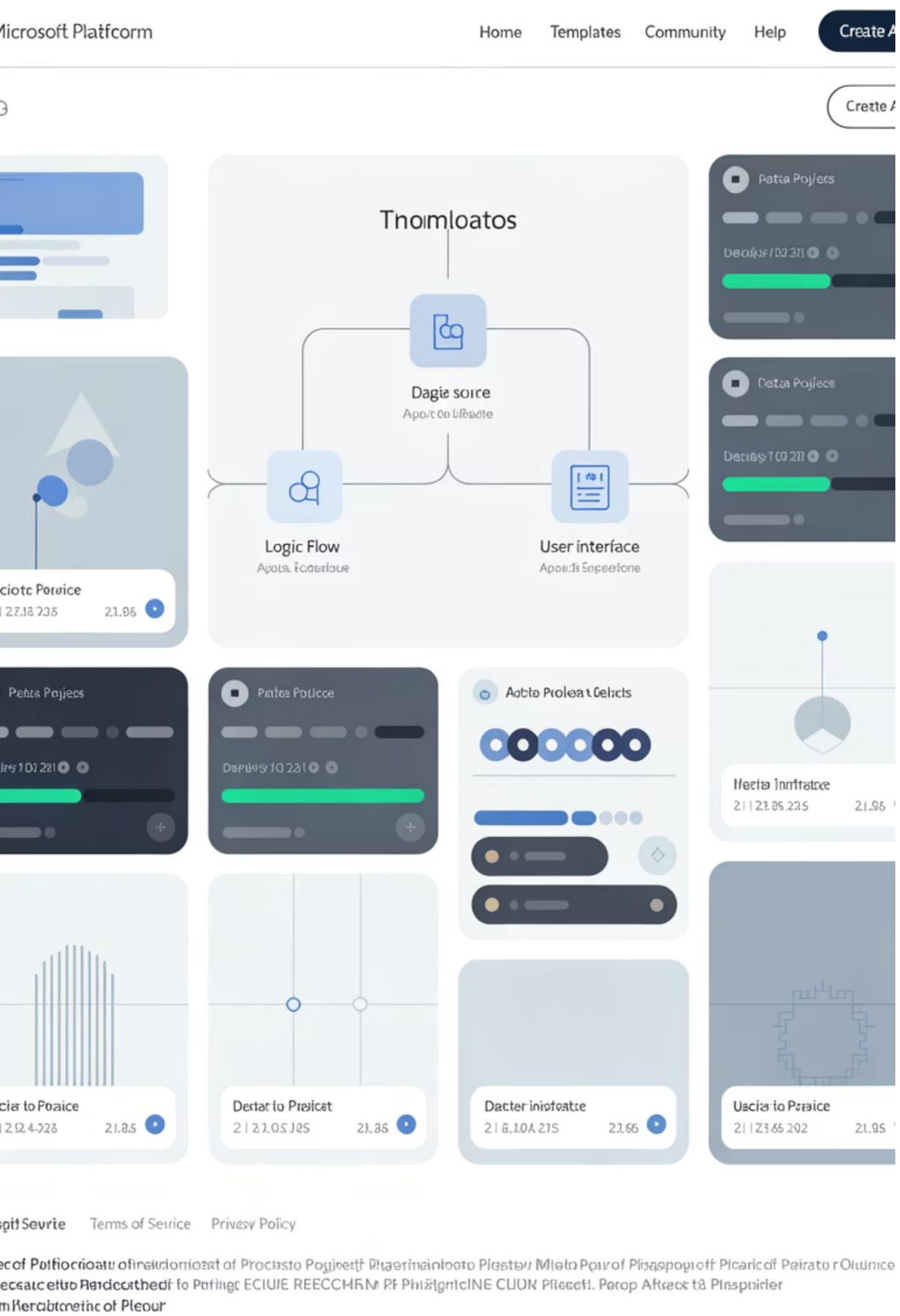


Microsoft Power Platform Application Lifecycle Management

Master the essential practices for managing Power Platform solutions across environments, from development to production deployment.

