- **Audit your current release strategy:** Are you relying on “big bang” deployments? Are you measuring changes?
- **Baseline your metrics** before you migrate.
- **Evaluate your feature management tooling:** Does it support progressive rollout, observability, experimentation, and rollback?
- **Start small:** Put your next cloud service or infrastructure change behind a flag — and test it safely.

Referenced Articles & Sources

1. Boston Consulting Group (BCG)

- **Quote:** “More than half of all transformations fail to achieve their intended benefits within three years.”
- **Use:** Appears in the Executive Summary as a key stat to underscore the high failure rate of transformation efforts.
- **Source:** [Overcoming Cloud Migration Challenges – BCG](#)

2. Jefferson Frank

- **Quote:** “Make a note of your current KPIs—page load time, CPU usage, memory used percentage, for example—and plot targets for improvement post-migration.”
- **Use:** Callout in Section 3 to support the idea of needing a performance baseline before migration.
- **Source:** [Top Reasons Cloud Migrations Fail – Jefferson Frank](#)

3. Martin Fowler

- **Concept Referenced:** Strangler Fig Pattern
- **Use:** Explained in Section 4 as a best-practice migration strategy enabled by feature flags.
- **Original Source:** Strangler Fig Application – [MartinFowler.com](#)



The AI-Native Software Delivery Platform™

Follow us on

 /harnessio

 /harnessinc

Contact us on

www.harness.io

