



The App Modernisation Playbook

A guide to modernising on-premises
and legacy applications



Contents

03

Modernise with confidence

- 04 Proven outcomes with agentic modernisation
- 04 Four proven patterns for secure, scalable, cloud-native apps

05

Get started and assess your digital estate

- 05 Fast-track discovery for feasibility
- 05 Quick assessment tools
- 06 GitHub Copilot modernisation
- 06 Azure Copilot migration agent
- 06 Use a quick prioritisation framework

07

Select the right target architecture

- 08 Containers
- 09 API-first
- 10 Event-driven
- 11 Microservices

12

Plan your data strategy

13

Choose a modernisation pattern

- 14 Rehost
- 15 Replatform
- 16 Refactor
- 17 Azure services at a glance

18

Modernise with containers: A decision framework

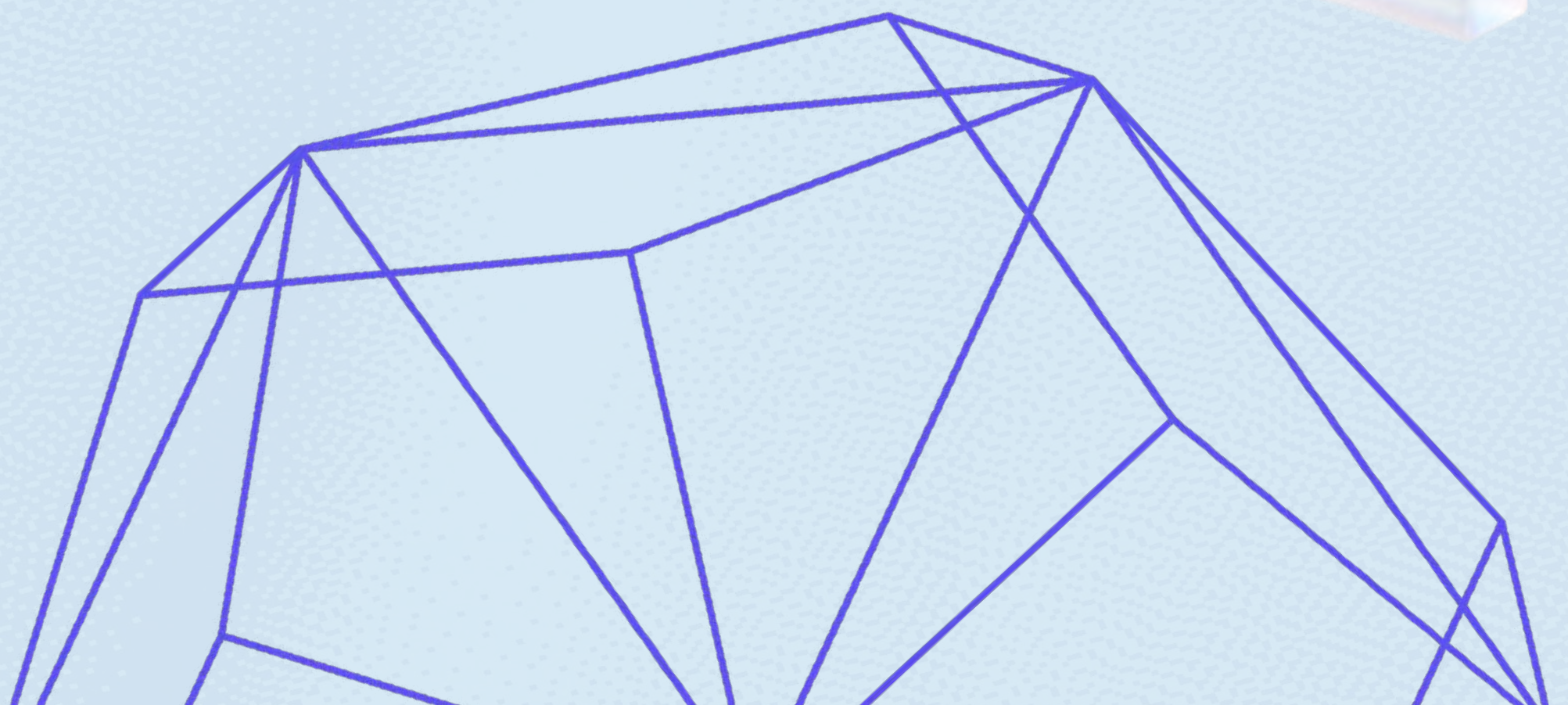
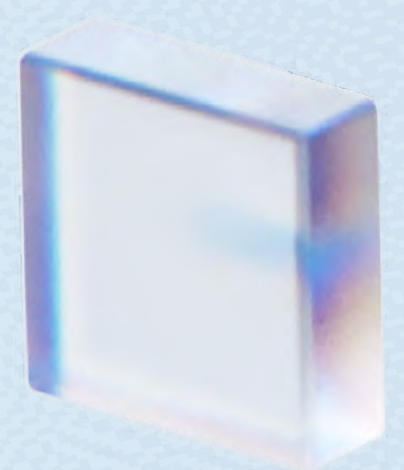
20

Execute and optimise

- 20 Core modernisation practices
- 21 Optimise costs
- 21 Drive continuous improvement

22

Next steps



Modernise with confidence

Organisations that have modernised their application estates are seeing 66% more performance per dollar for transaction processing workloads.¹ AI readiness – the ability to integrate intelligent services, run agentic workflows and leverage real-time data – is now the leading business driver for application modernisation.

Modernisation has long been slow, risky and resource-intensive. That's changing. Agentic AI tools can now assess, plan, upgrade and migrate applications in days instead of months, handling code transformations, dependency analysis, build validation and Azure deployment within the developer's IDE. Modernisation is no longer the bottleneck.

The question isn't whether to modernise. It's which workloads move first and what AI capacity those moves free up.

The Application Modernisation Playbook moves you past planning to specific tactics for implementing cloud-native patterns. Decision guides, checklists and actionable frameworks help you assess your application portfolio, select the appropriate cloud target and execute modernisation with minimal risk.

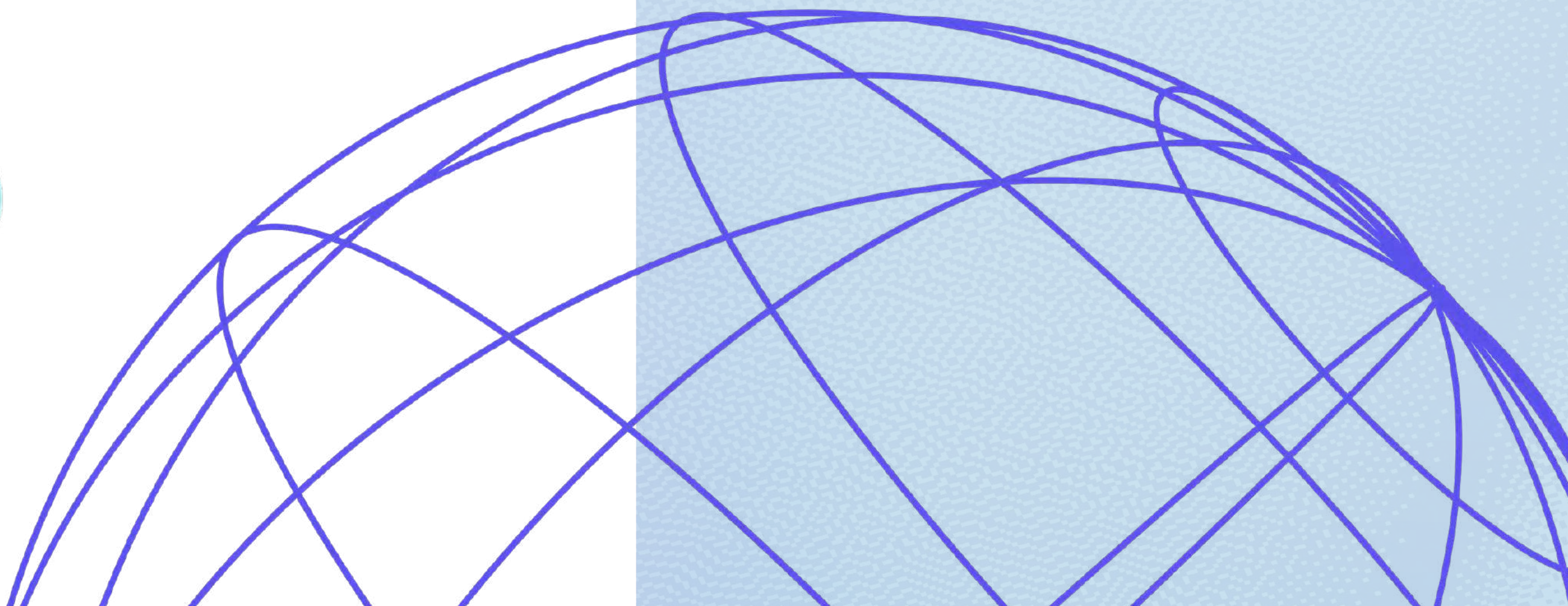
4×

faster overall modernisation with agentic AI²

70%

less migration time with guided execution³

Faster AI capability deployment after modernisation



Proven outcomes with agentic modernisation

With agentic AI, modernisation becomes a persistent system capability rather than a one-time programme.

- Up to **70% less migration** time with AI-guided planning and execution.⁴
- **50% less effort** required to upgrade legacy applications.⁵
- **500,000+ lines of code modernised in weeks**, not years, at a Fortune 500 company.⁶
- **4× faster overall modernisation** with agentic AI tools.⁷

No matter where you start, there's a proven path forward.

Agents span .NET (to .NET 10), Java (8 to 25, EE to Jakarta), mainframe (COBOL, CICS, JCL), VMware, Windows, Linux and all major databases. Application-aware migration discovers, groups and migrates dependent workloads as cohesive units, not one server at a time.