What We'll Cover Today

01

Solutions Management

Understanding managed vs. unmanaged solutions and when to use each

02

Catalogs & Reusability

Leveraging internal publishing for governance and consistency

03

Solution Layering

Managing versioning and component conflicts effectively

04

ALM Best Practices

Environment strategies and deployment pipelines

05

Automation Strategies

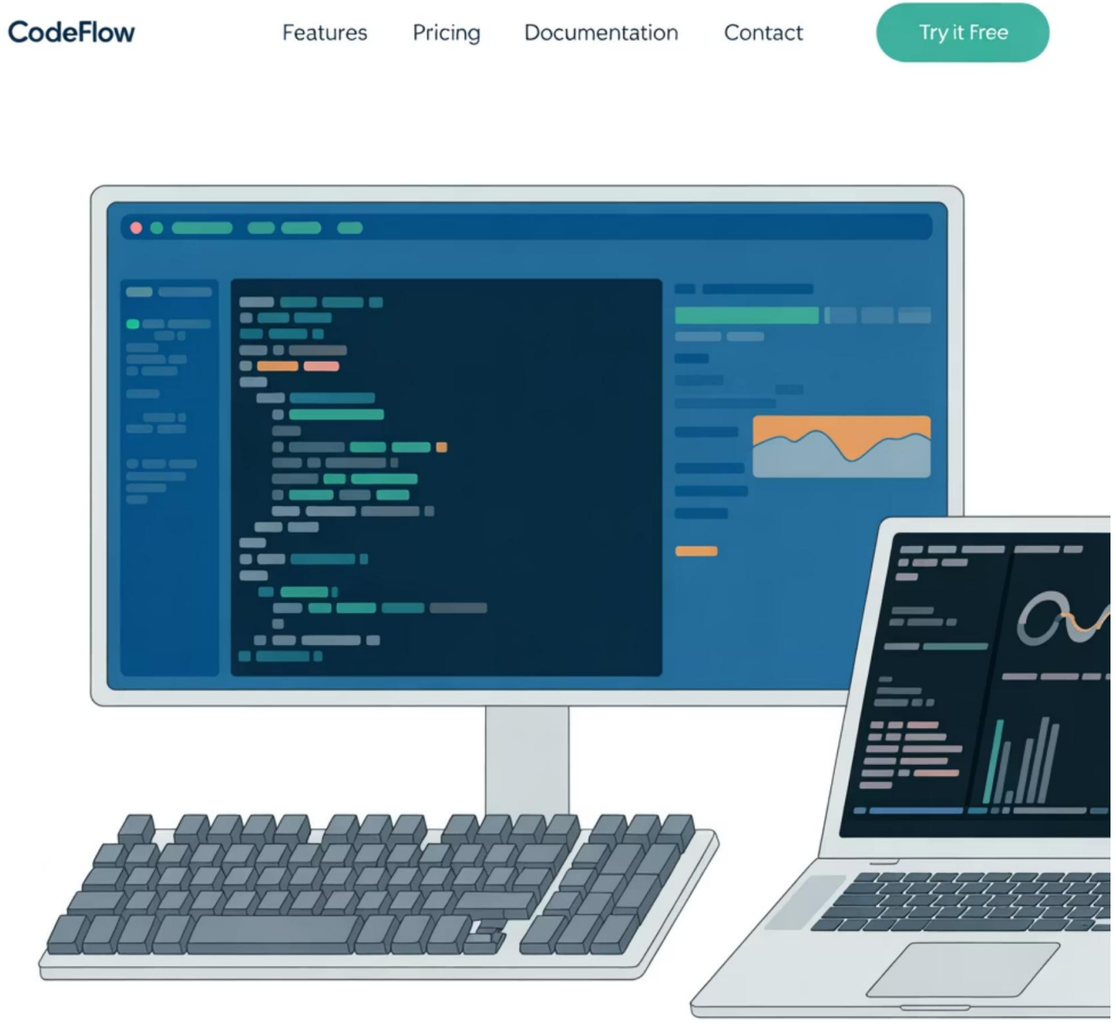
CI/CD workflows and automated solution management

Solutions Management

Understanding the foundation of Power Platform ALM

The Two Types of Solutions

Unmanaged Solutions



Managed Solutions

The One-Way Street

1	2
Unmanaged Solution	Managed Solution
Development environment	Production environment
Full editing capabilities	Protected and versioned

⚠ **Critical:** You cannot convert a managed solution back to unmanaged. This is a permanent, one-way transformation that protects production integrity.

Development to Production Workflow

Develop

Create and modify components in unmanaged solutions

1

2

Test

Export as managed solution for testing environment

3

Deploy

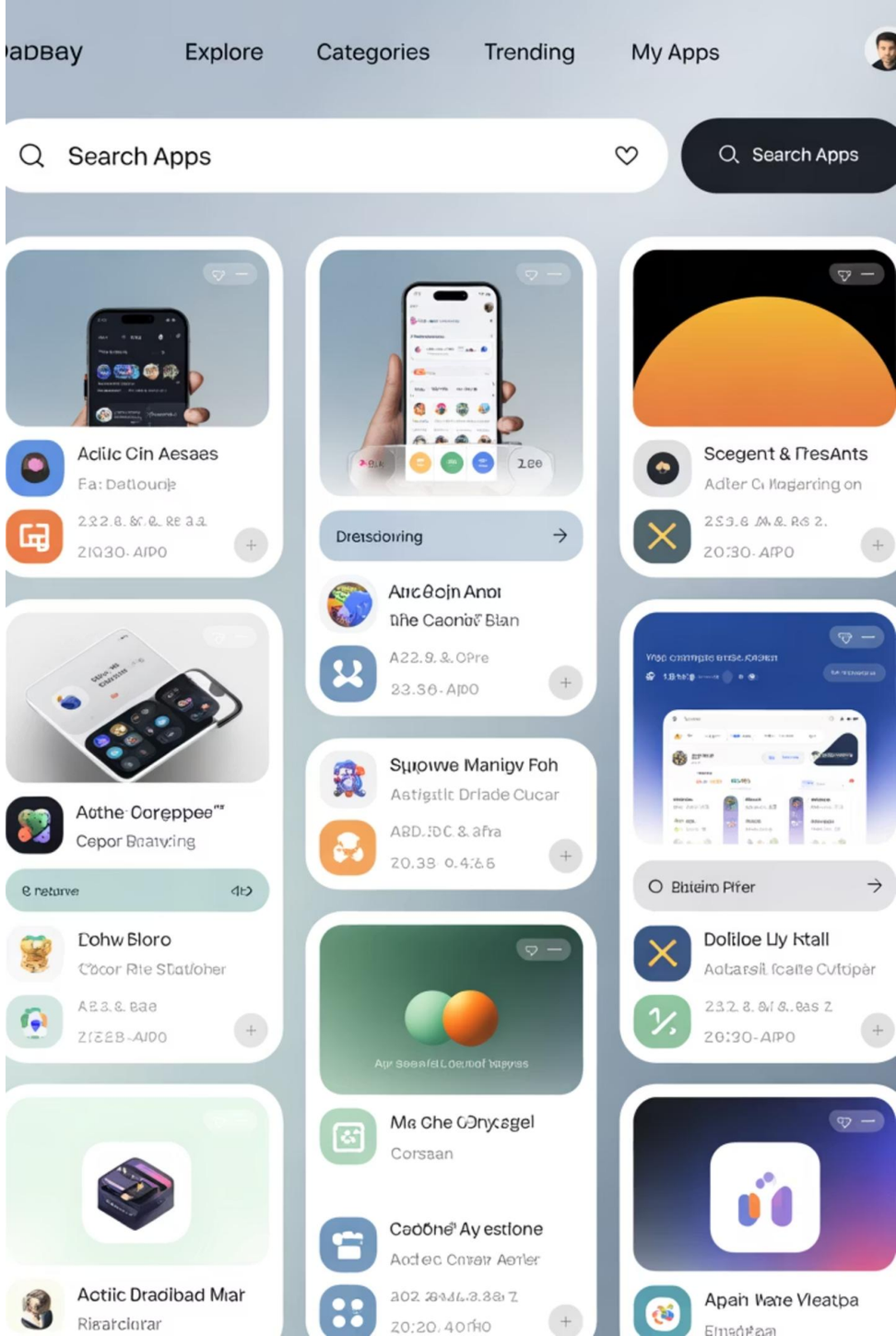
Import managed solution to production environment

