Application Modernization with Microsoft Power Platform

Authors:

Robert Standefer (Microsoft)

David Yack (Colorado Technology Consultants)

Table of Contents

In today's rapidly evolving digital landscape, organizations face the constant challenge of modernizing their legacy applications to keep pace with changing business needs. Application modernization is crucial for improving operational efficiency, enhancing customer experiences, and staying ahead of the competition. Microsoft Power Platform offers a comprehensive suite of tools and technologies that empower businesses to transform and modernize their applications quickly and effectively.

Introduction

Legacy applications present many challenges for organizations. These applications are often built on outdated technology stacks and frameworks, making them difficult to integrate with modern systems and tools. They often have scalability and performance limitations that hinder an organization's ability to handle increasing workloads and customer demands. Legacy applications lack flexibility and agility, limiting their ability to adapt quickly to changing business needs and market dynamics. Security vulnerabilities, high maintenance costs, limited integration capabilities, and the risk of vendor dependency further compound the challenges of legacy applications. To overcome them, organizations need to modernize their application infrastructure to take advantage of new technologies.

The low-code development capabilities of Microsoft Power Platform make it possible to build and deploy modern applications faster and more cost-effectively than ever before. Easily integrate your existing systems and data sources for seamless data exchange and collaboration. Add artificial intelligence to enhance user experiences, automate processes, and gain valuable insights from your data. Whether you're a citizen developer tinkering around the edges or a pro developer working on a complex customization, you can drive digital transformation intuitively, quickly, and at lower cost than with traditional approaches.

This whitepaper explores the benefits, strategies, and best practices of modernizing applications with Microsoft Power Platform. It provides insights and guidance on how the Microsoft low-code platform can help you ensure the success of your application modernization efforts as part of your organization's digital transformation.

Why Power Platform?

The comprehensive tools and technologies that make up the Power Platform dramatically lower the length, cost, and development requirements of modernization and digital transformation projects. Its low-code approach reduces—and can even eliminate—the need for expensive coding, data science, and AI engineering resources. Citizen developers and pro developers alike benefit. Citizen developers can take an active role in the modernization process, building applications directly based on their domain expertise and reducing their dependency on IT teams. Pro developers can deliver even complex solutions in far less time, freeing them to move on to the next project sooner.

Power Platform products and concepts

Each product in the Power Platform family has a unique focus area. Organizations can implement the products individually or in combination to meet their specific requirements. The products are interconnected, forming a seamless whole, however they're combined.

The following table offers a high-level overview of each Power Platform product.

Power Apps	Create custom applications in an intuitive, drag-and-drop canvas. With more than a thousand connectors, internal and external data sources and services are just a few clicks away. Your apps can run in a browser, on the desktop, or on mobile devices.
Power Automate	Build workflows to automate even complex processes. Incorporate internal and external data sources and services using built-in and custom connectors. Use digital process automation (DPA) when applications have an application

	programming interface (API). Use robotic process automation (RPA) to automate repetitive tasks performed in a browser or desktop app. Trigger workflows to run when events occur in other systems and services or schedule them to run at a specific time.
Power BI	Drag charts, tables, and other visuals to a canvas to easily create sophisticated reports that uncover insights locked inside your data. Include automated machine learning for predictive modeling and AI visualizations with decomposition trees for detailed root cause analysis drilldowns. Explore your data by asking natural-language questions in a simple Q&A format.
Power Pages	Quickly build attractive, data-driven websites on a secure, enterprise-grade, low-code software as a service (SaaS) platform. With rich, customizable templates and a fluid visual experience, creating, hosting, and administering modern external-facing business websites has never been easier.

The Power Platform product family relies on a few supporting capabilities and concepts. The following table describes the most important ones to understand.

Power Fx	Power Fx is an open-sourced, low-code language inspired by Excel formulas. Strongly typed, declarative, and functional, with imperative logic and state management, all expressed in human-friendly text, Power Fx makes common programming tasks easy for citizen developers and pro developers alike. It supports the full spectrum of development, from no-code for those who have never programmed before to "pro-code" for the seasoned professional, empowering diverse teams to collaborate and save time and expense.
Connectors	Connectors are vital for allowing low-code and traditional coding to work together to deliver modern apps. Connectors are a wrapper around an API that allows Power Apps and Power Automate to use internal and external data sources and services. More than a thousand pre-built connectors are available, and you can create your own for any RESTful API. The connector definition includes the necessary metadata to make the API easy for low-code apps to consume.
Dataverse	Dataverse is a cloud-scale hybrid data store built on Azure data management services—but it's much more than a database. It's the underlying data platform for both Dynamics 365 and Power Platform, with server-side logic in the form of workflows and plug-ins, business rules and process flows, a highly sophisticated security model, and an extensible development platform with built-in support for multi-language and multi-currency apps. Applications can be quickly constructed from the data model, making it one of the fastest ways to deploy a form-over-data solution.

AI Builder	AI Builder makes it easy to use artificial intelligence in Power Apps and Power Automate to find insights in your data, automate processes, and make your apps more productive. With AI Builder, you don't need coding or data science skills to access the power of AI. Prebuilt, customizable models are turnkey-ready for many common business scenarios, and you can build your own models to meet a specific business need.
Copilot	Copilot AI assistance makes Power Platform users and developers, citizen or pro, more productive, allowing them to spend more time on the best parts of their jobs and less time on mundane tasks. Describe your business scenario to Copilot in Power Automate and it can turn your description into an automated workflow. Tell Copilot in Power Apps what you want to do or what information you want to see and it can build an app for it. Copilot sets up connections, creates and populates tables, generates screens, and even offers suggestions to make your flow or app better. Your apps will have copilot-powered experiences built in from the first screen—so your users can discover insights through conversation instead of clicks.
Environments and solutions	Environments are boundaries that contain and make it easier to manage and secure Power Platform resources. They're also used in application lifecycle management (ALM), in which solutions are developed and tested in separate environments before being deployed to a production environment. Solutions are packaged Power Platform customizations and extensions. A solution can include apps, flows, tables, charts, dashboards, connectors, and other components the customization or extension needs. Solutions can be developed, tested, and deployed to production in separate environments as part of an organization's ALM policy. You can export solutions to share them with other users or organizations and import solutions from others. Solutions are either managed or unmanaged. Unmanaged solutions are used for development and testing. Managed solutions are used for production deployment and distribution.

Key benefits of Power Platform for app modernization

The benefits of modernizing applications using the Microsoft Power Platform extend beyond the initial business value of having a solution that uses modern technologies.

- **Lower costs.** Organizations can save money on app development and maintenance. A commissioned study by Forrester Consulting found that organizations that use Power Platform can see a 45 percent decrease in application development costs and realize a 140% return on their investment.

- **Expand the resource pool and eliminate bottlenecks.** Professional developers, data scientists, and AI engineers are highly paid—and in high demand. Small- to medium-sized organizations often don't have the luxury of in-house coding expertise and outsourcing is expensive. The low-code Power Platform is more approachable by a larger pool of resources. Subject matter experts and employees with business process expertise can help accelerate modernization efforts, even if they've never written a line of code.
- **Build the cart, not the wheel.** Traditional software development starts fresh every time, reinventing the wheel with every new project. With low-code, intuitive, maker-friendly Power Platform products as your wheels, you can focus on building a better cart—improving your business processes—and enjoy the benefits of your modernization efforts sooner.
- **Reduce technical debt.** The cost—both financially and in lost opportunities—of upgrading "quick and dirty" software solutions and maintaining legacy infrastructure is high. Power Platform reduces this technical debt by making it easier and cheaper to build solutions right the first time, simplifying data integration and governance with a common data model and connectors, providing a centralized platform for managing solutions, and supporting continuous improvement with analytics and AI.
- **Enhance security and ensure compliance.** All Power Platform products include fully integrated, enterprise-grade security, compliance, and governance out-of-the-box, starting with the environments they run in. Managed Environments is a suite of tools that allow admins to manage Power Platform at scale, with more control and less effort. Among other capabilities, you can limit who can share which flows and apps and with whom and use policies to restrict the connectors makers can use. Native, flexible data security models mean each application doesn't have to build its own.
- **Modernize as you go.** The more significant the apps are that you want to modernize, the less likely it is that you'll want to replace them all at once. A low-code approach lends itself well to building solutions in manageable increments.
- **Integrate legacy apps.** Older applications often don't have APIs. Power Platform's RPA capabilities can automate classic apps and include them in your new modern business processes. RPA can also be helpful in incrementally modernizing large and complex apps.
- **Innovate without spending more.** Power Platform capabilities continue to improve. Apps built on the platform benefit from Microsoft innovations without additional cost to you.
- **Boost worker productivity in a modern workplace.** Power Platform is part of the Microsoft modern workplace. Applications modernized on the platform can take advantage of the capabilities of Microsoft 365, including engaging mobile experiences and easy, intuitive collaboration. Cutting-edge AI like Copilot, AI Builder, and features soon to be announced make users and developers more productive with less frustration and shallower learning curves.

Innovation for the frontline worker

Frontline workers need modern applications they can use on any device, anywhere they're working. They need access to insights in real time to make better decisions faster. They need to collaborate with co-workers and management to keep everything working smoothly. When American Airlines decided to modernize aspects of its operations, they got all that and more.

In partnership with Microsoft, American Airlines created ConnectMe, a Microsoft Teams app built on Power Apps and Azure. Using the app on any mobile device, frontline teams have key arrival, boarding, baggage, and gate information at their fingertips in real time, streamlining ground operations, accelerating aircraft turn times, and making travel a more pleasant experience for customers. Learn more about the airline's transformation.

AI empowerment for knowledge workers

Knowledge workers are swimming in an ocean of data—and too often, they feel like they're drowning. Nearly all applications collect data. Few of them help users make sense of the data they collect, let alone tease out insights that might help workers do their jobs better. AI capabilities can be added to apps as part of modernization, not just automating data collection and analysis, but also making it easier for knowledge workers to spot patterns and trends. Predictive analysis can use AI models to forecast future outcomes based on historical data with high accuracy, allowing leaders to plan with confidence. Modernized apps can include copilot AI, acting as a partner to generate content in context—summarizing interviews, drafting targeted marketing and sales messages, and even offering helpful information in real time while a customer service rep or salesperson is on the phone with a customer.

An incremental journey to modernizing legacy apps

If your organization is like most, you have a growing backlog of outdated applications that would benefit from modernization. Legacy applications typically use obsolete technology and are built on infrastructure—hardware as well as software—that's no longer supported. Often, only a few employees, usually the ones nearing retirement, know how they work. New staff want nothing to do with them because they can't use the modern tools they're used to or want to work with. Maintaining them, let alone updating them, requires scaling a mountain of technical debt that grows higher the more you climb. Years, perhaps decades, of patchwork maintenance results in a codebase that no one dares touch—especially when major parts of the business rely on it.