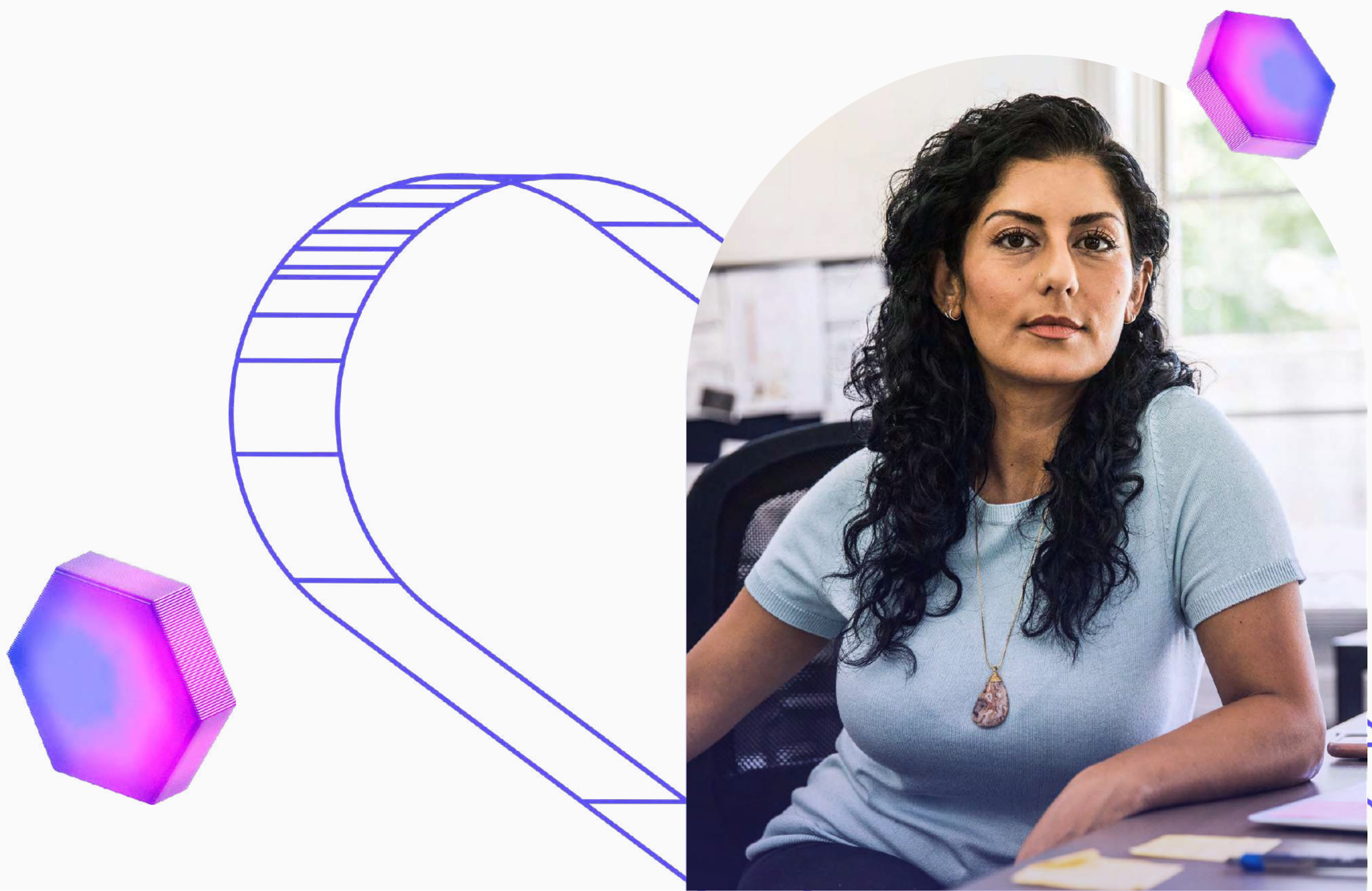
Four proven patterns for secure, scalable, cloud-native apps

Four architecture patterns stand out for their business and technical value:

- Containerisation
- An API-first approach
- Event-driven designs
- Microservices

These patterns pave the way to advanced capabilities like AI integration, agentic workflows, real-time analytics and DevOps automation. Each creates the foundation for embedding intelligent services – from retrieval-augmented generation (RAG) to autonomous agents that orchestrate business processes.

You can accelerate the transformation using **GitHub Copilot modernisation**, which automates code analysis, framework upgrades, containerisation and Azure migration directly in Visual Studio and Visual Studio Code.



Get started and assess your digital estate

A prioritised list of applications shows you where modernisation can drive the most value for your organisation. But many teams stall at this step, lacking a thorough assessment of their applications and data.

A clear, actionable view of your digital estate shows you what you have, how it's connected and which workloads offer the highest value for cloud-native transformation. You can also assess your organisational readiness. Skills, like schedules and budgets, affect your priorities.

Fast-track discovery for feasibility

Begin your modernisation journey with AI-guided discovery. Instead of manual inventories and static assessments, AI agents intelligently analyse your estate, mapping applications, dependencies, integration points and infrastructure patterns in real time. They surface modernisation opportunities, model cost scenarios and highlight risk before you move. With intelligent workload prioritisation and predictive landing zone recommendations, you can sequence migrations strategically, reducing dependency conflicts and unexpected rework during execution.

Quick assessment tools

✓ Create an inventory of applications and dependencies

Use AI-powered discovery and dependency maps. Agentless or agent-based dependency analysis groups components that must move together so you avoid stranded dependencies. You get a list of discovered assets, dependency maps, performance baselines, cost estimates and readiness flags.

✓ Assess high-level feasibility and costs

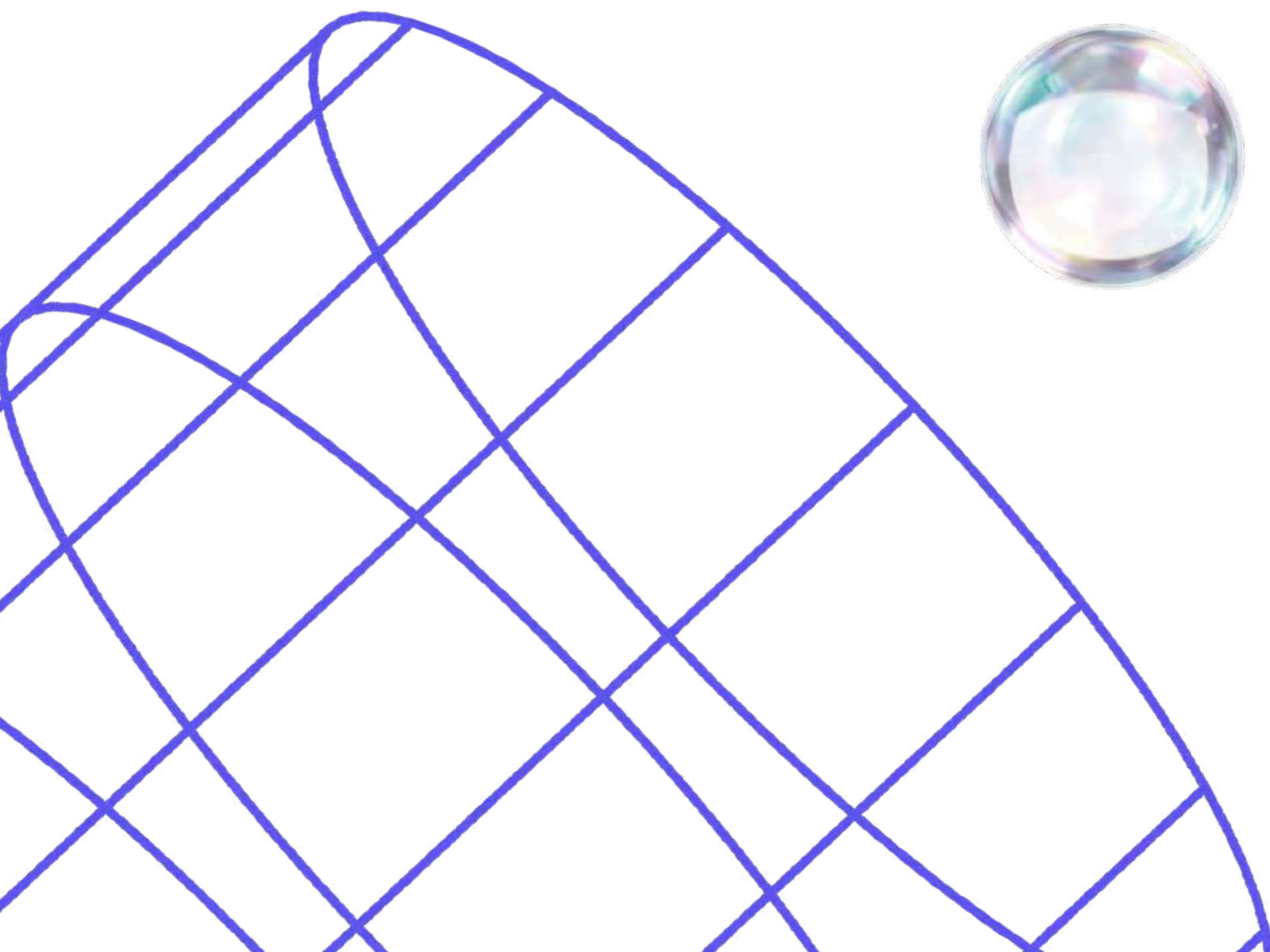
Use AI-guided assessment to get rough cost and sizing estimates and a fit assessment for the cloud. Combine with the Azure Pricing Calculator for service-specific information.

✓ Assess code-level modernisation

Scan source code and configuration files to assess complexity and provide migration recommendations. With AI-powered modernisation agents, application owners and architects can assess multiple apps at once, analysing code, dependencies and configuration to generate cloud-aligned migration plans at scale. Those assessments integrate with planning tools and flow directly into GitHub, where developers can pull them into their IDE and begin executing immediately.

✓ Assess risk and fit

Review the [Azure Well-Architected Framework](#), a self-service assessment that helps you score a workload's reliability, security, cost, operations and performance and export results to a backlog.