This workflow ensures clean separation between development flexibility and production stability, reducing risk while maintaining governance.

Catalogs: Your Internal App Store

Power Platform catalogs function as an internal marketplace where teams can publish and discover reusable solutions and components. This centralized approach transforms how organizations manage their Power Platform assets.

Think of catalogs as your company's private app store, where approved connectors, components, flows, and complete applications are made available for other makers to discover and implement.



Why Catalogs Matter

Governance

Centralized approval and publication process

Ensures compliance with organizational standards

Reusability

Eliminates duplicate development efforts

Accelerates project timelines through proven components

Consistency

Standardized user experiences across applications

Unified branding and functionality patterns

Catalog Benefits for Large Organizations

For Makers

- Quick access to pre-approved components
- Reduced development time
- Built-in best practices
- Consistent user experiences

For IT Leaders

- Controlled component distribution
- Reduced shadow IT risks
- Standardized architecture patterns
- Simplified maintenance and updates

Solution Layering

Managing complexity through strategic component organization

Understanding Solution Layers

1

Unmanaged Layer

Top priority - overrides everything below

2

Patch Solutions

Incremental updates to managed solutions

3

Managed Solutions

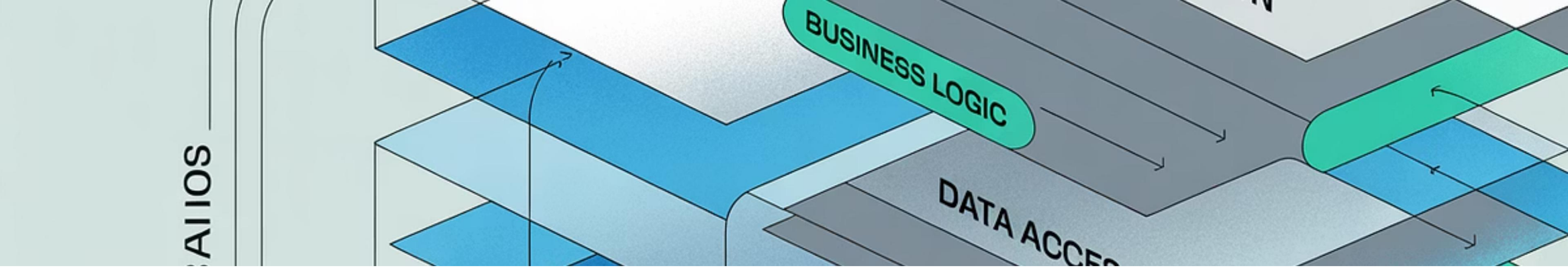
Production deployments with version control

4

System Layer

Base Dataverse functionality

The system merges these layers in a specific order, with the topmost layer determining the final behavior. This layering system is crucial when multiple solutions modify the same component.



The "Top Layer Wins" Principle

When multiple solutions modify the same component, Dataverse follows a clear hierarchy. The topmost layer always takes precedence, determining the final component behavior.

> Understanding this principle is essential for managing complex environments where multiple teams deploy overlapping solutions.

This approach ensures predictable behavior while allowing for strategic overrides when necessary.

Versioning Strategy

1	2	3
Major Versions Breaking changes or significant new features Example: 2.0.0 → 3.0.0	Minor Versions New features with backward compatibility Example: 2.1.0 → 2.2.0	Patch Versions Bug fixes and small improvements Example: 2.1.1 → 2.1.2

Solution Lifecycle Operations

Available Operations

- **Upgrade:** Replace existing solution with newer version
- **Patch:** Apply targeted fixes without full replacement
- **Rollback:** Return to previous stable version

Each operation serves different scenarios and has specific implications for component management and user experience.