Organizations often can't easily replace these applications all at once. Instead, they embark on an incremental modernization journey. An incremental approach maximizes the benefits of modernization while mitigating some of the risks of a go-for-broke, one-time modernization effort.

Options for modernizing apps

Modernization doesn't always mean rebuilding a legacy app from the ground up. Other options are retiring it, replacing it, rehosting it, refactoring it, and rearchitecting it.

The following table describes each option, the ALM stage when it's most appropriate, and drivers that can influence its selection.

	End of life		Migration	Modernization		
	Retire	**Replace**	**Rehost**	**Refactor**	**Rearchitect**	**Rebuild**
Description	Remove app	Replace app with SaaS or another app	Redeploy as-is to the cloud	Optimize existing code	Shift code to a new application architecture or break it into microservices	Rewrite app from scratch with original scope and specs
Drivers	• No longer needed • Reduce expenses	• Reduce capital expenses • Take advantage of newer technologies	• Reduce capital expenses • Recover data storage • Quick cloud ROI	• Faster, shorter updates • More portable code • Greater cloud efficiencies in resources, speed, cost • Improve performance • Reduce technical debt	• Improve scalability, reliability, and maintainability • Ease adoption of new cloud capabilities • Mix technology stacks	• Accelerate innovation • Accelerate development • Reduce operating expenses
Microsoft technologies		Power Apps Dynamics 365	Azure IaaS Azure VMWare	Power Platform Containers Azure PaaS	Power Platform Azure PaaS Serverless microservices	Power Platform Azure PaaS Serverless microservices

The following table suggests ways that a low-code approach can apply to each of the app modernization options.

Rehost	Rehosting moves an app as-is from an older environment to a newer one. A low-code approach doesn't apply directly, but rehosting can be the first step before applying other strategies that would include low-code solutions.
Refactor or rearchitect	Refactoring tweaks the code so that apps can get the most benefit from a cloud-first environment. Rearchitecting significantly modifies the code. It could include encapsulating existing logic by moving it to an API that can be exposed to low-code solutions through a connector.
Replace or rebuild	Replacing swaps an app with a completely different one. Rebuilding recreates an app from the ground up. This option is commonly where a low-code approach achieves the best business outcomes. Starting with an application from Dynamics 365 or Microsoft AppSource can help jump-start modernization when the use case matches a prebuilt capability. Organizations can then use Power Platform components to customize the app for their unique needs.

Power Platform's low-code approach has the potential to offer much more than just another development tool. Making low-code part of your modern app strategy can also establish a foundation for empowering nondevelopers like subject matter experts to participate in your modernization effort. Organizations have found that establishing a Center of Excellence (CoE) around Power Platform and using tools like the CoE Starter Kit to create guidelines and governance helps users build low-code apps and automations successfully and ensures assets like APIs and components can be reused. Low-code can accelerate application development and help organizations extract value from their data faster, regardless of where it's located. In fact, many organizations decide to integrate a low-code mindset into their culture.

A guide to your modernization journey

It's easy to become overwhelmed when you start thinking about modernizing legacy apps. A guide can help you plan your journey and keep you on the right path along the way. A good place to start is with these three steps, considering each one with a low-code mindset.

- **Planning.** Think carefully about your goals for modernizing a legacy application and define your strategy for reaching them. Have a clear statement of the problem you want modernization to solve. This is the time to assess your apps and environments with an eye to what's not working, what's working but could be improved, and—most importantly—what value, to the business or to users, will result from any changes. Evaluate each modernization opportunity for its potential to take advantage of a low-code approach. Prioritize opportunities that incorporate low-code solutions. Use the Cloud Adoption Strategy Evaluator to build a business case for application modernization.
- **Implementation.** Modernize your apps not just incrementally, but iteratively. An iterative approach gives organizations the flexibility to change their project scope or strategy as needed. Power Platform low-code solutions can be developed and tested faster than traditionally developed apps, and deployment in Managed Environments requires just a few steps. While low-code requires less upskilling than traditional coding, make sure your employees are appropriately trained in how to work as fusion teams, which blend low-code and traditional resources.
- **Operations.** App modernization doesn't stop at implementation. With a low-code, cloud-first approach, you can use cloud platform services and tools to secure, govern, manage, and optimize your apps.

Evaluate opportunities for low-code solutions

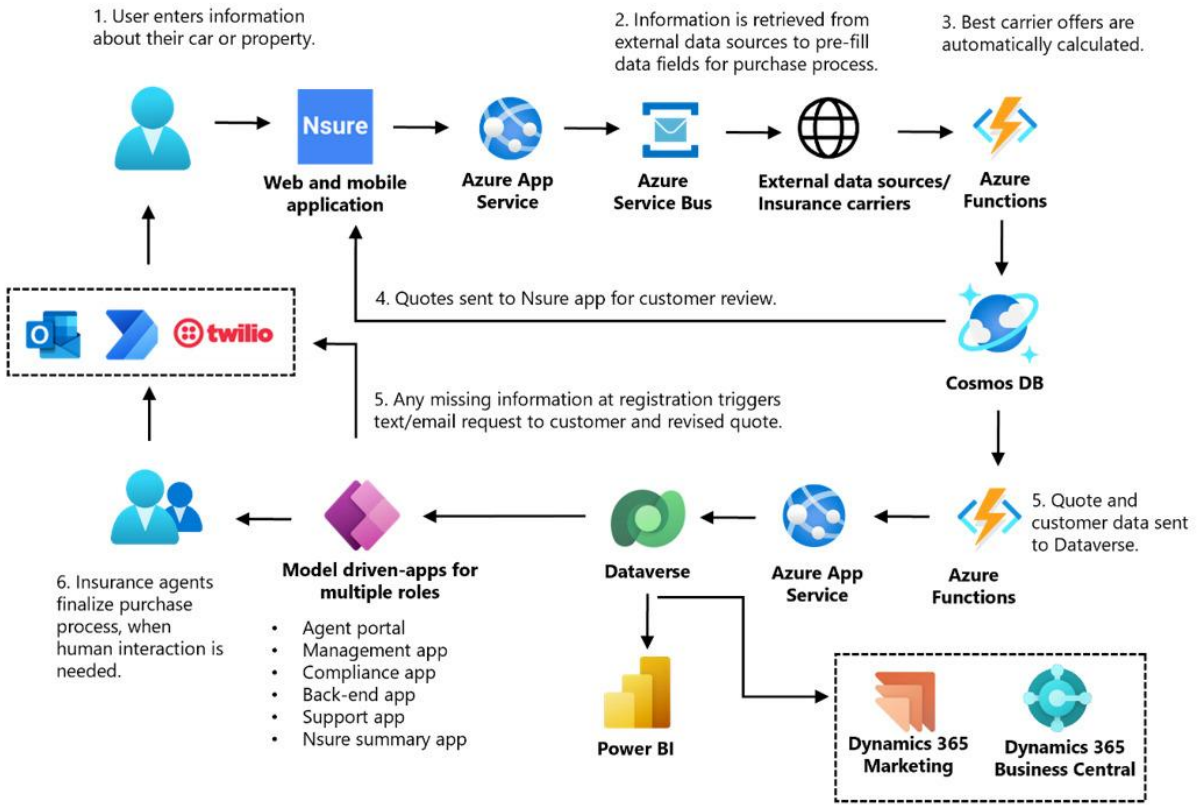
Organizations use a variety of methods, from informal review to detailed decision trees, to determine whether a low-code approach is the right way to modernize a legacy app. The most important thing to consider is that low-code isn't an all-or-nothing decision. Building part of an application out of Power Platform components and part of it out of components developed using traditional coding techniques is common.

To evaluate an application, we recommend that you first determine whether it's still needed and useful or should be retired. If you decide it's still needed, assess whether a low-code solution can replace the app as a whole. If the entire app isn't a good fit for a low-code replacement, assess whether one or more of the app's workloads or components might be. You may find that a low-code solution extended with traditionally developed code provides the best of both worlds.

For example, if you determine an application isn't a good fit because Power Apps is missing a required control, you can use the Power Apps component framework (PCF) and traditional code to build a custom control. Another example is an app that has complex logic. You could centralize the logic in an API that

Power Apps can access using a custom connector. In both these examples, Power Platform extensibility allowed most of the app to be built with low-code components, bridging the gaps with traditionally developed code.

NSure.com, a proprietary online insurance shopping platform, offers a real-world example. The company's initial launch relied on traditionally developed Angular, Xamarin, and Azure services. By adding Power Platform and Dynamics 365, NSure.com created a next-generation solution using both low-code and traditional coding techniques, as the following diagram illustrates. [Learn more about the company's journey.]



Just as important as identifying low-code opportunities is recognizing when a low-code approach isn't the right one. The following tables describe use cases that are commonly not a good fit for low-code solutions. Organizations face different challenges on the front end and the back end, so let's consider them separately.

Front-end scenarios that don't fit a low-code approach

User device isn't compatible	Power Platform recognizes mobile devices and specialized devices like barcode scanners. Devices that depend on specific APIs or drivers might not be supported and would require a more traditional approach.

Heavy volume of client-side data	The user experience in some applications requires large quantities of data, a challenge for any technology, not just low-code. Downloading and processing that much data can stress back-end systems and degrade the performance of both the app and the device it runs on. Users aren't as productive when they're forced to navigate a sea of data. Before turning to traditional coding methods to handle the load, explore whether proper filtering and navigation can provide a better user experience.
Complex offline requirements	Applications that need to work in places where connectivity is poor or nonexistent can be challenging to implement and support, whether they use low-code or traditional code. Power Apps offers basic capability for simple offline scenarios. For example, an app that captures leads during an event and uploads them to a marketing database after the event would work fine. For applications that require files and images, non-Dataverse connectors, or complex conflict resolution, you should look to traditional code techniques.

Back-end scenarios that don't fit a low-code approach

High-velocity data	Importing millions of rows of data as part of migrations and similar events is commonly supported. However, workloads that involve processing millions of rows of data hourly or daily should be subject to additional evaluation. For example, collecting high volumes of Internet of Things (IoT) telemetry into Dataverse wouldn't make sense. Instead, Azure cloud services could be used to collect and analyze the data and relevant signals added to Dataverse to trigger actions in the application. Applications that involve a high volume of updates to Dataverse data on a regular basis might require the assistance of traditional code to scale the updates.
Background workloads with complex logic	Background workloads that involve complex logic or a high volume of API calls might not be right for a low-code solution. Instead, the logic can be centralized in an API that a low-code solution can call.
APIs that use non-RESTful protocols	Power Platform connectors support only REST APIs. If you need to connect to another style API like SOAP or gRPC, provide your own REST API that communicates with the incompatible one.

We recommend keeping up with Power Platform release waves because they continue to close gaps in what you can do with low-code solutions. Creating a low-code proof-of-concept is a good way to determine whether your scenario is a good fit.

Prioritize low-code opportunities

As you evaluate your application portfolio, it isn't enough to identify good candidates for low-code transformation. Your team must prioritize them to maximize the success of your modernization efforts.