GitHub Copilot modernisation

GitHub Copilot modernisation is an agent-powered capability that helps you modernise applications end to end, from assessment and planning through code transformation, testing and deployment. It works directly within GitHub and developer workflows.

At the individual application level, you can:

- Assess applications and dependencies.
- Generate application-specific modernisation plans.
- Transform code, configuration and infrastructure.
- Validate changes and deploy directly to Azure.

Across the workflow, Copilot applies AI-driven guidance at each phase, helping you automate repetitive tasks, reduce manual effort and work directly within GitHub and IDE workflows with full control over review and execution.

Azure Copilot migration agent

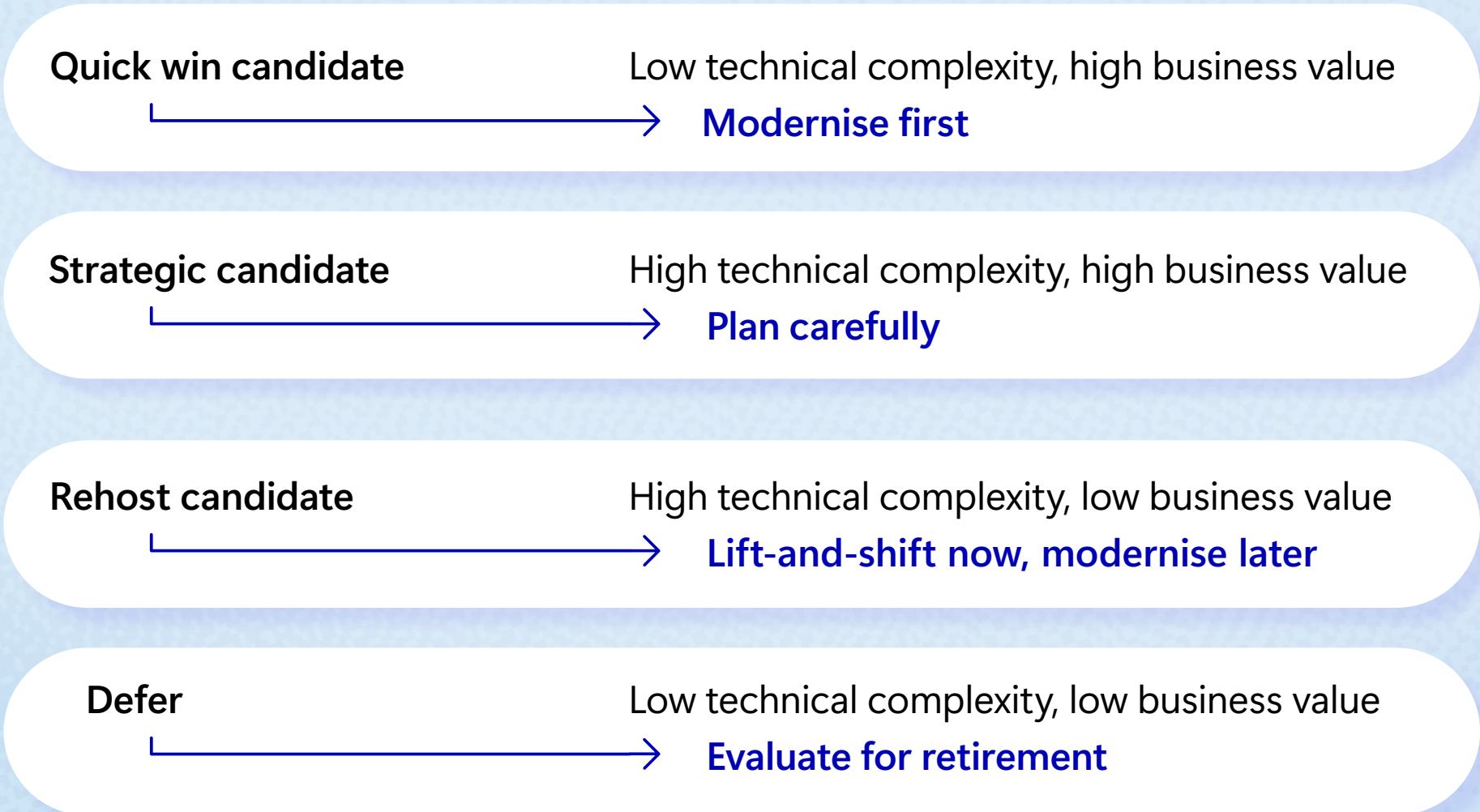
The migration agent extends GitHub Copilot modernisation beyond single applications. It coordinates modernisation across portfolios while evaluating each application according to its architecture, dependencies and business context.

Across your application portfolio, you can use it to:

- Assess readiness across multiple applications simultaneously.
- Generate unique, application-specific modernisation plans.
- Automate upgrades for Java and .NET applications.
- Create and update GitHub issues, pull requests and reports for each app.
- Connect application owners, architects and developers across the modernisation journey.

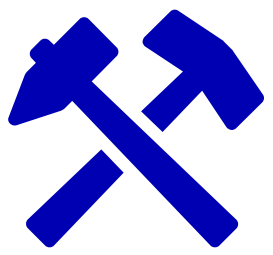
Use a quick prioritisation framework

You can prioritise modernisation candidates using a simple scoring and ranking system based on the criteria that matters most to your organisation. Assessing technical complexity and business value helps you determine the right approach for each application.



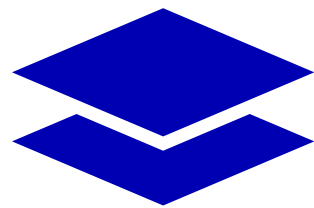
Select the right target architecture

After you've identified and prioritised modernisation candidates, the next step is to determine the optimal compute architecture. Most decisions fall into three broad categories.



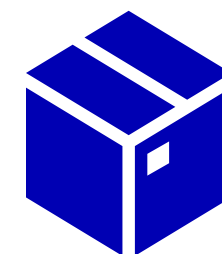
IaaS

A lift-and-shift can quickly reduce data centre costs and simplify migration. It largely replicates your existing environment with minimal code changes. It's a good fit when you need a quick data centre exit or end-of-support replacement.



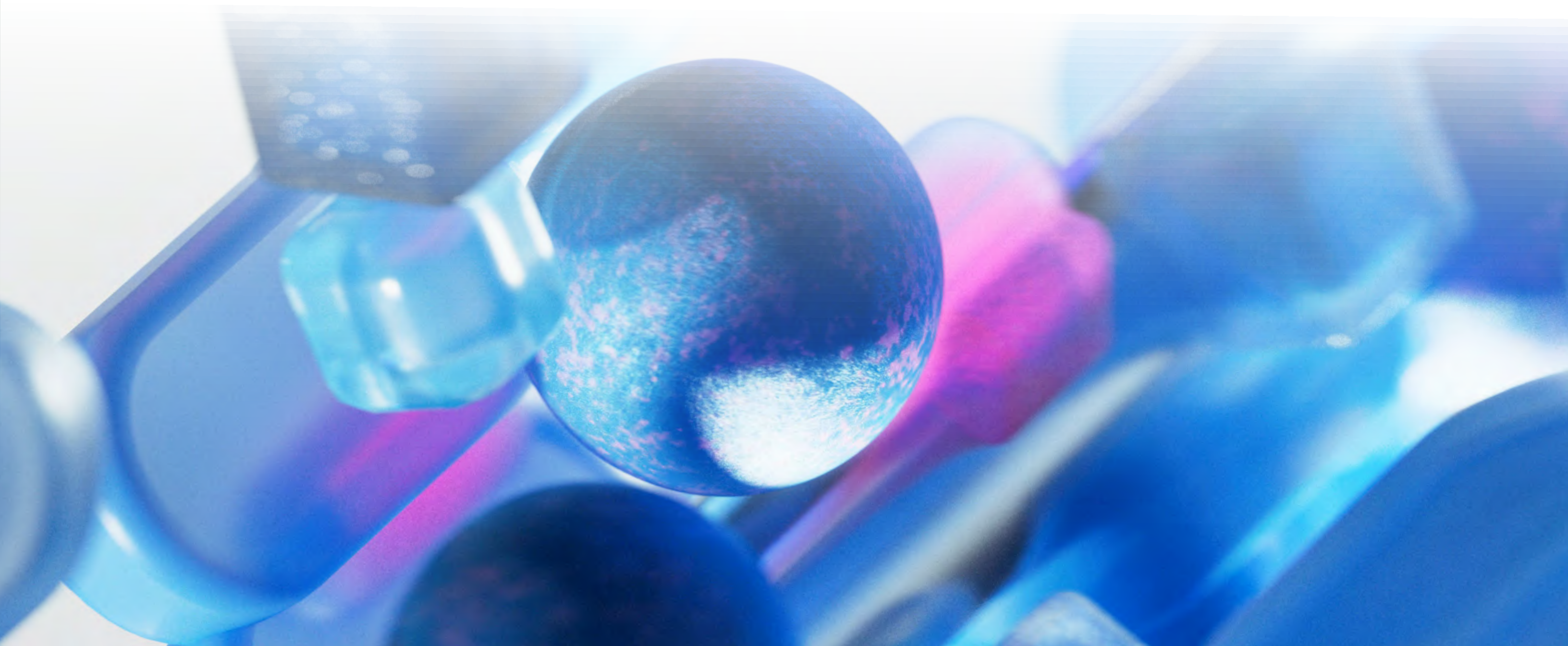
PaaS

When the goal is to unlock cloud-native capabilities, you can adopt platform services, redesigning components to improve agility and long-term scalability. Managed services abstract infrastructure management, enabling teams to concentrate on application logic and innovation.



Containers

For workloads that require portability, scalability, consistency and operational agility, containers are a natural fit. Running containers on managed platforms combines the benefits of cloud-native design with minimal code changes – ideal when you want speed without a full rewrite. Mature ecosystems for .NET, Java and Linux make containerisation a straightforward solution.