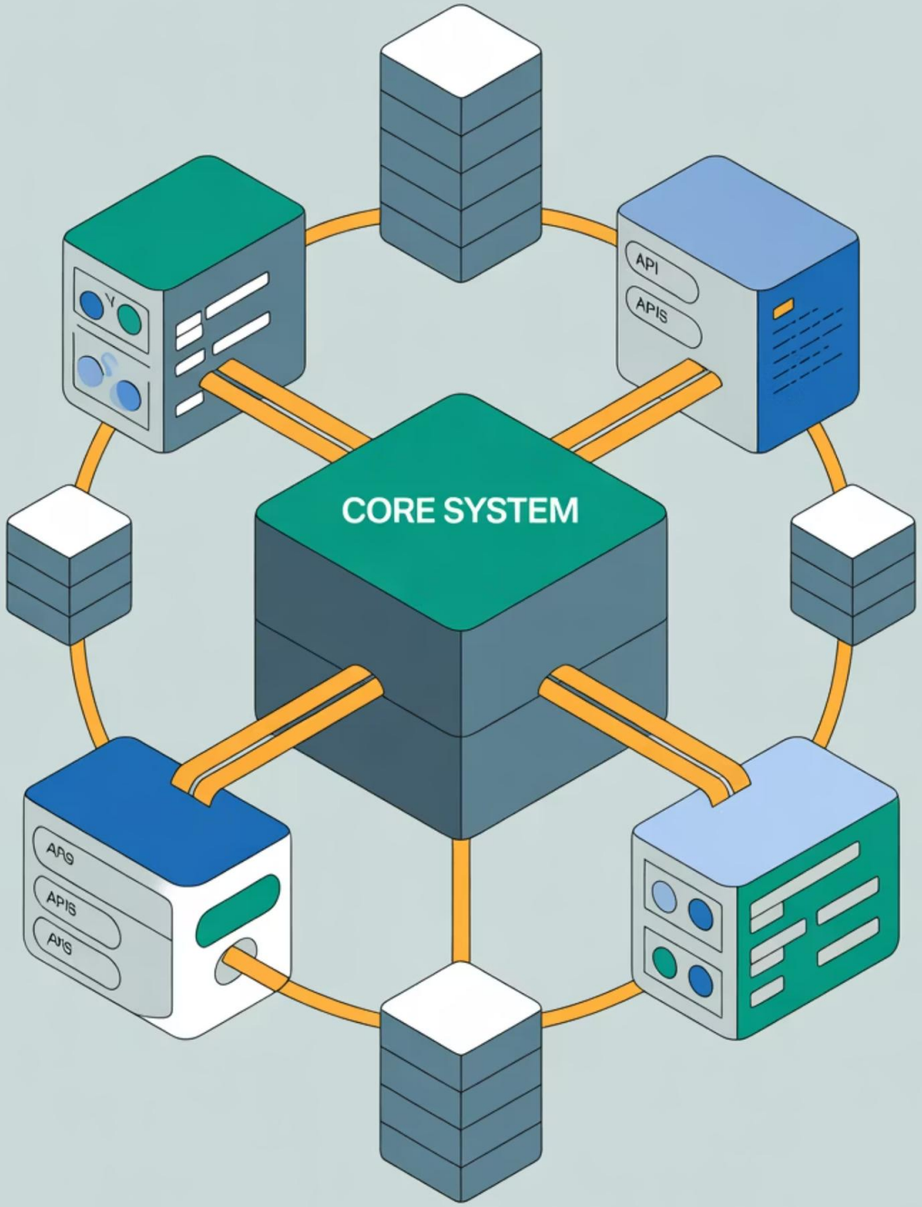
ALM Best Practices

Building robust environment strategies

Essential Environment Strategy

Development	Test/UAT	Production
Unmanaged solutions	Managed solutions only	Managed solutions only
Full editing capabilities	Validation and acceptance	Locked and stable
Experimentation allowed	User training environment	Performance monitoring

This three-environment minimum ensures proper separation of concerns while maintaining development agility and production stability.



Advanced Environment Considerations

Large organizations often benefit from additional specialized environments beyond the basic three-tier structure.

Sandbox/Training

Safe space for maker experimentation and learning without impacting development workflows

Pre-Production

Final validation environment that mirrors production configuration exactly

Deployment Pipeline Enforcement

1

Power Platform Pipelines

Native no-code deployment automation

2

Azure DevOps Integration

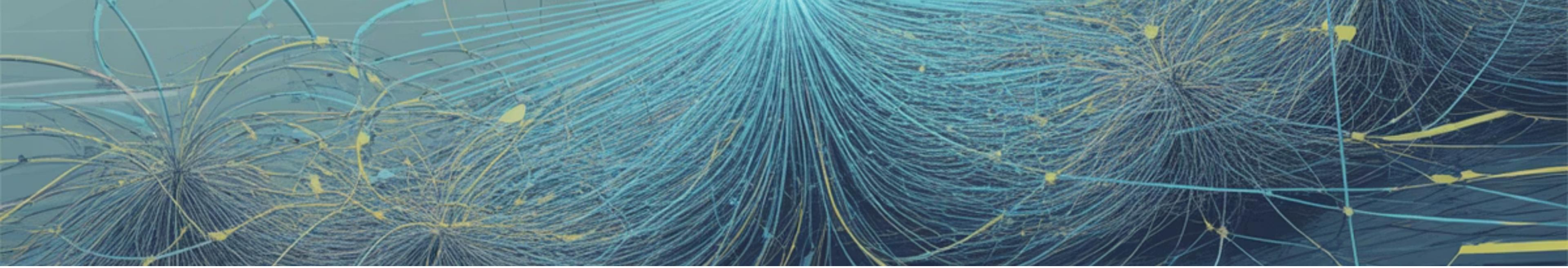
Advanced CI/CD capabilities with approval gates

3

GitHub Actions

Git-based workflow automation and collaboration

Choose the deployment method that aligns with your organization's existing DevOps practices and technical capabilities.