Prioritization should consider the following factors:

- The low-code maturity of your organization
- The complexity of the opportunity
- ROI for the organization, users, and IT
- Time to value

Being realistic about your organization's low-code capabilities can help you choose an opportunity that challenges your team to grow but doesn't overwhelm them to fail. You don't have to pick the most straightforward application without any challenges. An ideal one would offer some opportunities to explore how to combine traditional code with low-code solutions.

Applications with complex integration with other systems are often not the best place to start. Trying to tackle applications that are too large or too complex can lead to frustration and failure. Avoid any that are questionable low-code candidates. Save them for after your team has completed a few successful modernizations.

When modernizing a large, monolithic app, consider whether you can modernize small parts of it incrementally. Monolithic applications used to be common. Now, smaller role- or task-focused apps are more common. They allow for incremental development and enhancements as well as scaling of the teams building them.

The first few modernizations are important because they let the organization see the impact of low-code solutions. Evaluating the benefits and risks to an app's stakeholders is important when you're prioritizing opportunities. Choosing an application that nobody cares about or has a low impact on the organization won't be the best demonstration of the advantages of a low-code solution.

User adoption of the modernized application is important. Users need to feel that their new low-code application fits alongside the rest of the applications they use. Another risk to adoption is the degree of customization they're used to. If they expect a highly custom-built experience, they might be less inclined to adopt a low-code solution that feels less personal.

Organize and upskill your teams

Organizations that are successful in modernizing their legacy apps don't just assign a modernization project to a team of traditional code developers and hope they succeed. It's important to give your team the knowledge and confidence in low-code development it needs to successfully complete a modernization effort.

A team of low-code resources working alongside traditional code resources is referred to as a fusion team. Fusion teams are designed to encourage collaboration by training both types of resources on integrating low-code solutions with traditional code. A solution architect establishes how the solution will be architected between low-code and traditional code.

While it's easy to default to assigning all the work to traditional developers, low-code modernization efforts are good opportunities to expand the project team. Many business users make excellent low-code resources. They can accelerate the team's work because they already understand the business problem. They just need to learn how to complete the types of low-code work the team will take on and be familiar with testing and ALM procedures. That might mean learning how to build apps in Power Apps or workflows in Power Automate. They should also understand what traditional coders can develop to make their low-code efforts easier. This doesn't mean they need to know how to write traditional code.

Traditional code resources need to have a basic knowledge of low-code approaches and how these differ from the code they typically write. Most importantly, they need to learn the extensibility options of low-code solutions. They should be comfortable creating a test app or flow that uses code they write to make sure it works and to be ready to support low-code resources in consuming their extensibility assets.

Both low-code and traditional code resources need to understand where low-code and traditional code solutions begin and end and where they intersect.

Gather requirements

You may find that you have apps that are a decade or more old and not documented. They may require reverse engineering or business user knowledge to recreate. It's important to remember that while a low-code approach is efficient, it doesn't make gathering requirements and business process knowledge faster or make complex requirements less complex. Often there's an unrealistic expectation that a team that's modernizing an app will accomplish as much as a team that's building a new app with low-code. Establish your organization's expectations with these challenges in mind.

Two approaches can help accelerate the effort of gaining knowledge about a legacy app. First, expand the team to include business users with domain knowledge. Second, focus on understanding the business process and its desired outcome rather than documenting how it's implemented in the legacy system. The exception to this is if the legacy application requires specialized logic that executes business rules that you must follow.

Avoid working against low-code approaches

Organizations that are new to modernizing applications with low-code solutions often make the mistake of developing low-code the same way they develop traditional code. For example, an organization might apply UX standards written for Angular apps to its first Power Apps implementation. The project team would spend unnecessary time trying to meet standards that were designed for Angular framework capabilities and not business needs.

Teams that are used to working with traditional code might attempt to minimize low-code. For example, instead of using Power Apps controls, the team could build an application out of Power Apps component framework controls to avoid using low-code as much as possible. It's best for teams to get as far as they can with low-code until they hit blockers that can't be worked around. Teams that learn how to take advantage of the platform's capabilities are more successful in achieving the maximum benefits of low-code. Low-code continues to become more capable of taking on what used to be possible only with traditional code. A common challenge in the past was getting stuck because low-code couldn't replicate some needed functionality. Power Platform addresses this challenge with extensibility options that allow mostly low-code applications to incorporate specialized components written in traditional code when necessary.

Low-code approaches can play an important role in your modernization strategies. The best results require a clear statement of the problem the modernization effort is intended to solve, planning, staffing that reaches beyond default roles, training, and prioritizing. Making appropriate adjustments to their

standards and processes if needed also helps organizations realize the full potential of low-code. Modernization done right should improve the overall business value of the modernized applications.

Low-code platforms have evolved rapidly over recent years. While they have always been good at supporting individual productivity scenarios, the recent focus has been on enterprise capabilities. Organizations are building low-code applications that support the modern workplace, including hybrid work (remote and on-site) and the accompanying need for ways to encourage collaboration. Low-code platforms like Power Platform can now scale to handle applications that all users in an organization can rely on and that can integrate into enterprise security models. By connecting low-code capabilities to enterprise infrastructure, you can use low-code techniques side-by-side with traditional methods. Low-code abstracts much of the complexity and allows a broader set of people to participate in building solutions.

Understand the cost structure of a low-code approach

A common question that organizations ask when they consider a modernization effort is how much will it cost? While a complete discussion of licensing and cost analysis is beyond the scope of this paper, we can explore these topics at a high level.

Power Platform products are licensed products. You can license them individually to match your requirements. You can configure Azure billing for pay-as-you-go, which allows use without an up-front license commitment or purchase and includes some in-app Power Automate usage. Power Automate also has per-user and per-flow licensing for standalone work. The per-flow licensing works well when you have automation that benefits the whole organization. Power Apps licenses can be per user or per app. Power Pages sites are licensed per user, site, or month. An additional license is needed for authenticated sites. All licenses include using connectors and Dataverse, with the option to license additional storage and API requests for high-volume scenarios.

All Power Platform products have volume pricing that commonly applies to application modernization efforts and must be evaluated for each organization's unique strategy.

When evaluating the cost of low-code compared to traditional code, it's essential to realize that the comparison isn't apples to apples. With low-code, you pay for the labor to implement a unique business process in the low-code product and for the product license to use it. In general, the license includes multiple apps and automations without each one requiring additional cost.

With traditional coding approaches, you pay for the labor to implement a unique business process in code, the labor to build the app's infrastructure, and the cloud services needed to support the app.

All solutions, whether low-code or traditional code, require ongoing maintenance and upkeep. However, low-code solutions require significantly fewer resources to do it. They also incur less technical debt because the app infrastructure is provided by the platform.

Compared to a fully custom application that's not built on top of a low-code platform, a low-code solution has a more predictable cost. Avoid falling into the trap of simply comparing low-code licensing to the initial cost of traditional code deployment.

A look inside the Power Platform

Power Platform components are built on the same Microsoft Azure cloud services that are available if you use traditional coding methods. The integration of these components with each other and with security, scalability, and disaster recovery features has been done for you.

Inside Dataverse

Dataverse is powered by more than 25 fully managed Azure services such as Functions, Load Balancer, Cognitive Services, Synapse, DevOps, Active Directory, and Microsoft Purview. Built-in capabilities include comprehensive security, powerful analytics, AI, advanced business logic and event handling, data modeling, and integration with Dynamics 365, Microsoft 365, Azure, and more. All these capabilities are built on a polyglot Dataverse storage layer, which is based on Azure SQL DB (for relational data), Azure Cosmos DB (for NoSQL), Azure Blob Storage (for files), and Azure Data Lake Storage Gen 2 (for large-scale analytics and long-term data retention). They're available for transparent use in Power Platform's low-code components and through the Dataverse REST API.

High availability and business continuity and disaster recovery (BCDR) are important for business-critical applications. Dataverse maximizes availability with Azure reliability services. Replication of primary and secondary servers is synchronous, with a fabric underneath that can detect failures and choose a new primary following correctness protocols. High-availability failovers, which occur *within* an Azure region, are seamless and rarely noticed by users, typically happening in seconds. They're guaranteed to have zero data loss regardless of whether the outage is planned or unplanned.

Disaster recovery failovers occur *across* two Azure regions. To ensure faster failover with minimal data loss, a disaster recovery continuous copy is maintained using asynchronous replication. Planned failovers incur no data loss and, for production environments, can usually be completed in seconds or a few minutes.

In addition to the technical implementation of high availability and BCDR, the operations team regularly tests its readiness to respond to various types of events.

Inside Power Automate

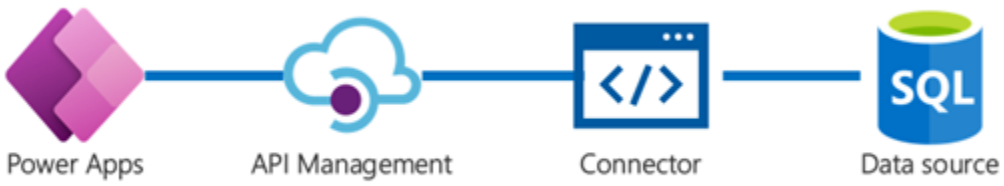
Power Automate cloud flows are built on top of Azure Logic Apps. Power Automate provides abstractions and integration with other low-code components like Power Apps and uses the Logic Apps runtime engine. Developers who are familiar with Logic Apps will find Power Automate uses similar concepts, including the expression language.

Inside Power Apps

The Power Apps runtime engine is built on the React framework. Applications are built in the Power Apps designer, which uses a drag-and-drop interface to build screens. Power Fx formulas implement logic. Connectors extend apps' access to other services and logic and components that allow reusable visual extensions. Developers can use the Power Apps Component framework (PCF) to create custom controls. While many UI frameworks can be used alongside PCF, Power Apps features built-in support for React.

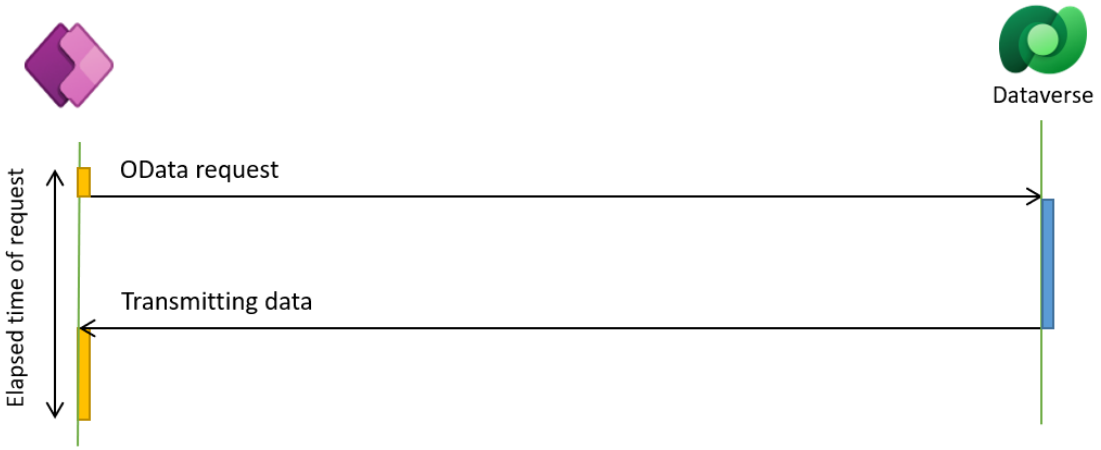
Inside connectors

Connectors use Azure API Management to manage credentials and connections from each user.