TARGET → PaaS + containers

Containers

Containers have become a common way to package and run applications for modernisation, because they deliver consistent, portable artefacts across environments and can accelerate cloud adoption without forcing a full rewrite. By packaging applications and their dependencies into containerised images, you gain portability and predictable behaviour across platforms.

Teams commonly use Docker tooling to build images and run them with production runtimes. Beyond portability, containers simplify modern delivery practices such as immutable deployments, declarative rollouts (rolling, blue-green, canary) via orchestrators and integration with CI/CD pipelines. For web applications and APIs that don't require container orchestration, Azure App Service provides a simpler managed deployment model using code or containers without cluster management.

Example use cases

Monolithic application

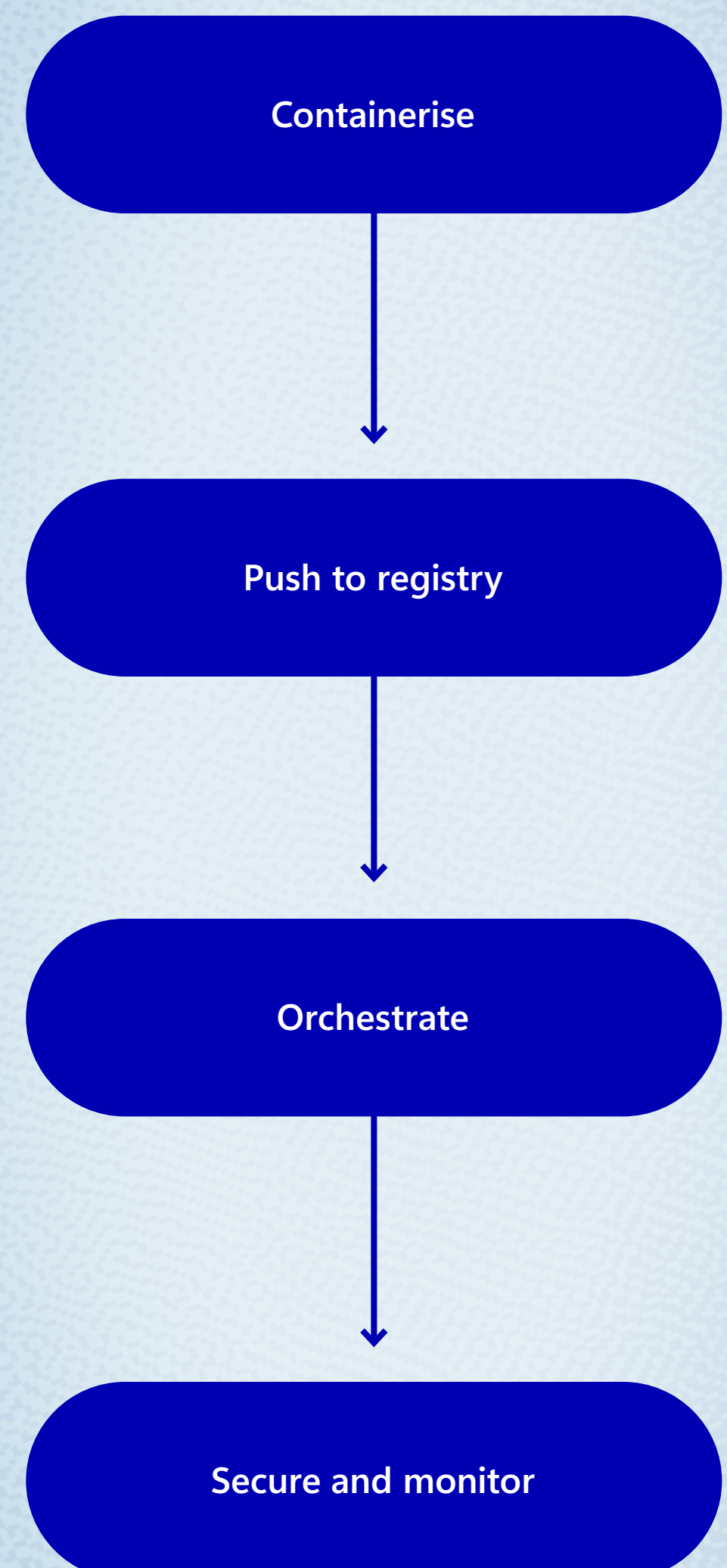
For portability, either replatform directly to a managed PaaS (such as Azure App Service) or containerise as a first step before refactoring.

Web services

Use containers or platform-managed PaaS to scale web services and host them in a managed execution environment.

Batch processes

Use containers for short-lived jobs or parallel workloads and run in a managed container execution environment to provide isolation and portability for scheduled or on-demand tasks.



TARGET → PaaS + simplicity

API-first

The API-first approach ensures that services are designed for interoperability from the start, reducing coupling and accelerating innovation across channels and platforms. By prioritising API contracts before implementation, you can wrap legacy systems with an API façade, unify access patterns and bridge synchronous and asynchronous communication. This approach simplifies integration, supports omnichannel experiences and prepares your architecture for event-driven workflows, microservices and emerging technologies.

Example use cases

Monolithic web app

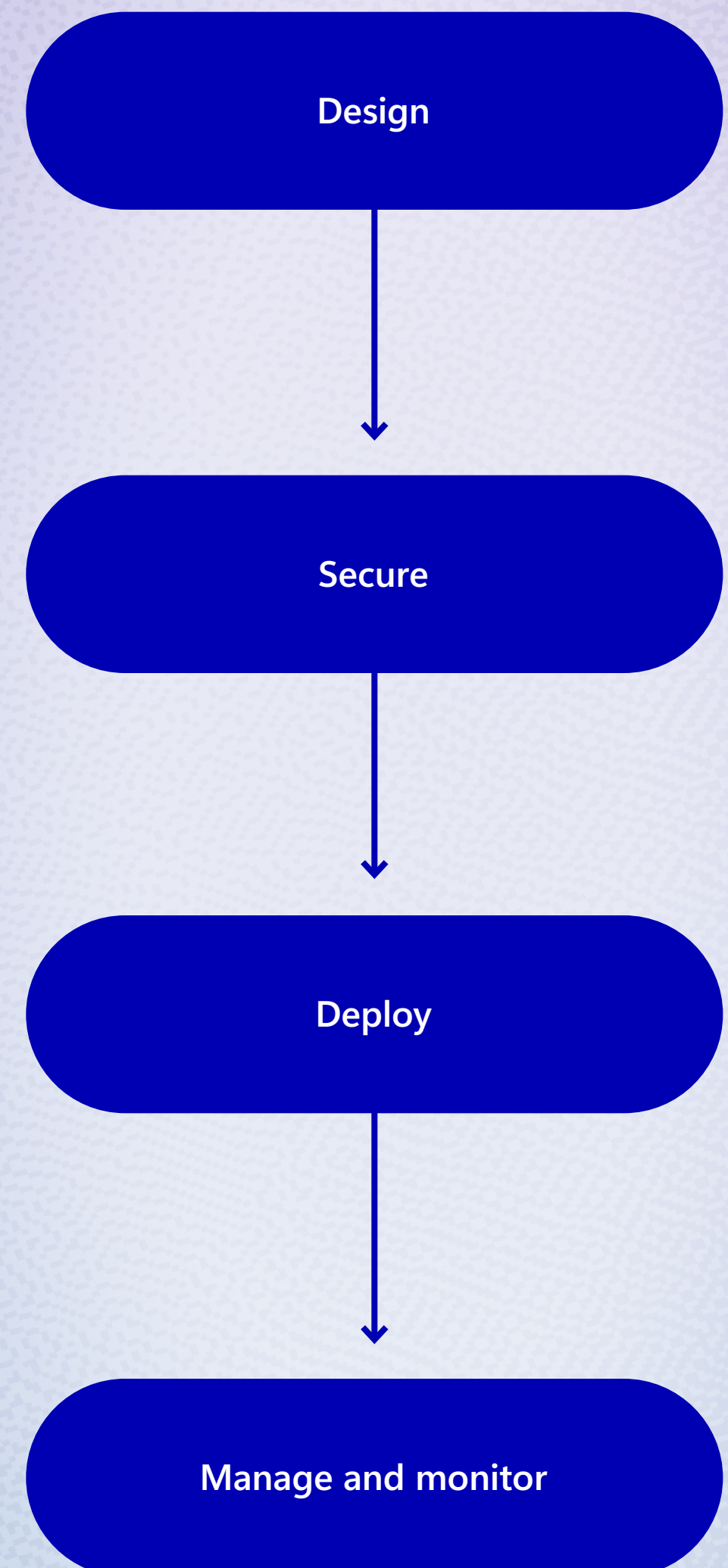
Expose core business functions as APIs to support mobile apps and partner integrations. Define RESTful APIs and wrap legacy logic with an API façade hosted on a managed compute service.

Java application with SOAP services

Replace SOAP endpoints with RESTful HTTP APIs or gRPC services, defined up front.

Custom app with batch processing

Design APIs for data ingestion and retrieval before refactoring the batch logic to support real-time data exchange and integration with other platforms.



TARGET → PaaS + redesign

Event-driven

Event-driven architectures promote loose coupling and scalability by allowing services to react asynchronously, making them well suited for real-time or high-throughput workloads. They can be implemented using serverless compute, where the platform manages scaling and execution or combined with containerisation, where you control runtime configuration, scaling policies and deployment environments.

- If your system is already highly decoupled, serverless may be the simplest choice.
- When modernising legacy applications, containerisation enhances agility and resilience by allowing microservices to scale independently and respond dynamically to events across distributed environments.