The Unmanaged Production Problem

Avoid at All Costs: Unmanaged customizations in production environments create "spaghetti layers" that break clean upgrade paths and make troubleshooting nearly impossible.

Maintaining discipline around managed-only production deployments is crucial for long-term maintainability and reduces technical debt accumulation.

Source Control Integration

Why Source Control Matters

- Complete change history tracking
- Team collaboration enablement
- Code review processes
- Branching strategies support
- Disaster recovery capabilities

Use the Solution Packager tool to unpack solutions into source-controllable formats, enabling standard software development practices.

Branching Strategies

Feature Branches

Individual development streams for specific features or fixes

Main/Release Branch

Stable codebase ready for production deployment

Development Branch

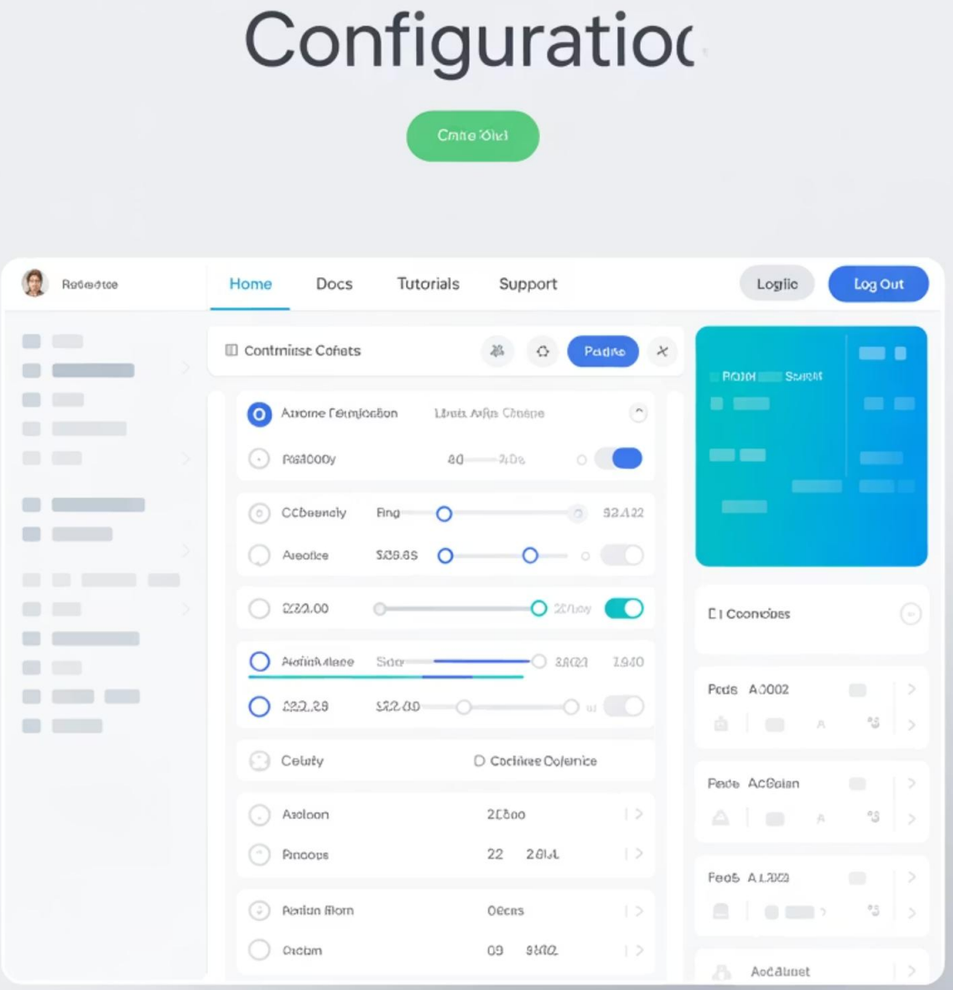
Integration point for completed features before testing

Align your branching strategy with your organization's release cadence and team structure for optimal collaboration.

Environment Variables: The Key to Portability

Environment Variables eliminate hardcoded values that make solutions environment-specific. Instead of embedding connection strings, API URLs, or credentials directly in your solutions, use variables that can be configured per environment.

> This approach makes solutions truly portable, enabling the same solution artifact to work seamlessly across Development, Test, and Production environments.



What to Store as Environment Variables

Connection Strings

Database connections and external service endpoints that vary between environments

API URLs

Service endpoints and integration points that differ across environments

Credentials

Authentication tokens and keys stored securely in Azure Key Vault

Configuration Settings

Feature flags and environment-specific behavioral controls

Secure Secret Management

❌ Don't Do This

- Embed API keys in flows
- Hardcode passwords in apps
- Store secrets in solution files
- Share credentials via email

✅ Best Practices

- Use Azure Key Vault integration
- Implement connection references
- Leverage managed identities
- Apply principle of least privilege

