Proto Comet

A PRACTICAL GUIDE FOR TECHNOLOGY LEADERS

# The 6 R's of .NET Modernization

Six proven strategies for moving legacy .NET applications to Azure

What they mean, when to use them, and how real organizations combine them.

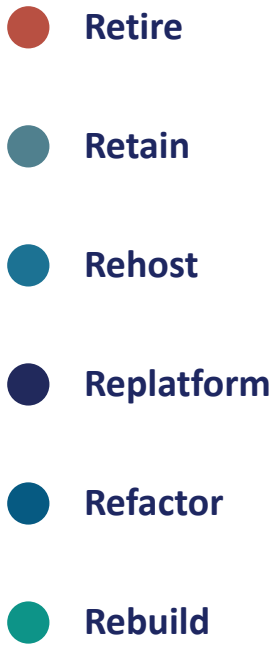
Yogesh Verma | .NET Modernization & Azure Cloud Transformation

INTRODUCTION

# Why the 6 R's matter

Every legacy .NET estate eventually faces the same question: what do we do with this application? The 6 R's framework gives technology leaders a shared vocabulary for answering that question, application by application, instead of treating modernization as one giant all-or-nothing project.

This guide walks through each of the six strategies, grounds each one in a real-world .NET scenario, and then shows how mature organizations blend two or more strategies across a single application portfolio.



1

# Retire

*Decommission the application entirely. The fastest, cheapest modernization move there is.*

STRATEGY 1 OF 6

## Definition & When to Use

### What it means

Retire means switching an application off rather than moving it anywhere. It applies to systems that have outlived their business purpose, duplicate functionality found elsewhere in the estate, or were kept alive purely out of inertia. Every retired app is one less thing to patch, license, secure, and migrate.

### Use it when...

- Usage logs show the app is barely or never used
- Its functionality is duplicated by another system
- It exists only to serve a handful of legacy reports
- No clear business owner can justify keeping it

**TYPICAL AZURE TARGETS**

None required generally, workload is decommissioned and infrastructure de-provisioned.

2

# Retain

*Keep it exactly where it is for now. A deliberate pause, not a rejection of modernization.*

STRATEGY 2 OF 6

## Definition & When to Use

### What it means

Retain means consciously leaving an application on-premises or in its current state, typically because migrating it now is not worth the cost, risk, or disruption. This is a phased-roadmap decision, not a dead end, retained apps are revisited once dependencies, contracts, or hardware cycles change.

### Use it when...

- Regulatory or data-residency rules block a cloud move today
- The app is tied to specialized on-prem hardware
- A planned retirement or replacement is already on the roadmap
- Migration cost clearly outweighs near-term benefit

**TYPICAL AZURE TARGETS**

Hybrid connectivity (ExpressRoute / VPN) to keep retained systems integrated with newly migrated ones.

3

# Rehost

*"Lift and shift". Move the application to the cloud with minimal to no code changes.*

STRATEGY 3 OF 6

## Definition & When to Use

### What it means

Rehost moves an application's existing servers, in roughly their current form, onto cloud infrastructure. It's the fastest way to exit a datacenter and start realizing cloud economics, even though the application architecture itself isn't changed yet. Many organizations use Rehost as Phase 1 of a longer modernization journey.

### Use it when...

- Datacenter contract or hardware refresh deadline is approaching
- Team needs a **quick win** before tackling deeper re-architecture
- The app is stable but the underlying infrastructure is end-of-life
- Business can't tolerate the downtime a re-architecture would need yet

**TYPICAL AZURE TARGETS**

Azure Migrate, Azure Virtual Machines, IIS on Azure VMs.

# 4

# Replatform

*"Lift, tinker, and shift" — move to the cloud while swapping in managed platform services.*

STRATEGY 4 OF 6

## Definition & When to Use

### What it means

Replatform sits between Rehost and Refactor: the application's core architecture stays largely intact, but underlying components are swapped for managed Azure equivalents. Moving a hand-managed SQL Server to Azure SQL Database, for example. It captures meaningful cloud-native benefits without a full rebuild.

### Use it when...

- App is fundamentally sound but the team is tired of managing infrastructure
- Database or middleware is consuming disproportionate operational effort
- There's appetite for moderate change, but not a ground-up rewrite
- Cost or performance gains are needed faster than a refactor allows

**TYPICAL AZURE TARGETS**

Azure App Service, Azure SQL Database, Azure Cache for Redis.

5

# Refactor

*Improve the code without changing what the application does from the outside.*

STRATEGY 5 OF 6

## Definition & When to Use

### What it means

Refactor means changing existing code internally, cleaning up structure, improving how components talk to each other, optimizing data access, without altering the application's external behavior. It sits **below Rebuild** in terms of cost and risk: the business logic and user experience stay familiar, but **performance, scalability, and maintainability improved underneath**.

### Use it when...

- The application's structure is slowing releases, but the logic itself is sound
- There's a clear opportunity to decompose key components without a ground-up rewrite
- The business wants better performance and integration with no visible disruption
- There are resources for meaningful change, but not for a full rebuild

**TYPICAL AZURE TARGETS**

Azure Service Bus, Azure Event Hubs, Azure SQL Database, Azure App Service. However, the major work would be around the code level with this approach.

6

# Rebuild

*Start over. Designing and building the application fresh on cloud-native foundations.*

STRATEGY 6 OF 6

## Definition & When to Use

### What it means

Rebuild is the right call when the cost of replatforming or refactoring an application outweighs the benefit of keeping any of its existing code. Rather than evolving the legacy codebase, the application is redesigned from the ground up, decomposed into microservices, containers, or serverless components, to address deep legacy issues and incorporate functionality the old architecture could never support, such as AI-readiness or true multi-tenancy.

### Use it when...

- The application is highly complex and needs major performance or scale gains
- Legacy code is so tangled that refactoring it would cost more than starting fresh
- The business needs capabilities (AI, real-time, multi-tenant) the old design can't support
- There's organizational readiness for a longer, higher-investment initiative

**TYPICAL AZURE TARGETS**

Azure Kubernetes Service, Azure Container Apps, Azure Functions, Azure Cosmos DB.

IN PRACTICE

# Real migrations blend two or more strategies

No mid-market .NET estate is one single application, it's dozens, sometimes hundreds, each at a different age and level of business importance. Mature modernization programs apply the right R to each application individually, then sequence the work so quick wins fund and de-risk the deeper transformations.

DECISION FRAMEWORK

# How to choose the right R

Start with two questions for every application in the portfolio, asked in this order. The answers narrow the field fast.

**1** **Does this application still need to exist?**

**No → Retire**

Yes → continue

**2** **Can we move or change it right now?**

**No → Retain**

Yes → continue

If it's still in play, the remaining choice is about depth — how much should actually change:

| Rehost | Replatform | Refactor | Rebuild |
|---|---|---|---|
| Need speed, architecture is fine | Want managed services, moderate change | Code needs cleanup, behavior stays the same | Need true cloud-native scale or AI-readiness |

NOT SURE WHICH R FITS YOUR ESTATE?

# Start with an AI Readiness & Legacy Health Check

A focused 1–2 week engagement that scores your .NET estate across security, scalability, observability, DevOps maturity, and AI readiness — then maps each application to the R that fits, backed by a prioritized modernization backlog.

**Book a Discovery Call**

# Yogesh Verma

Independent .NET Modernization & Azure Cloud Transformation Consultant

*Nearly two decades of hands-on Microsoft .NET and Azure experience.*
*From legacy .NET Framework to modern cloud-native architecture.*

**protocomet.com**

Proto Comet