Example use cases

Monolithic with heavy data processing

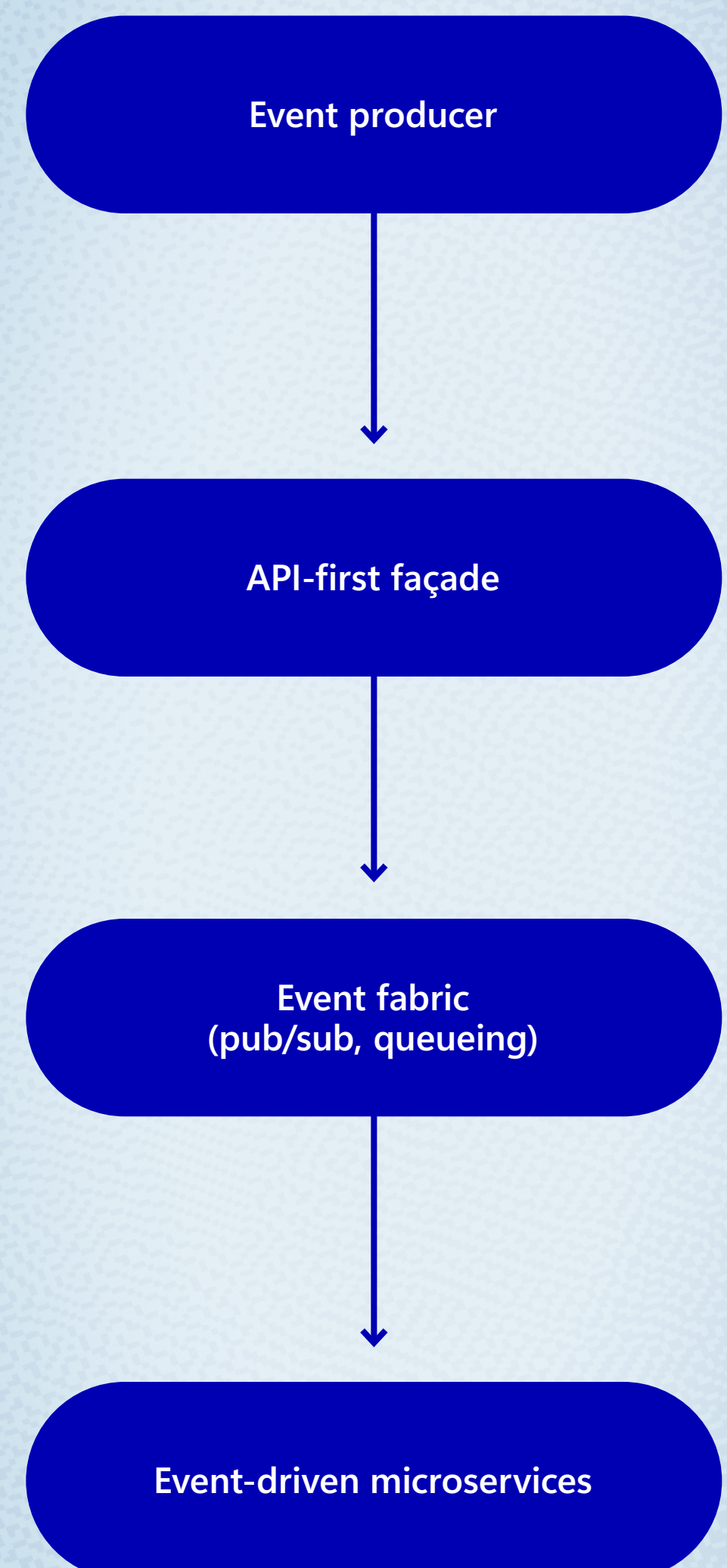
Decouple large data flows or asynchronous tasks like retail inventory updates using event-driven patterns for scalability.

Legacy batch-oriented systems

Modernise data warehouses, ETL pipelines or IoT systems using event-driven pipelines for streaming ingestion and analytics.

Agentic AI workflows

Decompose tasks using microservices to emit events, then use agents to react and orchestrate downstream actions.



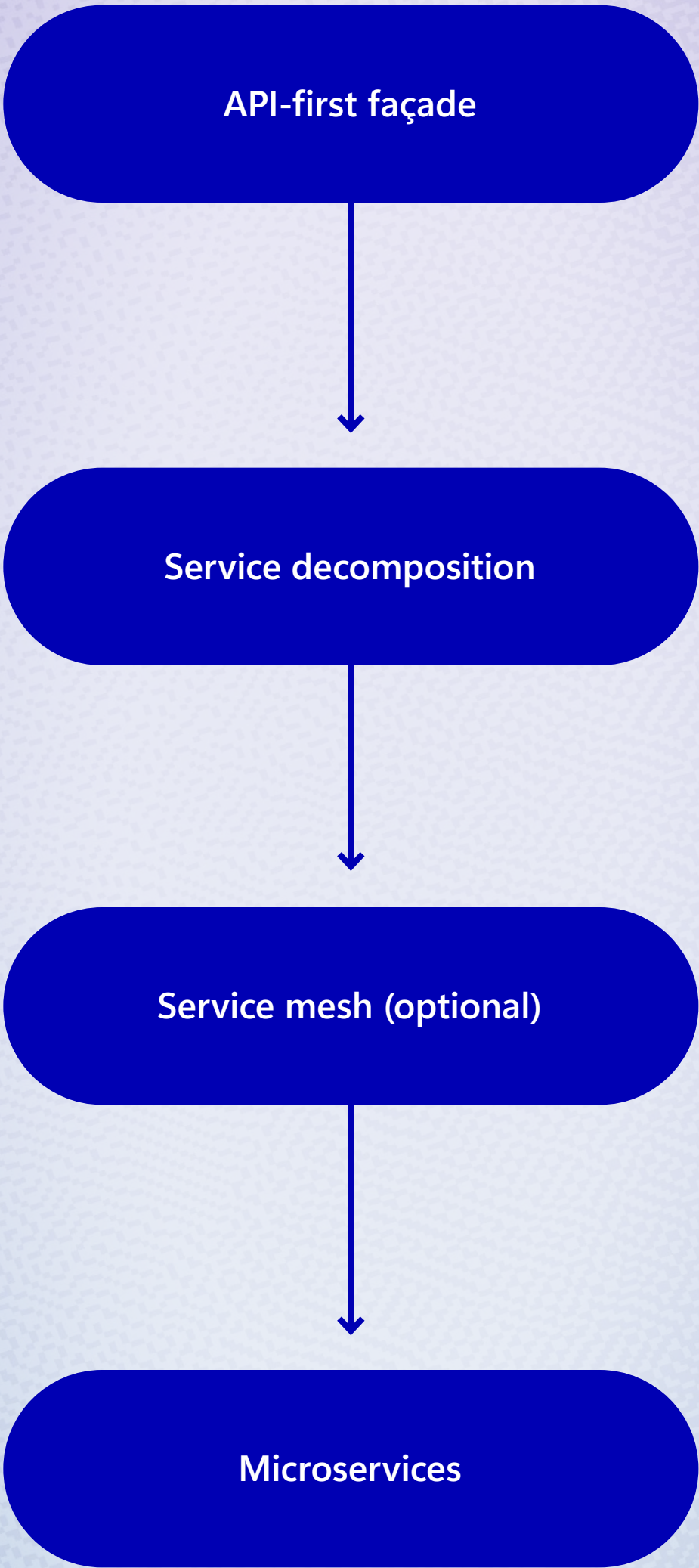
TARGET → PaaS + redesign

Microservices

As a strategic, higher-effort shift, microservices improve reliability and performance by isolating failures and enabling targeted scaling – both critical for modern business agility. By applying domain-driven decomposition, you can break monolithic architectures into independently deployable and scalable services. This approach also allows teams to choose the most suitable programming languages, frameworks, databases or runtime environments for each service.

Example use cases

- E-commerce platform**
Break up tightly coupled layers into services such as Catalogue, Orders, Payments and Inventory. Scale and deploy each service without impacting others.
- Mission-critical systems**
Decompose a core business platform into microservices; for example, Policy Management, Claims, Billing and Customer Onboarding.
- Agentic AI workflows**
Use microservices with an API-first design to make it easy for agents to call discrete functions safely.



Plan your data strategy

When selecting an architecture pattern, it's important to consider how your system will access, integrate and secure data. Each pattern brings its own strengths and trade-offs in performance, flexibility and scalability. A unified data foundation ensures that applications, analytics and AI all work from consistent, high-quality data rather than fragmented silos.

The right database for each pattern depends on your workload's performance, consistency and integration requirements. Cloud-managed databases remove the operational overhead of patching, backups and scaling while adding native capabilities like vector search and AI integrations that are essential for modern intelligent applications.

Azure Databases: Deliver AI apps and agents faster, with built-in capabilities for generative AI and LLM workloads

Azure offers fully managed services for SQL, NoSQL, PostgreSQL and MySQL, providing the foundation for any size business-critical workload your organisation uses to achieve AI transformation and business innovation.

With Azure Databases, companies get up to five times faster performance, intelligent threat detection and up to 58% lower total cost of ownership. New apps, AI agents and data-driven solutions benefit from GitHub Copilot-assisted development, zero-touch recovery, single-digit millisecond latency, advanced vector search capabilities and integrations with the most loved developer tools, analytics platforms and AI technologies.