Modular Solution Architecture

Application Solutions

Specific business applications that consume core components

Core Solutions

Shared components, connectors, and base functionality used across applications

Extension Solutions

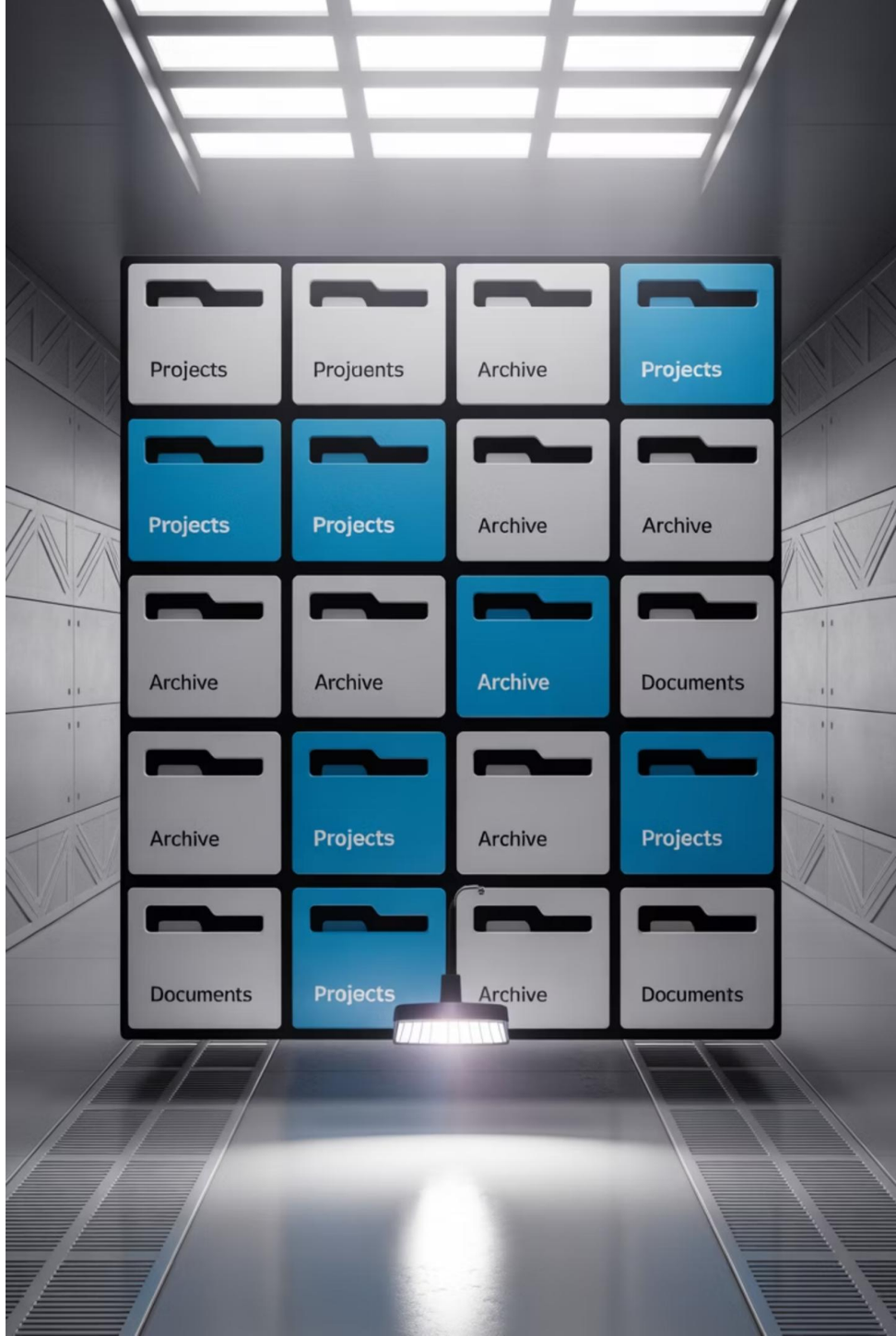
Add-on functionality and customizations for specific use cases

This modular approach enables better maintenance, reduces duplication, and allows for independent versioning of different solution components.

Naming Conventions Matter

Consistent naming conventions significantly improve long-term maintainability and team collaboration. Establish clear patterns early and enforce them across all solution components.

- **Solutions:** [Company].[Project].[Component] (e.g., Contoso.HR.TimeTracking)
- **Applications:** [Department] [Function] [Version] (e.g., Sales Lead Tracker v2)
- **Flows:** [Process] - [Action] (e.g., Employee Onboarding - Send Welcome Email)
- **Tables:** [Prefix]_[Entity] (e.g., hr_employee, sales_opportunity)