The same architecture is used for all connectors, including custom connectors that you create for your own APIs. The use of Azure API Management ensures a consistent interface for Power Platform products like Power Apps and Power Automate with all connectors.

An exception is the Dataverse connector. It appears in the list of connectors for apps and flows, but it's implemented differently. When an app or a flow uses Dataverse data or actions, the interaction is direct though Dataverse's OData API.



Power Platform extensibility options

Extensibility is a key feature that differentiates Microsoft Power Platform from other low-code platforms. A guiding principle of the platform is "no cliffs"—you shouldn't be blocked from accomplishing something using low-code, even if it requires traditional code. You can build an entire workload as part of a larger app using traditional code if necessary. However, the platform offers many extensibility options that allow low-code and traditional code to be used together in the same workload.

The following table provides a high-level overview of some of the common extensibility options. We'll refer to some of them again later, when we discuss how to approach modernization and patterns that you can apply.

APIs and custom connectors	Custom connectors for your REST APIs centralize app logic and allow it to be exposed to low-code components in a secure and governed way. You can use this approach in an API-first strategy for application modernization. The custom connector uses an OpenAPI document to define how a low-code component can interact with the REST API. For example, you could create an API using Azure Functions and publish it to Azure API Management. Azure API Management can export an OpenAPI definition to automatically create the custom connector for use in a low-code solution. This approach decouples client applications from the APIs, allowing them to evolve independently. The APIs are centrally managed, adding a layer of security by not exposing the API directly and using authentication techniques like subscription keys, tokens, client certificates, and custom headers.

Power Apps Component framework	The Power Apps Component framework is an extensibility framework for creating custom visuals for Power Apps and Power Pages. The code components are created using HTML, JavaScript, or TypeScript. Think of code components as UI building blocks that can be reused to build one or more apps. The components include a manifest that defines how a low-code component can interact with the code component. The component interface allows the hosting runtime engine to communicate the hosting container's lifecycle events. This allows the code component to render its visuals using context information provided by the hosting container. For ideas, browse the community gallery at https://pcf.gallery.
Virtual tables	Virtual tables make it easier to integrate data that resides in external systems. They seamlessly represent the external data as tables in Microsoft Dataverse, without replicating the data and often without the need for custom coding. Dataverse ships with data providers for OData v4 and Azure Cosmos DB. A virtual connector provider, currently in preview, expands the available data providers to include a subset of the Power Platform connectors, including SharePoint and SQL Server. For more advanced scenarios, developers can create custom data providers. Creating custom data providers requires deep knowledge of both the external data and Dataverse. The ability to create Dataverse plug-ins using Power Fx for the logic is in preview.
Dataverse plug-ins	A Dataverse plug-in is a custom event handler that executes in response to a specific event. Think of plug-ins like stored procedures in a database engine but written in .NET. For example, events are raised during processing of a Microsoft Dataverse data operation or on demand for custom API events. The plug-in is implemented as a custom class compiled into a .NET framework assembly that can be uploaded and registered with Dataverse. Using a defined interface, the plug-in can get context information about the event being processed. Plug-ins can run as part of the Dataverse transaction and can perform other data operations that are part of the current transaction. Plug-ins are intended for small units of work. Their performance must be optimized so they don't negatively affect overall performance. Plug-ins are always executed, regardless of operations from the user interface or API, making them a powerful way to consistently enforce business logic.

Exploring low-code modernization architecture scenarios

As with most platforms, you can compose an endless number of architecture scenarios using Power Platform components and other Microsoft cloud services. In this section of the paper, we'll explore some

of the more common scenarios and discuss some considerations you should keep in mind when using them.

Application experiences

Modernizing the user experience can make a big difference to users. Power Apps is the primary way to build internal application experiences with the Power Platform. You can use Power Pages for internal web applications, but it's more common for external-facing applications.

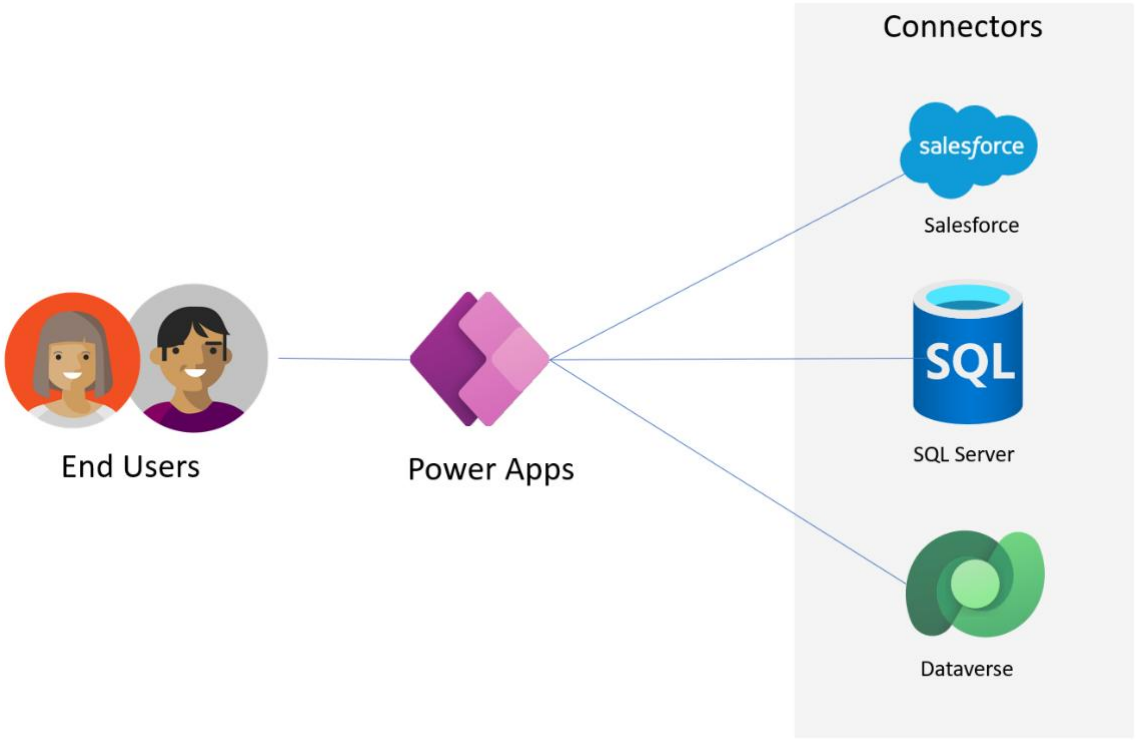
Power Apps

Workloads should be designed so that users can complete much of their work without switching apps. When you're modernizing a large, monolithic application, you might split its functionality into multiple apps. Conversely, if users need to work with multiple apps, you might consolidate them in a single app that presents a unified view of their multiple data sources.

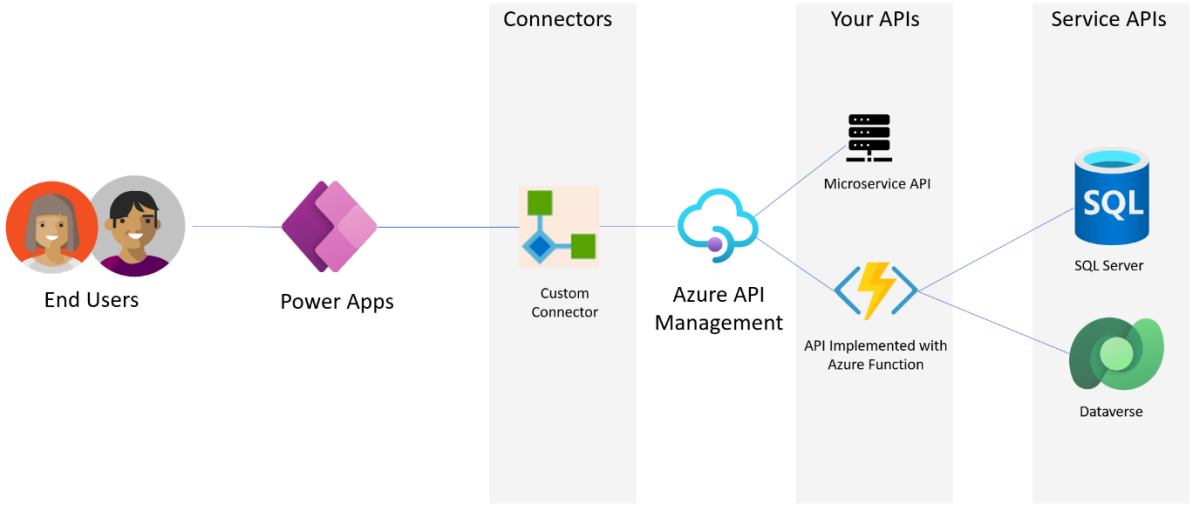
The following table describes the two types of apps you can build with Power Apps, canvas apps and model-driven apps.

Canvas apps	Canvas apps are highly customizable. They consist of one or more screens that users interact with. You control the layout of each screen and the navigation across screens. Canvas apps work with data using connectors. A single app can work with multiple connectors, making it good for integrating across multiple data sources as a composite app.
Model-driven apps	Model-driven apps use Dataverse as the primary data source. They consist of one or more pages, which can be Dataverse tables or custom pages. A Dataverse table page can be drilled down to a detail page for viewing and editing. Custom pages can incorporate a canvas app screen and data from connectors. Model-driven apps have a customizable built-in navigation structure. It's consistent across all model-driven apps, which helps with user adoption.

The following diagram illustrates a basic architecture for a canvas app or a model-driven app, in which the app connects directly to data sources.



To minimize direct connections to a data source, you can have the app use a custom connector to the API, which does any necessary work in the data source. This approach allows you to control what operations are exposed to low-code components and can abstract the complexity of the underlying logic. The following diagram illustrates this API-first approach.



Power Apps can also directly run Power Automate cloud flows that can return results to the application or run asynchronously.