[Learn more](#)

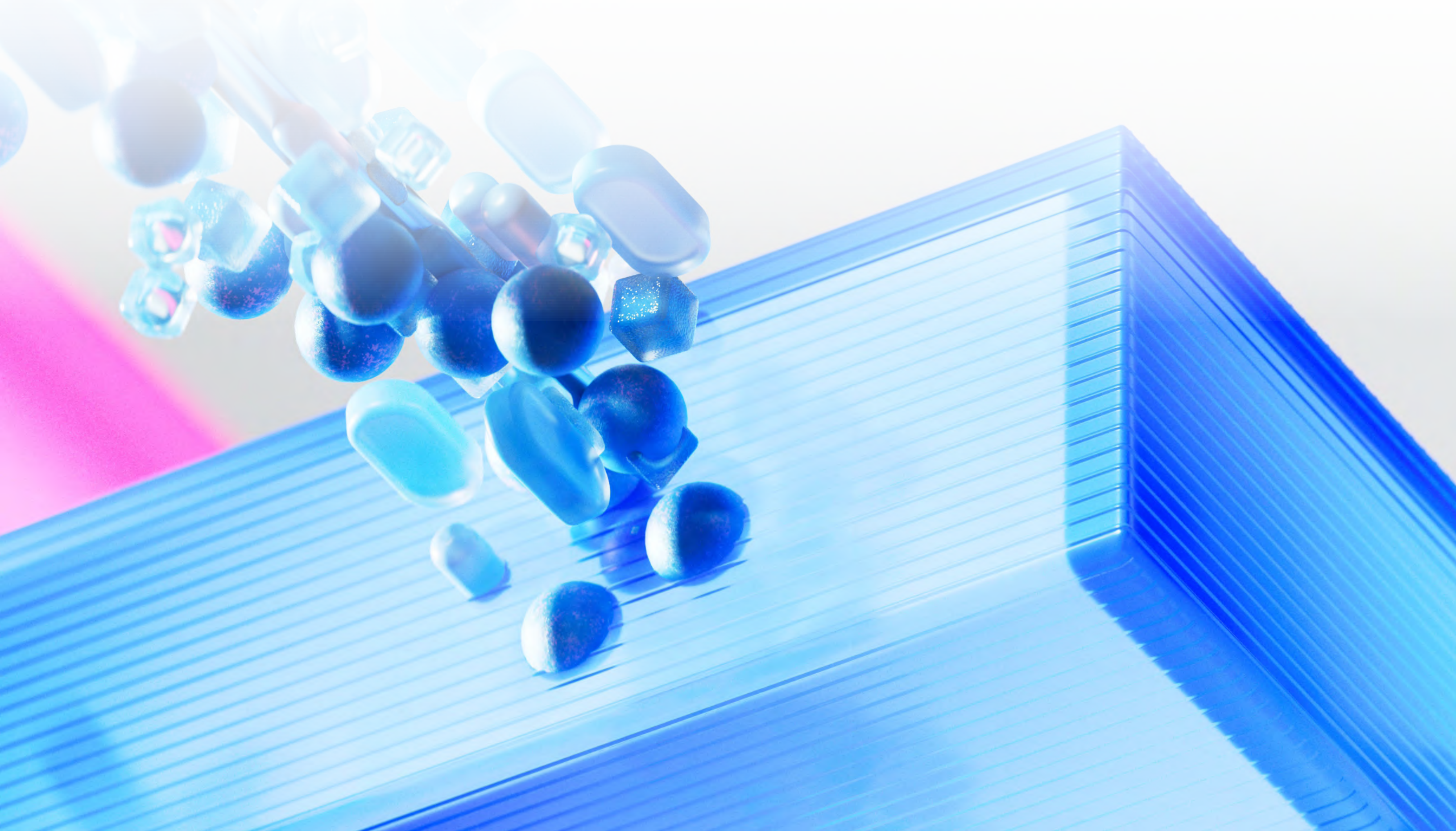


Choose a modernisation pattern

There are several versions of cloud rationalisation (the 'R's), but for application modernisation, these six 'R's most often apply:

- **Rehost** lift-and-shift to VMs.
- **Replatform** move to managed services (PaaS) without major code changes.
- **Refactor** optimise for cloud services, revising the code without a complete redesign.
- **Rebuild** rewrite the app from scratch using cloud-native capabilities.
- **Retire** decommission.
- **Retain** keep as-is for now.

All six 'R's offer valuable paths, but rehosting, replatforming and refactoring are the most practical and commonly adopted strategies for modernising existing applications. These approaches strike a balance between effort and impact, making them ideal starting points when you want to modernise incrementally while delivering real business value.

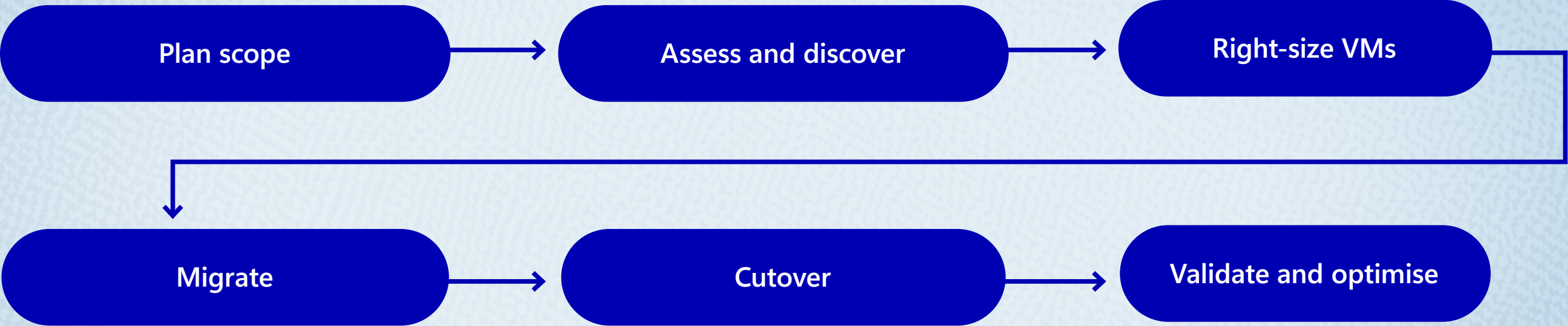
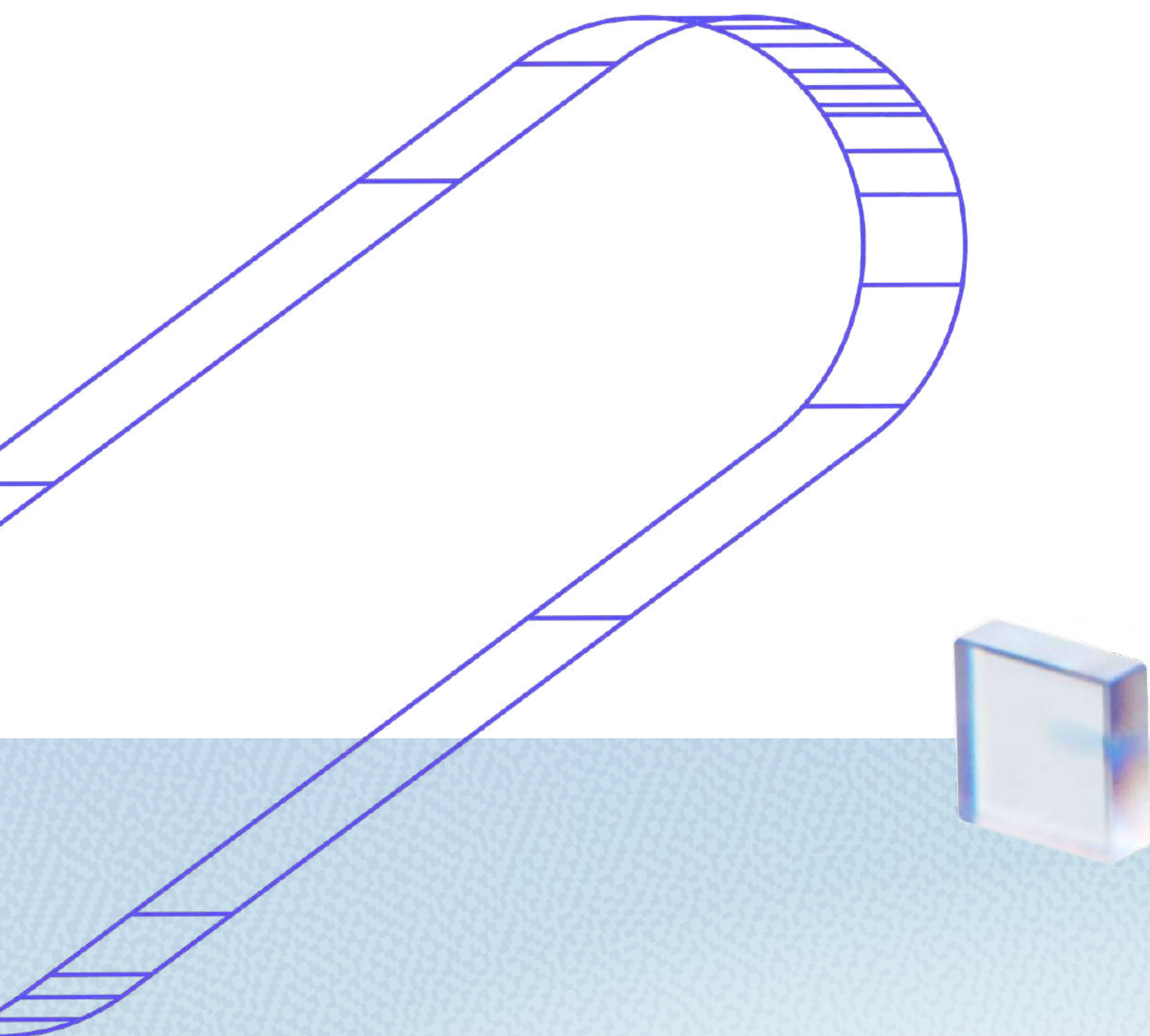


Rehost

Rehosting is often used to get started – lifting existing workloads to matching IaaS without changing the application. Your app won't be more modern yet, but rehosting simplifies later steps like replatforming, because the infrastructure is already in the cloud. Migration is typically fast, thanks to AI-powered automation tools, and delivers a significantly better price-to-performance ratio compared to deploying on-premises.

Quick checklist: Validation after rehosting

- **Application functionality:** Confirm that your app runs as expected after migration, including core features, workflows and integration points.
- **Performance baseline:** Compare post-migration performance against pre-migration benchmarks to ensure acceptable response times and throughput.
- **Connectivity:** Verify network access to dependent services, databases and external endpoints, including DNS and firewall rules.
- **Security checks:** Apply security baselines for VMs, enable Microsoft Defender for Cloud and confirm patching and OS hardening.
- **Observability set-up:** Configure Azure Monitor and Log Analytics for VM health and performance and enable alerts for critical metrics.