Release Management Excellence

01

Version Planning

Define scope and impact assessment for each release

02

Change Documentation

Comprehensive release notes and user communication

03

Deployment Execution

Coordinated rollout with rollback preparation

04

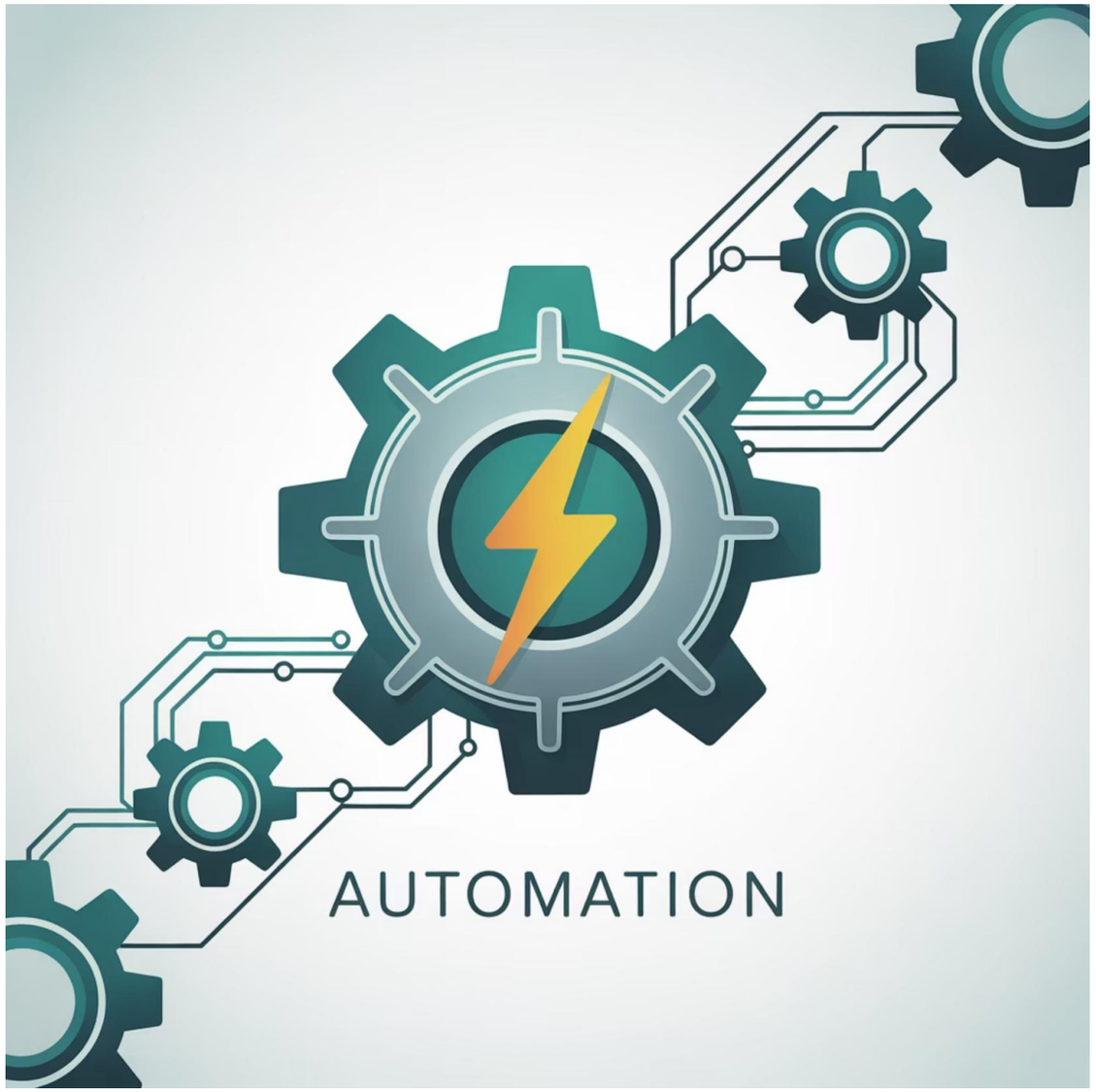
Post-Release Monitoring

Performance tracking and user feedback collection

Automation Excellence

Transforming ALM through intelligent automation

Why Automate ALM Processes?



The Automation Imperative

Manual deployment processes are inherently risky, time-consuming, and error-prone. As your Power Platform footprint grows, automation becomes essential for maintaining quality and velocity.

- Eliminates human error in deployments
- Ensures repeatable, consistent processes
- Accelerates time-to-market
- Enables DevOps alignment
- Provides audit trails and compliance

Automation Tool Comparison

Power Platform Pipelines

Best for: Organizations preferring no-code/low-code approaches

- Native Power Platform integration
- Visual pipeline designer
- Minimal technical expertise required

Azure DevOps

Best for: Enterprises with existing Microsoft toolchain

- Advanced CI/CD capabilities
- Comprehensive build tools
- Enterprise-grade security and compliance

GitHub Actions

Best for: Teams prioritizing Git-based workflows

- Seamless Git integration
- Community-driven actions marketplace
- Flexible workflow definitions

ALM Accelerator: Enterprise-Ready Automation

The ALM Accelerator for Power Platform, part of the Center of Excellence Starter Kit, provides prebuilt pipelines and governance guardrails specifically designed for enterprise Power Platform deployments.

> This accelerator dramatically reduces the time required to implement robust ALM practices, providing battle-tested templates and configurations.

Organizations can achieve production-ready automation in weeks rather than months, with built-in best practices and compliance frameworks.

The CI/CD Pipeline Journey

1. **Export**
   Extract unmanaged solution from Development environment
2. **Unpack**
   Convert to source-controllable format for change tracking
3. **Commit**
   Store changes in Git with peer review process
4. **Build**
   Package managed solution artifact with version control
5. **Deploy**
   Import to target environments with validation

Detailed CI/CD Workflow

Development Completion

Developer completes feature in unmanaged solution and triggers export process

Automated Export & Unpack

Pipeline extracts solution and converts to source-controllable JSON/XML files

Source Control Integration

Changes committed to Git branch with pull request for peer review

Build & Package

Approved changes trigger automated build creating managed solution artifact

Automated Testing

Solution deployed to Test environment with validation and smoke tests

Production Deployment

After approval gates, solution promoted to production with monitoring

Essential Automation Practices

Peer Review

Mandatory pull requests ensure code quality and knowledge sharing before merging changes

Automated Testing

Unit tests, functional tests, and security checks integrated into pipeline execution

Approval Gates

Human validation checkpoints for critical deployments, especially production releases

Artifact Storage

Version-controlled storage of deployment packages enabling quick rollback capabilities

Proactive Quality Assurance

Implement nightly builds and continuous integration to catch problems early in the development cycle, before they impact team productivity or project timelines.