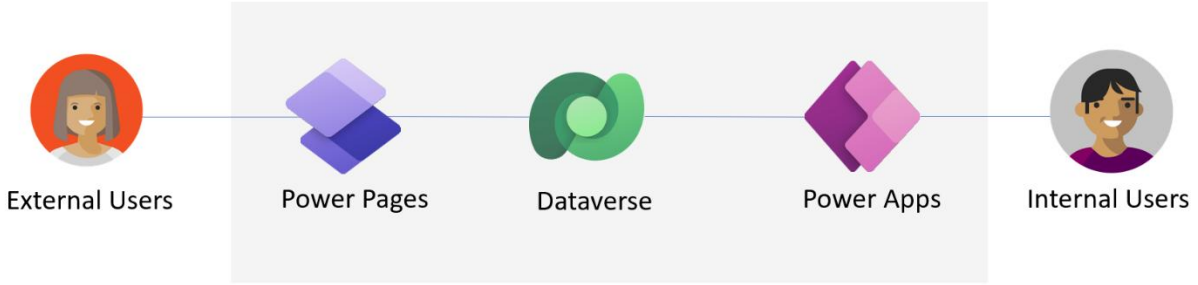
Using Power Apps with your data repositories or APIs allows you to modernize the user experience while minimizing disruption to other parts of a legacy solution. This approach can also allow you to connect multiple legacy systems into a single application, giving users a single place to complete their work.

Power Pages

The primary data source for Power Pages is Dataverse. When you add pages to a website, you store the page definitions in Dataverse. Pages can both present Dataverse data and collect data from users to store in a Dataverse table.

You can configure pages for anonymous access or for authenticated access using Azure Active Directory (Azure AD) for internal users or identity providers for external users. When authenticated users access data, only the data they have permission to access is available.

A common application of a Power Pages site is providing external users with self-service access to an organizational business process. Internal users can use a Power Apps application. The following diagram illustrates such an architecture.



Data management

Application modernization requires evaluating the data that's used in the overall solution. Modernized applications have multiple options for handling data. In many cases, multiple applications use the same data repository. It becomes difficult to simply migrate the data to a new repository as part of modernizing one of the applications. A core tenet of Power Platform is that data can be used where it is or brought into the platform in either Dataverse or a data lake.

You have the following options for the data architecture of the modernized application:

- **Leave the data in place:** Use connectors or APIs with custom connectors to access the data where it resides. When the data is on-premises, the data gateway can facilitate secure connectivity. Use virtual tables to integrate compatible external data as a Dataverse table.
- **Migrate to Dataverse:** Dataverse is a good option for transactional data and for consolidating multiple sources into a single system of record. Data can be mapped and migrated from many sources using Power Query and automated flows. Dataverse also supports elastic tables, currently in preview, designed for ingesting high-volume data that's stored in unstructured or semi-structured formats.
- **Migrate to data lake:** For historical, analytical, or telemetry data, use a data lake. The data in the lake can be used to generate Power BI analytics or processed to generate AI-powered insights.

When evaluating options for the data architecture of a modernized application, keep the following considerations in mind:

- **Impact on other applications:** While migrating to a more efficient data store might be ideal for one application, the initial impact on other applications that use the data might be too high. Some organizations consider leaving the data in old data stores and creating new data in Dataverse, migrating from the old store as more applications are modernized.
- **Impact on new applications:** Leaving data in an old data store, while easy, may negatively affect how easily modernized applications can use it. Older data stores may not have good integration with other cloud services, making it harder to incorporate the data in the new overall architecture.
- **Data consolidation:** It's common as part of application modernization to identify data that doesn't have clear ownership or responsibility for ensuring its proper use. By consolidating their data in Dataverse, organizations can improve their governance of it and better ensure it's used properly.

  Data privacy and security: You should evaluate privacy and security based on your current needs and target modernization architecture, not just on how the legacy application handled them. Cloud solutions have more options for implementing privacy and security controls. Often, a single data store can simplify them. You must also consider how to implement unified data security in hybrid applications that split data across multiple repositories.
- **Integration issues.** Older data stores may lack the APIs necessary to allow access without either migrating the data or creating an API that applications can use with a custom connector. Connectivity from the old data store to the applications using it should be evaluated to determine if performance would be acceptable.

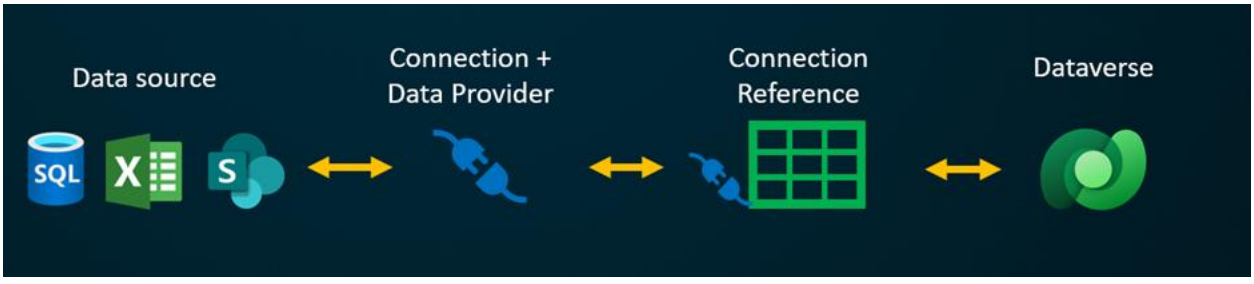
You should determine a data architecture for each application that will be modernized. A first step is to establish an overall vision of how your data architectures will incorporate Dataverse. If the goal is to maximize the value of low-code, then you should use Dataverse whenever possible. Having a vision at the start can help you avoid propagating more silos of data.

External data and Dataverse

Legacy applications often rely on data that resides outside the organization and existed long before Dataverse. Modernizing these applications doesn't need to involve duplicating the data in Dataverse. Instead, you can represent the data as virtual Dataverse tables. Virtual tables can participate in relationships with other virtual tables and with local tables. The modernized applications see a unified set of tables that appears to exist entirely in Dataverse.

Virtual tables are implemented using a data provider architecture. Dataverse includes an OData provider that can be used with OData V4 web services. A virtual connector data provider, currently in preview, allows using tabular Power Platform connectors as virtual tables.

The following diagram illustrates the use of the virtual connector.



Developers can also create custom providers for other external data sources. However, they must understand and implement all Dataverse mappings and operation support.

The following considerations can help you evaluate the use of virtual tables in your modernization projects:

- All external data sources must have a primary key, and the data provider must present it as a GUID to Dataverse. You can accommodate non-GUID keys with padding if the padded value is stable and unique.
- Data security is configured at the virtual table level. Row- and column-level security isn't available.
- The performance of virtual tables depends on the data provider, the external data source API, and the connectivity to the data source. In most cases, virtual table access is slower than with local Dataverse tables.
- Some Dataverse features like search, auditing, charts and dashboards, and offline access aren't available for virtual tables.
- Using virtual tables for reference data can result in reduced synchronization.

File and images

When modernizing applications that use files and images, it's important to consider where the new solution will store them. Dataverse has specialized capabilities for storing files and images. Both can be added to tables as a column and are stored in Azure blob storage that's managed by Dataverse. Apps can work with them using the Dataverse connector, no separate authentication or API required.

Using Dataverse for files and images is appropriate when they have a direct connection to the data and multiple users don't need to collaborate on them; for example, a photo of a product or a location or the final copy of a legal contract. However, if multiple users need to modify the legal contract concurrently, using SharePoint would provide greater collaboration capabilities. Consider using Azure blob storage directly if you need to have security managed separately from Dataverse or if you need to use certain file-specific features.

Integrations

Modernizing applications often includes integrations with internal or external systems. Integrations can be broadly categorized as data, application, or process.

- **Data integration** combines data from different sources to give the user a unified view. It offers a decoupled approach but doesn't allow the construction of logic or processes in real time. Performance can be better because all the data is local.
- **Application integration** connects at the application layer and is commonly done through APIs or, with low-code solutions, connectors. Application-level integration provides a defined boundary between two solutions but also creates a real-time dependency in many cases. This type of integration also creates a security boundary, where access can be controlled by the system that's providing the API.
- **Process integration** connects multiple disparate systems, each of which is part of an overall business process. This type of integration is the most decoupled, allowing the participating systems to handle each part of the business process. In modernization scenarios, it can be helpful to partition a part of a process for modernization with low-code, integrated with other parts still handled by a legacy system.

When evaluating how you'll implement integrations, it's important not to assume the old approach is the best one for the application you're modernizing. For example, if a process is real-time and synchronous, consider whether you could do it asynchronously. Synchronous integration can be more fragile in a cloud solution. For example, a low-code Power Automate flow with appropriate error handling could orchestrate

the integration. Not only would this improve reliability, but it would also improve users' productivity because they would no longer have to wait for the integration to complete.

The following considerations can help you evaluate how to bring forward existing integrations:

- **Is the integration still needed?** It's not uncommon to find out that nobody uses the results of the integration anymore and it can be retired.
- **Are there connectivity challenges if the modernized application is in the cloud?** Challenges could include latency as well as access to an on-premises API or data store. In some cases, the on-premises data gateway can help provide the ability to access the service or data from the cloud. Where the access to the data or service is too slow, consider whether you can make the data local to the modernized app or perform the integration in the background.

Integration can also help right-size a modernized application. You can partition one or more parts of the legacy application to leave behind or implement in a separate application. This approach would work well when users of different roles use different parts of the legacy application. You could implement one or more of the roles using low-code, and use process integration to let the existing application handle the remaining parts of the process. Using this approach, you could incrementally modernize the remaining parts over time. Having independent parts of the process implemented separately can also facilitate a more agile way of rolling out enhancements independently of the other parts of the process.

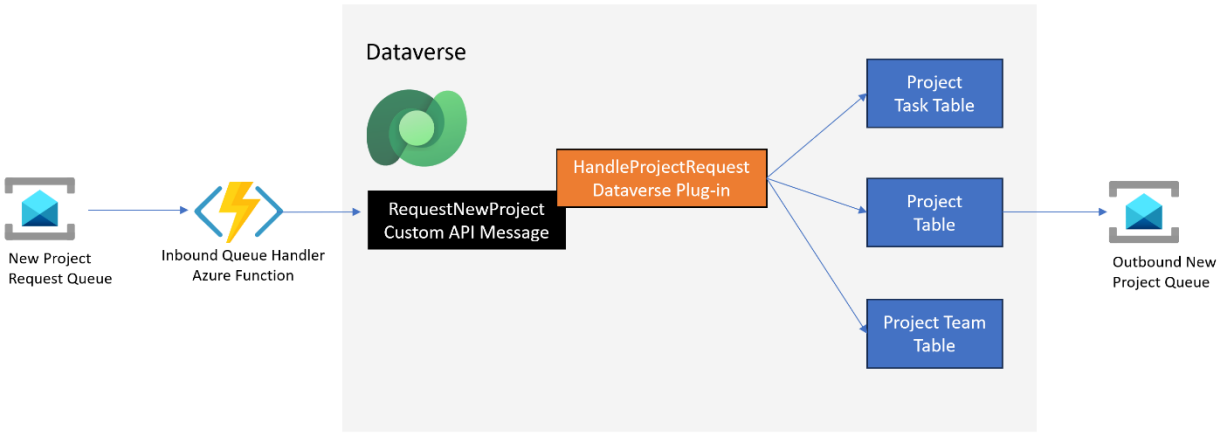
Before carrying forward any custom integrations, you should evaluate the integration capabilities built into Power Apps.