Replatform

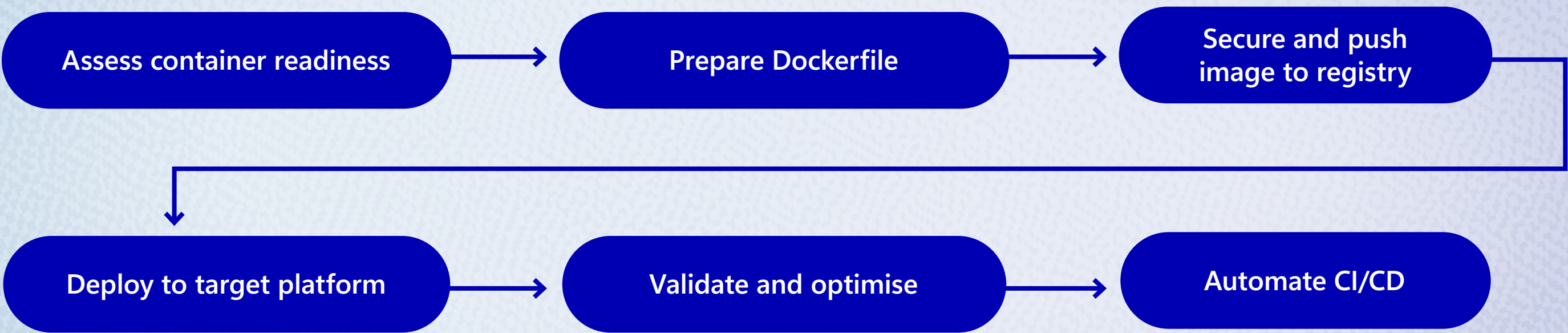
Replatforming moves your application into containers and runs it on managed Azure services. This approach shifts responsibility for infrastructure provisioning, patching and scaling to the platform.

For teams, replatforming reduces manual infrastructure work and supports consistent DevOps practices, while preserving most of the existing application structure. For the business, it lowers total cost of ownership by consolidating operations and improving resource efficiency, while preparing the application for future change. It also aligns with key modernisation patterns: containers for portability, event-driven designs for responsiveness, microservices for scalability and API-first design for integration.

Quick checklist: Validation after replatforming

- **Functional testing:** Ensure the application behaves correctly in the target managed environment (containerised or platform-managed).
- **Performance comparison:** Benchmark containerised performance against the original deployment to confirm acceptable response times and resource usage.
- **Integration testing with the data layer:** Verify connectivity and operations with databases, APIs and other dependent services after containerisation.
- **Security checks:** Validate platform security controls, vulnerability posture and compliance with baseline security policies (for example, image scanning for containers or Defender for Cloud integration).
- **Observability set-up:** Enable logging, metrics and health probes to proactively monitor container performance and ease troubleshooting.

If containerisation is part of your replatform strategy:



If not, you can replatform applications directly to Azure App Service, simplifying these steps to configuration, deployment and CI/CD set-up. No Dockerfiles or image management needed.

Refactor

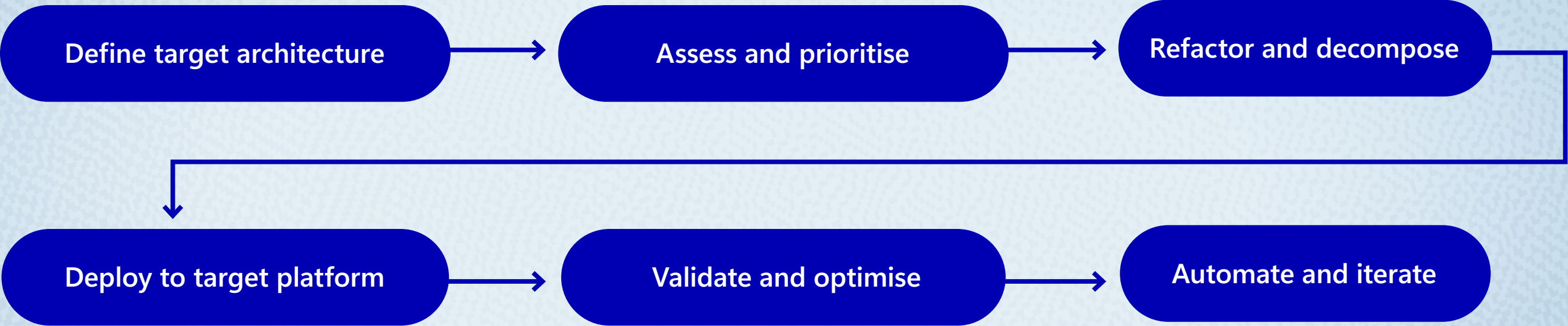
Refactoring updates your application code to reduce technical debt, improve maintainability and boost performance. It often involves redesigning parts to better fit cloud patterns and SDKs, which is why it's sometimes called rearchitecting. Refactoring is the path out of technical debt and toward agility, enabling you to execute business changes up to 78% faster.⁸

For example, breaking apart tightly coupled applications into modular components allows teams to scale and update parts of the system independently. This approach supports API-first and domain-driven designs, as well as microservices and event-driven architectures.

A practical decomposition strategy starts by identifying clear service boundaries, extracting high-value components first and maintaining backward compatibility during the transition. Incremental rollout helps reduce risk while delivering value earlier.

Quick checklist: Validation after refactoring

- **Functional integrity:** Verify that each refactored service works as intended and that overall application workflows remain consistent.
- **Performance benchmarks:** Compare service-level and end-to-end performance against pre-refactor baselines.
- **Integration testing:** Confirm that new services communicate correctly with existing components, APIs and data sources.
- **Security posture:** Validate that each service meets security requirements, including identity, secrets management and compliance checks.
- **Observability readiness:** Ensure logging, metrics and distributed tracing are in place for all services.
- **Backward compatibility:** Test that legacy consumers and interfaces still function during phased rollout.



Azure services at a glance

Whether you prioritise API-first integration, microservices scalability, container portability or proven platforms, you can align your choice with specific Azure services and business outcomes.

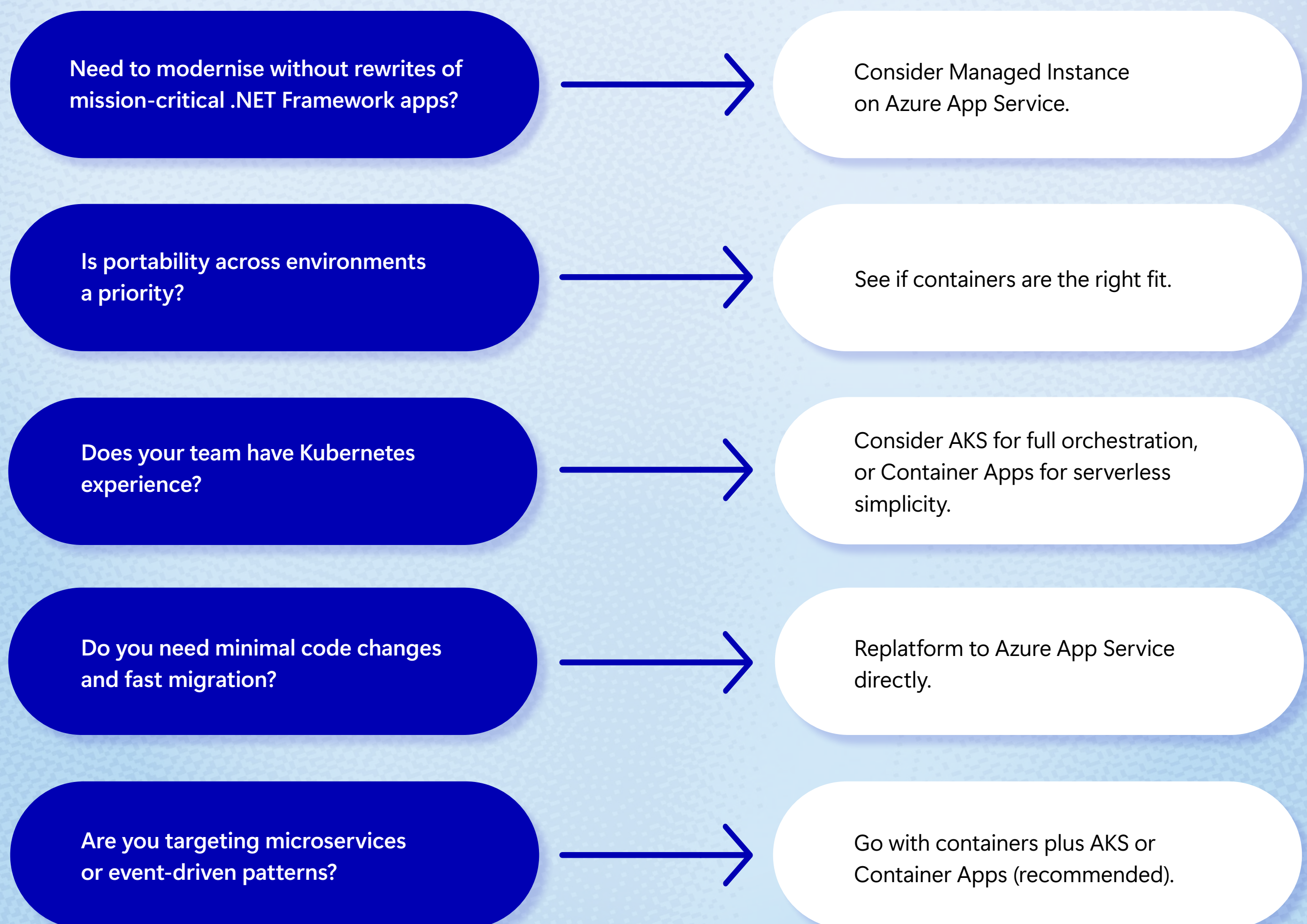
Azure Kubernetes Service (AKS)	Fully managed Kubernetes for containerised workloads at scale. Best for microservices and complex orchestration needs.
Azure App Service	Managed platform for web apps and APIs. Simplest path for replatforming without container orchestration
Azure Container Apps	Serverless containers for event-driven and microservices workloads. No cluster management required.
Azure API Management	Centralised API gateway for publishing, securing and monitoring APIs across environments.
Azure Service Bus / Azure Event Hubs	Message and event streaming services for decoupled, event-driven architectures.
GitHub Copilot modernisation	AI-powered agent for end-to-end code transformation, upgrade planning and Azure deployment from within the IDE.
Microsoft Defender for Cloud	Unified cloud-native application protection across development and runtime.