Nightly Builds

Scheduled builds of development solutions to identify integration issues quickly

Continuous Validation

Automated checks for solution health, performance, and security compliance

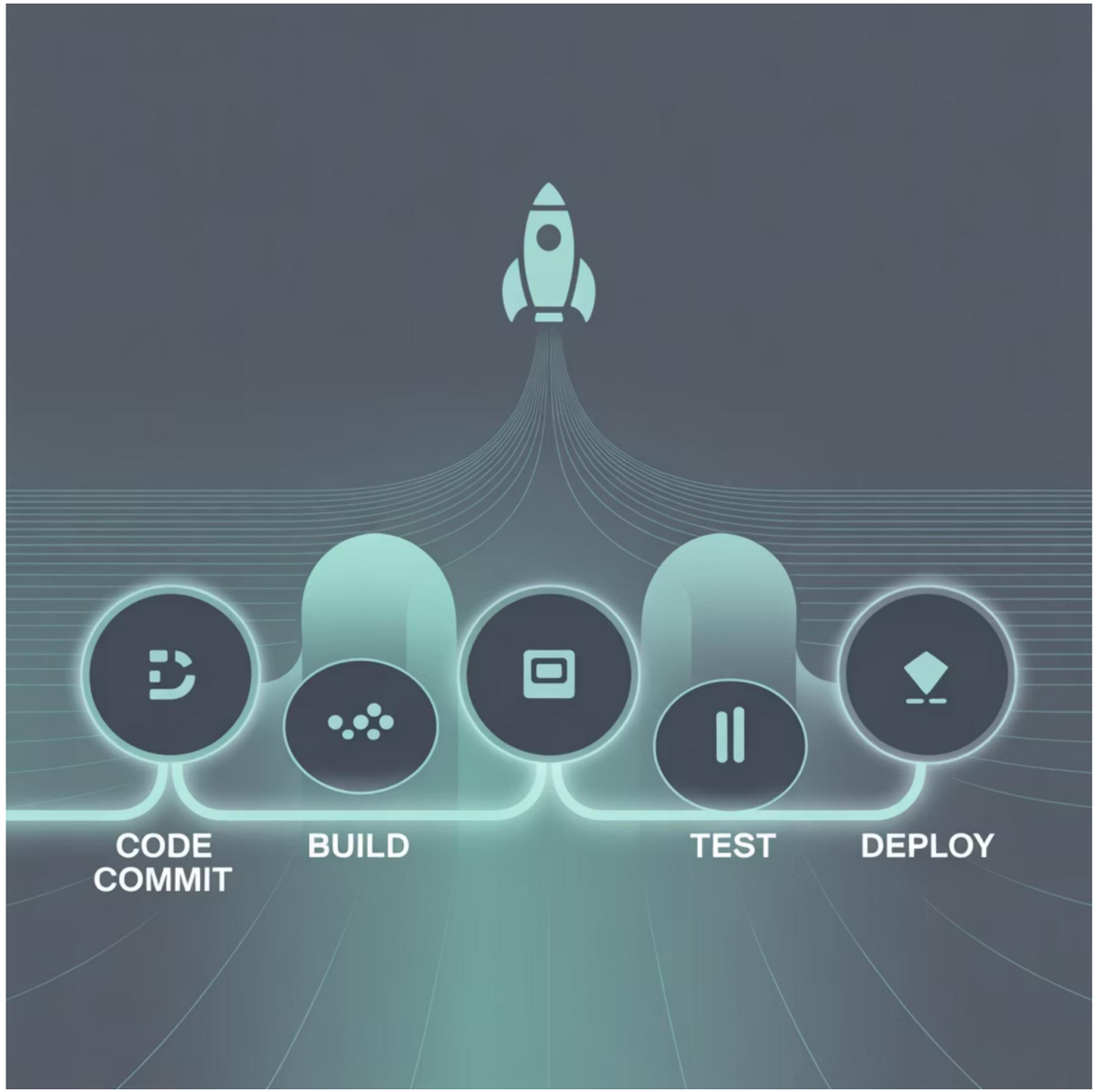
Advanced Automation Examples

Version Management

- Automatic version incrementing during export
- Semantic versioning enforcement
- Release note generation
- Change log automation

Environment Synchronization

- Configuration drift detection
- Environment variable updates
- Connection reference validation



Quality Gates

Environment Variables in Automation

1

Single Solution Artifact

Same managed solution works across all environments

2

Environment-Specific Configuration

Variables and connection references configured per environment

3

Automated Deployment

Pipeline handles environment-specific configuration automatically

This approach eliminates the need for environment-specific solution variants while maintaining proper separation of configuration from code.

Measuring ALM Success

75%

Deployment Time Reduction

Typical improvement with automated pipelines

90%

Error Reduction

Decrease in deployment-related issues

3x

Release Frequency

Increase in deployment cadence capability

60%

Time to Market

Faster delivery of business value

Your ALM Journey Next Steps

01

Assess Current State

Evaluate existing deployment practices and identify improvement opportunities

02

Establish Environments

Implement proper Dev/Test/Prod separation with managed solution policies

03

Implement Source Control

Integrate solution packaging with Git workflows and peer review processes

04

Deploy Automation

Choose appropriate CI/CD tools and implement automated deployment pipelines

05

Monitor and Optimize

Continuously improve processes based on metrics and team feedback

Excellence in Power Platform ALM

> "Mature ALM practices transform Power Platform from a productivity tool into a strategic enterprise platform capable of supporting mission-critical applications."

By implementing these practices—proper solution management, strategic environment design, comprehensive automation, and continuous improvement—your organization will achieve greater reliability, faster delivery, and reduced risk in your Power Platform initiatives.