- **Microsoft Teams:** Power Apps canvas apps can be embedded in Teams channels. Using the Teams connector, apps and flows can easily post and consume Teams messages. Power Apps cards can be used like micro-apps to share actionable information in a Teams channel.
- **SharePoint:** Power Apps model-driven apps can be configured to connect to documents stored in a SharePoint library to make them available in a Dataverse row. Microsoft Lists or a SharePoint list, users can run Power Automate flows in the context of a list item.
- **Power BI:** Power BI insights can be shown in the context of a Power Apps canvas app. You can embed a model-driven app in a Power BI report to allow users to act on the insights without leaving Power BI.

Using Dataverse as the primary data repository for the modernized application provides a few built-in capabilities that can be helpful for integration.

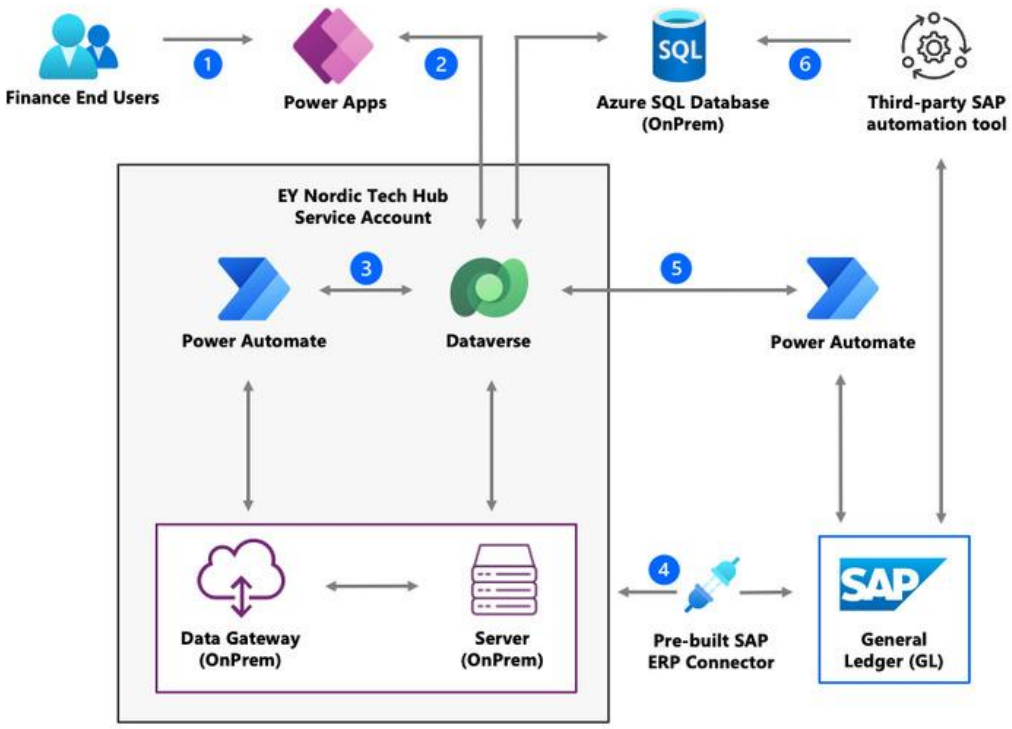
- Dataverse custom APIs can be used for inbound application-level integration. Custom APIs provide a unique operation that's associated with a small amount of custom code logic. For example, a sending system could use the `RequestNewProject` custom API and the associated logic would know how to place the received data into the appropriate Dataverse tables. The sending system would be abstracted from the Dataverse table structure.
- Outbound integration could be done using the publish events capabilities of Dataverse. Dataverse can be configured to publish to Azure Service Bus, Azure Event Hubs, or any Webhook receiver. For example, when a new Dataverse project table row is created, it could be published to an Azure Service Bus queue. You can also publish more conceptual events that match a business process event. For example, you could define and publish events when a project is completed.

The following diagram illustrates an example of inbound and outbound events in a Dataverse environment.



Organizations should also consider pre-built integration options available from third parties on Microsoft AppSource. For example, Microsoft has a pre-built solution for organizations that need to integrate SAP with the Power Platform. This pre-built solution incorporates apps and flows and adds new functionalities that facilitate communication between your organization's SAP system and Power Platform.

For example, Ernst & Young used the pre-built SAP integration to rapidly develop a solution to optimize a high-frequency global finance process. The following diagram of the company's PowerPost solution shows how finance users post documents to its GL SAP ERP system using Power Platform.



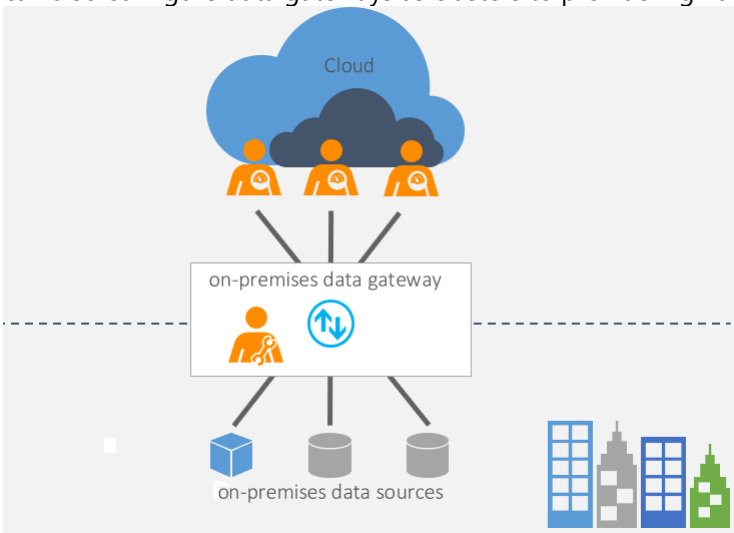
1. Finance end users log into PowerPost app enabled by Power Apps to perform General Ledger (GL) document posting. Data access is secured based on user type configurations within the app and Dataverse.
2. Submitted document entries are posted to Dataverse and are stored using relational data configurations to track the document status by header.
3. Once the necessary approvals are made in PowerPost, the document header status changes to trigger a Power Automate cloud flow enabled by the OnPrem Data Gateway.
4. When the document is ready for posting, Power Automate triggers the pre-built SAP ERP connector to pull data and post into EY's SAP ERP system.
5. Document postings will either be accepted or rejected within SAP which triggers a Power Automate return output to Dataverse. Rejected postings will notify the finance submitter to correct the identified error.
6. Third-party SAP automation tool periodically pushes necessary SAP data to Azure SQL Database (OnPrem). Scheduled Microsoft Dataflows query and import SAP data into Dataverse.

Integration connectivity options

As solutions move to the cloud, connectivity back to on-premises resources can be essential in ensuring that integrations still work with the modernized application. These applications must also be able to integrate with other traditional cloud resources that might reside in different network environments. The

Power Platform supports the four primary options for secure connectivity: data gateways, virtual network data gateways, private links, and ExpressRoute.

- **Data gateways** allow low-code components from Power Apps, Power Automate, and Power BI to reach back to on-premises resources to support hybrid integration scenarios. Gateways provide a quick way for modernized low-code applications to access data sources that are still on-premises. With a gateway, you can connect to on-premises data from sources like a local file system, DB2, Oracle, SAP ERP, SQL Server, and SharePoint. One gateway can support multiple users and access to multiple sources. You can also configure data gateways as clusters to provide high availability.



  Gateway support is integrated into the connection process for connectors, allowing indication when a gateway is required and selection of the configured gateway. After the connection is configured, apps and flows can use the connector just like one without a gateway.

- **Virtual network data gateways** allow Power BI and Power Platform dataflows to connect to data services in an Azure virtual network without the need of an on-premises data gateway on a virtual machine inside the virtual network.

- **Azure private link** and Azure networking private endpoints allows apps and flows to access Power BI securely. Private endpoints are used to send data traffic privately using Microsoft's backbone network infrastructure instead of going across the internet. Private endpoints ensure that traffic going into your organization's Power BI resources, such as reports or workspaces, always follow your organization's configured private link network path.

- **Azure ExpressRoute** provides an advanced way to connect your on-premises network to Microsoft cloud services using private connectivity. A single ExpressRoute connection can access multiple online services like Power Platform, Dynamics 365, Microsoft 365, and Azure cloud services without traversing the public Internet. ExpressRoute requires significant planning and configuration and involves additional cost for the ExpressRoute service and the connectivity provider.

Regardless of which approaches you use for integration connectivity, you should evaluate your connectivity to ensure it has low enough latency and enough bandwidth to support both your integrations and the modernized application.

Business Logic

When building modern applications, you can choose what to implement business logic with and where to implement it in your application architecture. Without guidance, most organizations would end up with

business logic chaos. Having reusable logic that ensures consistency in implementation can help speed up modernization efforts.

For our purposes, we'll define business logic as being different from user experience logic. For example, logic to navigate from screen to screen based on inspection data values is user experience logic. The logic you implement to determine if an inspection is complete might include several condition evaluations and would be considered business logic.

When architecting a solution that includes low-code, the following considerations can help you decide where you might place business logic.

- **In the Power Apps application:** Placing business logic in the low-code application is the simplest approach, but it provides limited options for reuse or to enforce consistency across apps and automations. Generally, you should limit this approach to non–mission-critical, simple logic that other applications or automations don't need to use. The low-code tooling doesn't provide a line-by-line debug experience. If logic spans more than one screen or is hard to read, you should consider other approaches that would be more maintainable. It's not uncommon for some business logic to be duplicated locally in the application as well as in the cloud. For example, if a user is entering a hotel booking, the business rule is that the check-out date cannot be before the check-in date. If the application didn't validate this, the user would get all the way to the end and submit the booking only to find the custom connector rejected it. Handling validation locally in the application as well as in the cloud delivers a far better user experience.
- **In a Power Automate cloud flow:** You can express business logic in the actions in a flow, and the flow can trigger in response to an event or a run-on-demand request from other apps and flows. The flow can provide a low-code approach to centralizing logic. Steps in the flow are independent and aren't part of a transaction; however, flows can implement compensation to handle rollback if errors occur. Flows can run steps using connections that have permissions beyond what the app user might be able to do, providing a way to elevate permissions. This approach also allows for minimizing the permissions the app user might require.
- **In a Dataverse plug-in:** Plug-ins run in response to a data row event like create, update, or delete. This logic will run any time the event occurs, regardless of which app or flow performed the action or if it was done directly from the Dataverse API. The benefit of this behavior is that it ensures consistency across all uses. Additionally, all the Dataverse data changes from the plug-in logic are transactional and either all complete or all roll back. Plug-in logic must be short and efficient and not try to implement long-running work. Sometimes plug-ins on events aren't the best approach if you must listen to events on multiple tables to complete a single business event like Close Inspection. For example, you might consider a Dataverse custom API instead of having plug-ins on multiple tables. Plug-ins can perform logic with elevated permissions that the user might not normally have. This approach also allows for minimizing the permissions the app user might require. Plug-ins can deploy in a Dataverse solution alongside apps and flows.
- **In Dataverse custom APIs:** Dataverse custom APIs allow you to implement your own custom API message that can run logic. For example, you could create a Close Inspection custom API that's called to do all the work to check and close an inspection. It wouldn't be event-driven, but used on demand by the apps and flows that need it. Like event-driven plug-ins, the data changes done in the custom API plug-in are transactional. A custom API is best when the only service it uses is the Dataverse API for other data work. Plug-ins for custom APIs can deploy in a Dataverse solution alongside apps and flows.