Modernise with containers: A decision framework

Containers are a powerful enabler of modernisation, but the best approach depends on your workload's characteristics and your business goals. While containerisation often delivers portability and operational efficiency, some scenarios may benefit more from replatforming or refactoring first.

This decision framework helps you quickly assess whether containers are the right fit for your application and guides you toward the most effective modernisation path.



Best practices and built-in guardrails

Azure includes guardrails that make it easier and safer to implement containers. To avoid common pitfalls before they happen:

Enforce non-root and approved base images in your Dockerfile.

Apply Azure Policy for AKS to enforce non-root containers. For runtime security and compliance checks, use Microsoft Defender for Containers.

Don't hardcode configurations.

Externalise configuration to reduce drift and exposure. Use platform-native mechanisms for environment variables and secure stores like Azure Key Vault for sensitive values.

Use multi-stage Docker builds.

Minimise image size and speed up deployments, and store optimised images in Azure Container Registry for versioning and vulnerability scanning.

Implement liveness and readiness probes.

Monitor container health and enable automatic recovery. Use built-in probes in AKS or health checks in Azure App Service, with Azure Monitor for alerts and visibility.

Define CPU and memory requests and limits.

Set boundaries so pods don't exhaust node resources. In AKS, these limits support predictable scheduling and scaling, and Azure Monitor helps track utilisation.

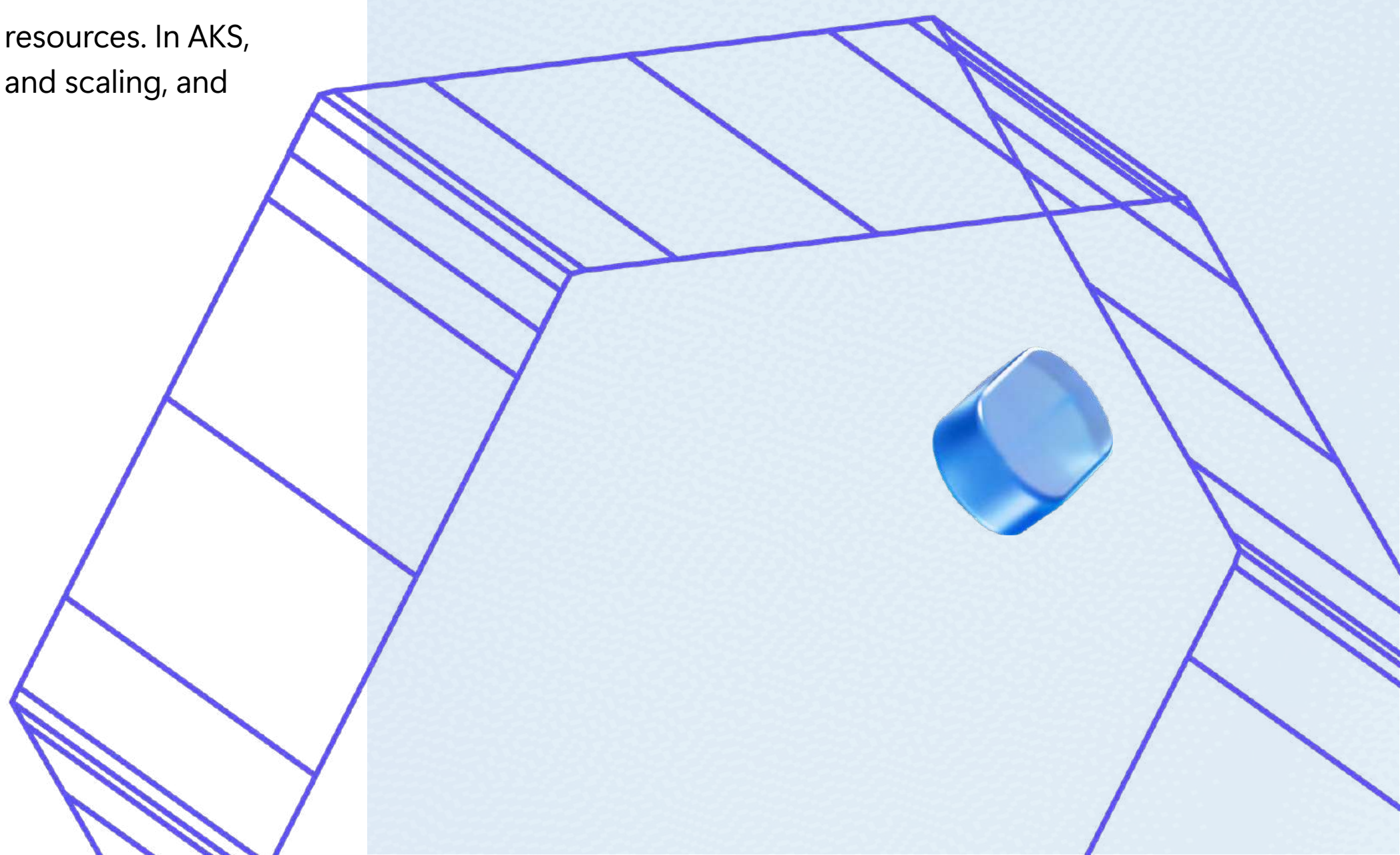
Microsoft and digital sovereignty

Microsoft Sovereign Cloud: The most comprehensive set of sovereign solutions across AI, productivity, security and cloud platform.

Our approach is simple: digital sovereignty is foundational to our cloud and services, delivered through a well-governed combination of technical, contractual and operational controls. Because every workload and every organisation has different levels of sensitivity and criticality, sovereignty requirements are adaptable to the needs of each customer.

The Microsoft Sovereign Cloud brings together AI, productivity, security and cloud workloads to span public, hybrid and disconnected environments, so customers can apply the right level of control, capability and compliance based on the sensitivity and criticality of each workload, rather than relying on a single deployment model.

[Learn more](#)



Execute and optimise

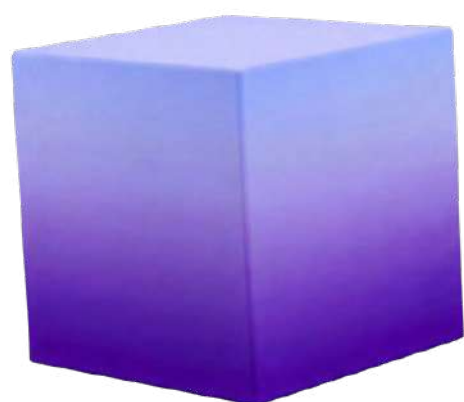
Post-migration success depends on how well your workloads perform, scale and stay resilient. Without optimisation, cloud costs can spiral and performance can lag, undermining the value of modernisation. Strong governance, coupled with a culture of continuous improvement, forms the cornerstone of sustained operational excellence.



Core modernisation practices

Certain practices help protect your workloads, accelerate delivery and ensure business continuity across every modernisation path.

- Adopt CI/CD from day one. Automated pipelines reduce deployment risk and accelerate delivery cycles, giving your teams more time for innovation.
- Shift security left. Integrate security scanning and compliance checks into the development pipeline so issues are caught before they reach production.
- Establish observability standards. Logging, metrics and distributed tracing across all services give you the visibility you need to detect issues proactively and maintain SLAs.
- Govern with policy as code. Automate compliance and configuration standards so guardrails are enforced consistently.
- Enable incremental rollout. Blue-green and canary deployment strategies let you release updates more safely with rollback capability if issues arise.



Optimise costs

There are several practical steps you can take to reduce cloud costs while maintaining performance and scalability.

- Right-size resources. Use monitoring data to identify over-provisioned VMs, containers and managed services and adjust to match actual workload demand.
- Apply reserved capacity and hybrid benefits. For predictable workloads, reserved pricing and Azure Hybrid Benefit can significantly reduce compute costs.
- Use auto-scaling. Configure automatic scaling policies so you pay for capacity when you need it, not when you don't.
- Tag and track spend. Implement consistent resource tagging for cost allocation, and use cost management tools to monitor spend by team, workload or project.

Drive continuous improvement

By regularly measuring performance, gathering insights from teams and aligning operations with business goals, you can refine your environments to support cloud-native scalability and prepare for AI-driven innovation.

Modernisation succeeds when it's deliberate. When you use proven patterns, validate each step and choose the right services for your workloads, you can build sustainable, future-ready systems that evolve with your business.



Next steps

Modernisation is the foundation for AI transformation. The patterns in this guide provide building blocks that help teams move faster, trust their results and scale systems reliably.

Speed up cloud and AI adoption with expert support and funding

Get end-to-end guidance with [Frontier Accelerate](#)

Modernise faster with a modern AI-ready platform

Explore tutorials at [AI Skills Navigator](#)

Design and operate AI-ready infrastructure

Follow a guided path and take the [Microsoft Learn module](#)

¹ Phenix, Eric. [GigaOm Benchmark: Migrating Windows Server Workloads to Azure](#). GigaOm. 10 March 2025.

² [Microsoft's Agentic Tools Reshape Application Modernisation for the AI Era](#). LavX News. 13 February 2026.

³ [Modernise your applications in days, not months](#). GitHub App Modernisation website.

⁴ [Accelerate migration and modernisation with agentic AI](#). Azure Blog. 25 September 2025.

⁵ [Modernise your applications in days, not months](#). GitHub App Modernisation website.

⁶ [Modernise your applications in days, not months](#). GitHub App Modernisation website.

⁷ [Microsoft's Agentic Tools Reshape Application Modernisation for the AI Era](#). LavX News. 13 February 2026.

⁸ [The Business Value of Migrating and Modernising IT Estate with Microsoft Azure](#). IDC. April 2025.