Implement a code API

You can implement APIs in your favorite API hosting runtime, such as Azure Functions, Azure Container Apps, or any service that's capable of hosting a REST API. These custom APIs can implement any logic, and both low-code and traditional code applications can use them. Custom APIs don't provide any transaction support other than what might be provided by an API they use. For example, a custom API could use SQL Server transaction constructs if it used SQL Server. Deployment of a code API would be independent of the low-code resources that might use it. You can use Azure API Management to govern the use of these APIs and help make them more discoverable.

Security

Security, including authentication and authorization, is an essential part of the architecture of a modernized application. Modern applications are often more challenging to secure than legacy apps. They include multiple cloud services, and users work with them from more diverse locations. Conceptually, security in the platform is there to ensure that users can do the work they need to do with the least amount of friction while still protecting data and services.

Power Platform takes a multiple-layer approach to security that you can use to build your security architecture. A core tenet of these capabilities is that low-code solutions should integrate with your existing security apparatus to minimize the impact of introducing them.

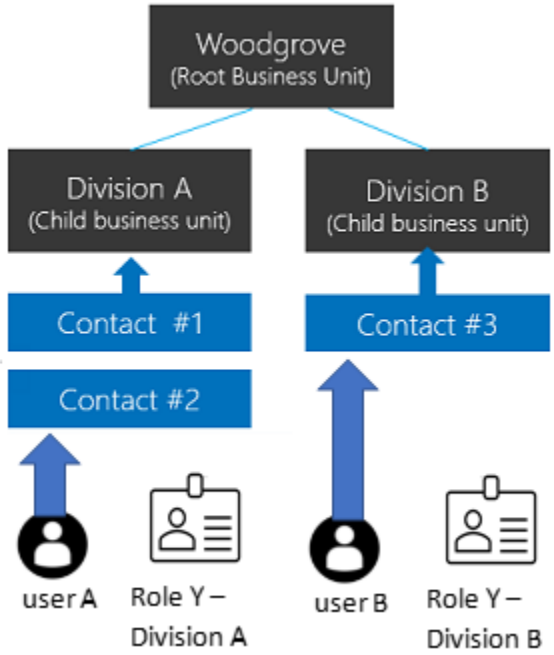
Let's take a high-level look at the multiple layers of security that make up the Power Platform security model.

- Users are authenticated by Azure AD, and use can be restricted using conditional access policies.
- Licensing is the first control-gate to allowing access to Power Platform components.
- The ability to create applications and workflows is controlled by security roles in the context of environments.
- Users' ability to see and use Power Platform resources is controlled by sharing the application with them. Resources are shared directly with the user or Azure AD group.
- Environments act as security boundaries, allowing different security needs to be implemented in each one.
- Power Automate flows and canvas apps use connectors. The specific connection credentials and associated service entitlements determine permissions when apps use the connectors.
- Environments with a Dataverse instance support more advanced security models that are specific to controlling access to data and services in that Dataverse instance.
- Connector use can be further restricted with data loss prevention (DLP) policies. Cross-tenant inbound and outbound restrictions can also be applied to the connectors.

It's important to note that when accessing data sources using connectors, all the underlying security that the data source offers is in addition to the layers of security just described. Power Apps and Power Automate don't by default provide users with access to the connector data source that they don't already have. Users should only have access to data that they genuinely require access to.

When you use Dataverse as part of a solution, it includes a role-based security model that can be adapted to many business scenarios. Data can be secured all the way down to an individual column on a data row. Users are assigned one or more security roles that together determine their overall privileges. Dataverse provides security modeling building blocks like business units and teams. For example, business units can be used to define security boundaries to keep data isolated between two different groups of organization users. You can use teams to group users who need similar access to data. You can even assign group

ownership of data rows. The following diagram illustrates using business units to isolate data for divisions of an organization.



Design your security model

Tailor the security model of the modernized application for the application's overall architecture. Applications that use a single data repository and no connectors require minimal security design work. As applications use more connectors and data repositories, your security modeling must include other considerations.

- **User identity:** How do users authenticate and has that already been mapped into Azure AD in scenarios coming from on-premises? This includes mapping of groups necessary to support application group or team assignment in cloud data stores like Dataverse.
- **Connector connection identity:** When applications use one or more connectors, what type of authentication is done for the connection, and does it provide the level of control necessary to implement the required security controls? For example, applications that use a service principal to connect don't require the user of the application to have direct access to the connector, which can be beneficial in some scenarios. Individual user connections can be appropriate for applications that need to know which user performed an operation or to scope responses to specific users.
- **Security construct portability:** As your applications use more connectors and data repositories, it's important to remember that not all security constructs of one map directly to another. For example, in Dataverse there are multiple ways a user can get access to a data row, including simply sharing the row with the user. If an application associates a SharePoint document library with the row, the security that gives access to the document library is separate from the security that controls Dataverse access. They don't map directly. Modernized applications must accommodate these types of mismatches across the connectors and data repositories they use.

Artificial intelligence

Over the last few years, AI has found its way into application modernization efforts. When modernizing applications, organizations should consider using artificial intelligence to help make users more productive and better able to make informed decisions. The results of infusing AI can also translate into better customer experiences that positively affect business outcomes.

Including AI used to involve heavy lifting in integrating application logic and building custom-trained models. With the availability and power of large language models, applications can now introduce new ways of using AI to help users get answers and perform tasks. Users can use natural language prompts to interact with AI capabilities in a broad range of assistive business scenarios.

Microsoft has introduced Copilots in core products and services to make accessing advanced AI technology easier. A Copilot uses modern AI techniques and large language models and can be interacted with by users in the applications they use every day, like Microsoft 365, Windows, Dynamics 365, and Power Platform.

Extend with plugins

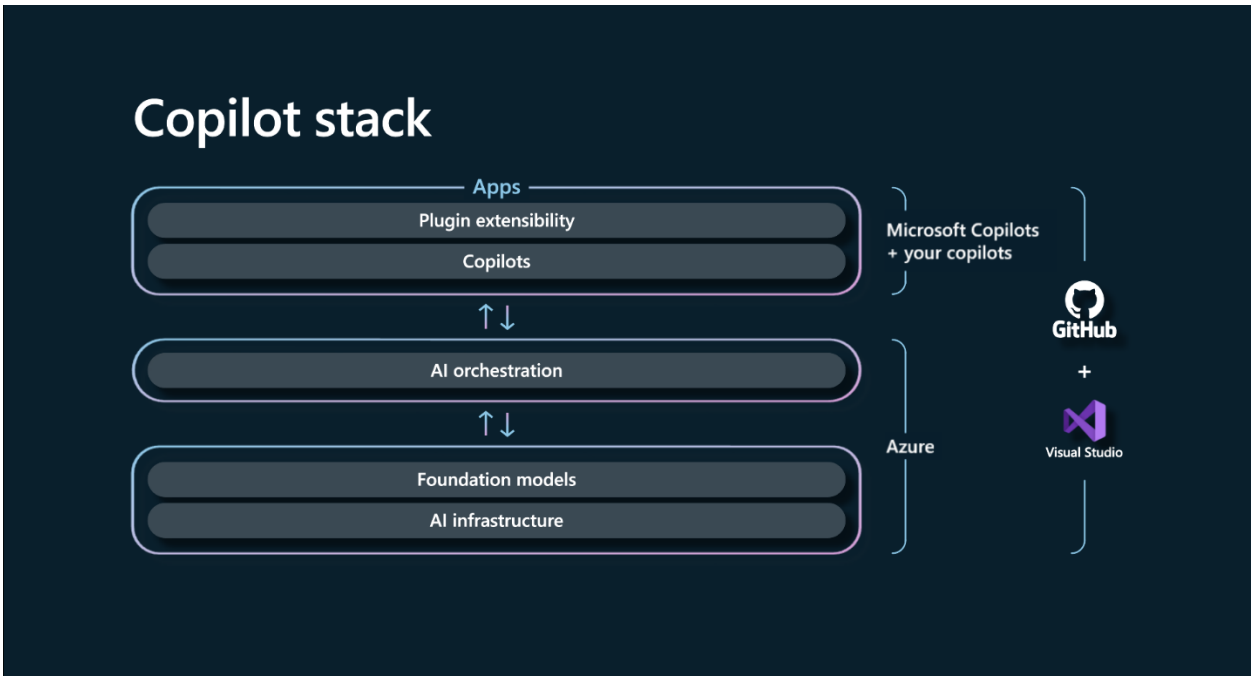
Users of Copilot-powered applications can ask the Copilot for help with common tasks in the application. You can extend Copilots to include data and tasks they don't already know and that are beyond the scope of the app the user is working with. The Microsoft 365 Copilot can incorporate Power Platform data stored in Dataverse, so that users don’t have to switch back and forth between applications. For example, from Outlook, a user can ask Copilot to generate a status update for all failed inspections completed today. Microsoft 365 Copilot automatically inherits the native security and governance framework of Dataverse and applies user security and permissions at runtime.

Connectors as plug-ins

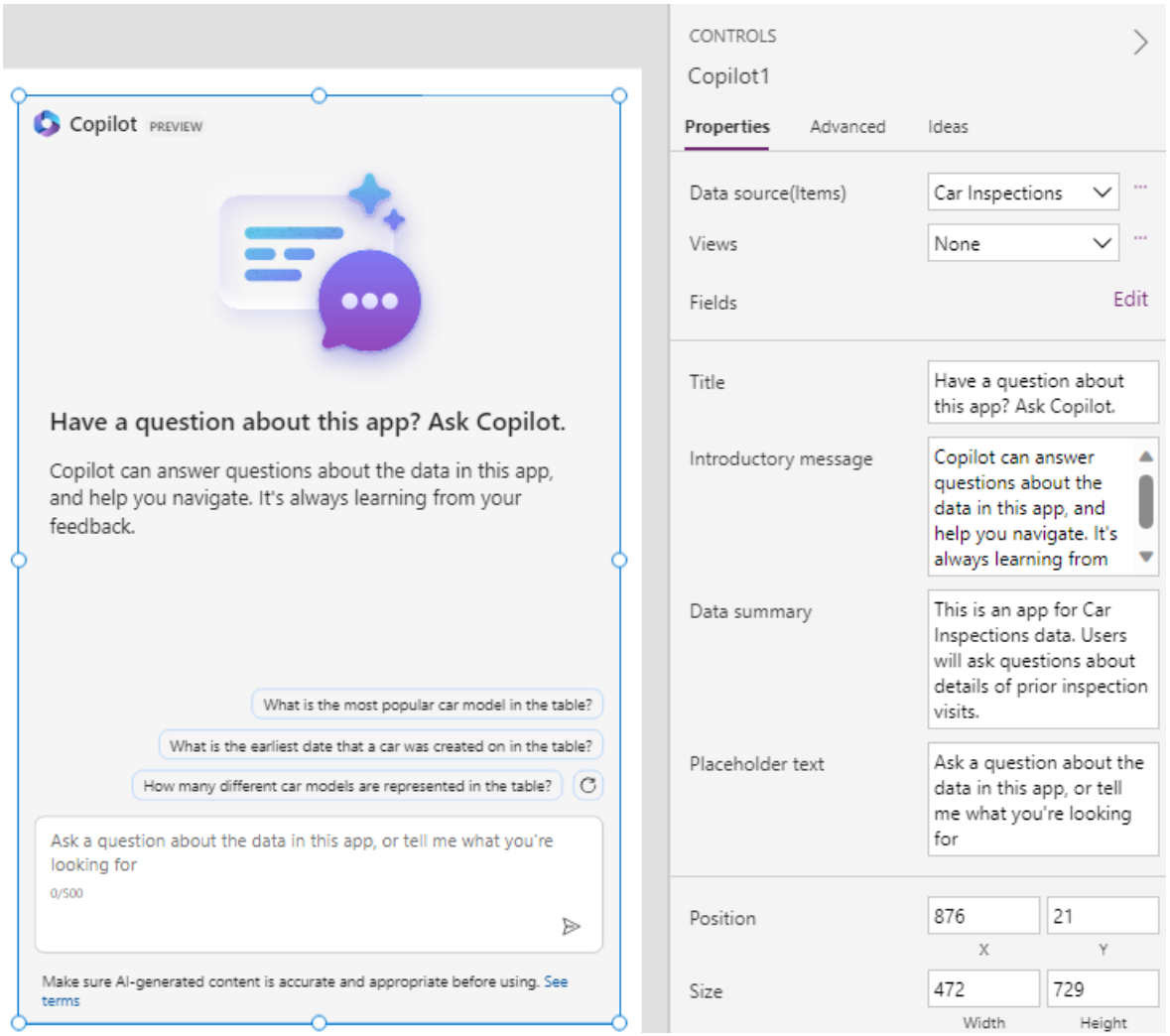
Power Platform connectors are also important to the Copilot experience. Connectors will be able to connect as plug-ins to extend Copilot’s capabilities. For example, Microsoft 365 Copilot with the Power Platform connector for Jira Software can enable a project manager to request the status of a Jira support ticket and act based on the response, such as routing it for additional approval or starting a purchase order for new hardware. Using plug-ins, you can integrate your business processes and data with Copilot to empower users to interact from whatever apps they use.

Build your own copilot

As users become more accustomed to having copilot AI assistance in their apps, they’ll expect it in all applications. You’ll be able to make your modern applications more engaging by including copilots that you create with Copilot stack, an AI development framework.



You can use the pre-built Copilot control in Power Apps to add copilots to your canvas apps and model-driven apps. By configuring a view of a data source and some basic prompt information, you can quickly provide an in-app copilot experience of your own.



Application lifecycle management

An important part of any modernization effort is establishing an appropriate application lifecycle management process. Organizations often want their low-code efforts to fit in with how they work with traditional code ALM. Power Platform provides ALM tools so that you can include low-code artifacts in or alongside the processes you typically use.

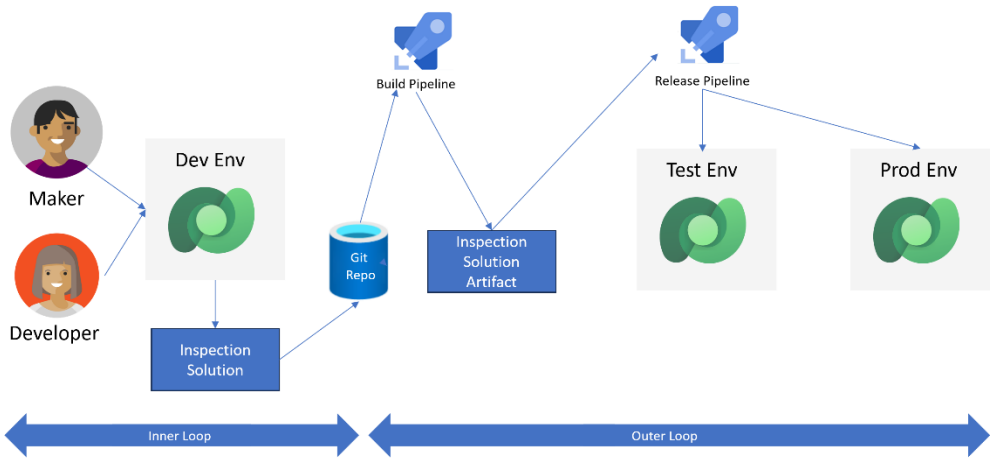
ALM in the Power Platform starts with how you build your low-code resources. Resources you build are in the context of a Power Platform environment. An environment can have one Dataverse data store. You can use multiple environments—typically dev, test, and production—to implement landing zones in an ALM process that includes low-code. The number and purpose of environments are flexible, and organizations can adjust them to meet individual project needs. A Dataverse solution is a container for related low-code resources, facilitating version control and transport from one environment to another.

Power pipelines provide a low-code approach to automate deployments and implement continuous integration and continuous delivery (CI/CD). Power Platform manages the process when pipelines are configured. Admins can centrally manage and govern the pipelines.

Organizations can also use the CI/CD tooling of their choice. Power Platform CLI is a command-line tool that you can use with most CI/CD automation tooling. Power Platform Build Tools provides actions for

GitHub and tasks for Azure DevOps that provide all the common actions required to build CI/CD automations that include low-code artifacts.

The following diagram illustrates an example of a team building an Inspection app. In their inner loop, they work in a dev environment and store their work in a Git repository. The outer loop consists of a test environment and a production environment. A build pipeline takes the version-controlled solution, performs any necessary checks, and produces an Inspection solution artifact. A release pipeline then deploys the solution to test, where testers can verify it's ready for production. Once approved, the release pipeline deploys the solution to production.



When you export a solution from Dataverse, it's exported as a single compressed file. To store the low-code resources in version control, you could use the build tooling to unpack the solution file into individual component files. During build automation, the build tools combine individual files from version control into a single compressed file.

Deployment of a solution into an environment that contains a previous version of the solution uses an intelligent upgrade process that only applies changes. This upgrade process avoids the need for differencing scripts or other ways of determining what needs to deploy.

When your modernized application includes low-code and traditional code resources, you can combine them in a single CI/CD process or manage them independently. With independent management, resources can be deployed individually, and project teams can achieve greater agility. For example, the API that a low-code application uses could deploy independently if the team doesn't introduce breaking changes.

Monitoring and insights

Modernized applications need to integrate into operational environments that provide the ability to diagnose issues across different environments, from development to production. Application Insights, an Azure Monitor extension, collects telemetry from Power Apps and Dataverse. This information not only helps with identifying and resolving issues, but also provides insights into what users do in an application. You can use these insights to help improve the apps and processes in your modernized application.

While a Power Apps application is in development, developers can include logic to log custom events. After you connect the deployed application to Application Insights, the extension automatically collects basic telemetry, including additional context from logged events, as users interact with the application.

Administrators can also configure Dataverse environments to export telemetry to Application Insights. Data that's logged can include Dataverse API calls, plug-in execution, SDK operations, and exceptions. Developers creating custom plug-in logic can log additional custom telemetry data directly to Application Insights.

Using Application Insights across your applications can make it easier to correlate issues with multiple resources. Operations staff can create alerts in Azure Monitor to trigger when high numbers of exceptions are detected. Regular analysis of your modernized applications can identify trends that require additional investigation.

Conclusion

In this whitepaper, we explored the benefits, strategies, and best practices of modernizing your legacy applications with Microsoft Power Platform. You gained insights and guidance on employing Power Platform's low-code capabilities to ensure the success of your modernization efforts as part of your organization's digital transformation.

Legacy applications present many challenges for organizations. To overcome them, organizations must embark on application modernization initiatives to revitalize their infrastructure and take advantage of modern technologies. In this whitepaper, you saw how to take a low-code approach to your modernization efforts—specifically, how the low-code development capabilities of Microsoft Power Platform allow you to quickly build and deploy modern applications.

Low-code opens the door to a broader set of people than just traditional coders. A key factor of organizations that succeed with a low-code approach is making sure the people involved in modernizing applications have training in low-code development, whether they're pro developers or business users. Business users are closer to the business problem being solved and can contribute time-saving expertise. Traditional code developers can apply the techniques and skilling they already have to build extensions to fill any low-code gaps. Organizations can be most effective by blending business and technical resources on what we like to call "fusion teams."

This whitepaper establishes the foundation for you. The next steps are yours. Here are some suggestions:

- Take a few minutes to find out what your organization is already doing with low-code. It might surprise you!
- Evaluate your application modernization opportunities.
- Identify and prioritize a good first candidate.
- Staff a team that will modernize the application. For the best results, make sure it's a fusion team.
- Make sure the team has the necessary training to be successful.
- Allow the team to modernize the application.
- Reflect on the modernization effort. Refine and scale it to other legacy applications.

Every organization's journey to application modernization is unique. Your Microsoft account team or Power Platform partner can help you plan your journey and keep you on the right path along the way.

Resources

The Total Economic Impact™ of Microsoft Power Platform Premium Capabilities

American Airlines ConnectMe app creates a smoother travel experience for customers and better technology tools for team members

Power Fx open-sourced repository on GitHub

CoE Starter Kit

Power Platform Adoption Assessment

Digital insurance agency automates a complex purchasing process using Power Platform

PCF Gallery

EY helps enable entry at source for a global finance process with Power Platform, reducing lead times by 95 percent

Azure private Link

Microsoft Azure ExpressRoute

Power Platform Release Planner

Microsoft Power Platform Licensing Guide

Microsoft outlines framework for building AI apps and copilots; expands AI plugin ecosystem