

# Power Platform Expert Bootcamp – Day 1

**Htoo Shwe Yee**  
Cloud Solution Architect

# Conditions and terms of use

© Microsoft Corporation. All rights reserved.

You may use these training materials solely for your personal internal reference and non-commercial purposes. You may not distribute, transmit, resell or otherwise make these training materials available to any other person or party without express permission from Microsoft Corporation. URL's or other internet web site references in the training materials may change without notice. Unless otherwise noted, any companies, organization, domain names, email addresses, people, place and events depicted in the training materials are for illustration only and are fictitious. No real association is intended or inferred. THESE TRAINING MATERIALS ARE PROVIDED "AS IS"; MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED IN THESE TRAINING MATERIALS. 1

# Session Schedule (Day 1)

Time	Activity Title
09 00-09 15hrs	Opening Introductions
09 15-10 15hrs	Solution Architecture
10 15-10 30hrs	Break
10 30-12 00hrs	Advanced Canvas Apps
12 00-13 00hrs	Lunch
13 00-15 30hrs	Hands-On Lab (Build the Invoice Intake App)

# Session Schedule (Day 2)

Time	Activity Title
09 00-10 15hrs	Advanced Power Automate
10 15-10 30hrs	Break
10 30-12 00hrs	Hands-on Troubleshooting Flow
12 00-13 00hrs	Lunch
13 00-15 30hrs	Hands On Lab Invoice Approval Flow

# Session Schedule (Day 3)

Time	Activity Title
09 00-10 15hrs	ALM/DevOps
10 15-10 30hrs	Break
10 30-12 00hrs	Power BI Integration & Solution Documentation
12 00-13 00hrs	Lunch
13 00-15 30hrs	Hands On Lab Invoice Approval Power BI dashboard

# Self Introduction

## Who are you?

- Name, role
- What have you developed so far using PowerApp, Power Automate or PowerBi?
- Expectations for this workshop



# Agenda Overview

- Analyzing user requirements
- Component Selection and Project Planning
- Data Modeling and Security in Power Platform
- Designing scalable navigation & app structure
- Error Handling in Canvas App
- Complex Power Fx formulas
- Building and Using Components

# Analyzing User Requirements



# End-to-End Solution Design and Stakeholder Engagement

## Role of Solution Architect

The Solution Architect leads design, aligning Power Platform solutions with business and technical needs.

## Project Phase Involvement

Solution Architects engage in discovery, analysis, design, implementation, and quality assurance phases.

## Responsibilities and Deliverables

They make key design decisions and produce documentation covering scalability, security, and integration.

## Guiding Principles

Focus on business goals, design for maintainability, and prefer configuration over custom code.



# Understanding the User Requirements

## **Capturing Functional Requirements**

Functional requirements define what the system must do, gathered through workshops and user interviews.

## **Identifying Non-Functional Requirements**

Non-functional requirements specify system qualities like performance, security, and usability constraints.

## **Conducting Fit-Gap Analysis**

Fit-gap analysis compares requirements with standard features to identify gaps needing custom solutions.

## **Adopt Before Adapt**

Encourage using out-of-the-box functions to cover requirements and reduce customizations.

## **Justify Customizations**

Only customize when significant business needs exist to avoid complexity and longer development time.

## **Process Catalog Alignment**

Align business processes with platform capabilities to identify fits and gaps across workflows.

## **Feasibility and Prioritization**

Assess gap feasibility and prioritize requirements to manage development effort efficiently.

# Component Selection and Project Planning

# Selecting Power Platform Components: Apps, Automation, BI, AI, and Integration

## First-Party vs Custom Apps

- Leverage existing Power Platform tools if closely aligned or build custom Power Apps for unique requirements.

## Power Apps Form Factor Choice

- Choose Canvas, Model-driven, or Portal apps based on user type and access needs, including internal or external users.

## Automation vs Application

- Decide if background automation suffices or if a full user interface app is necessary for the solution.

## BI and AI Integration

- Incorporate Power BI for analytics and choose AI Builder, Copilot Studio or Azure AI for intelligent capabilities based on complexity.

## External Integration and Extensibility

- Use Azure Logic Apps, Functions, or custom connectors for integration; develop custom PCF components for advanced UI needs.

## Licensing and Future Planning

- Consider licensing costs and scalability to ensure the solution fits budget and future enterprise-wide expansion.

# Mapping Requirements to Solution Components and Design Decisions

## Defining Solution Components

Identify Power Platform components and external integrations with their specific roles in the solution.

## Component Interactions

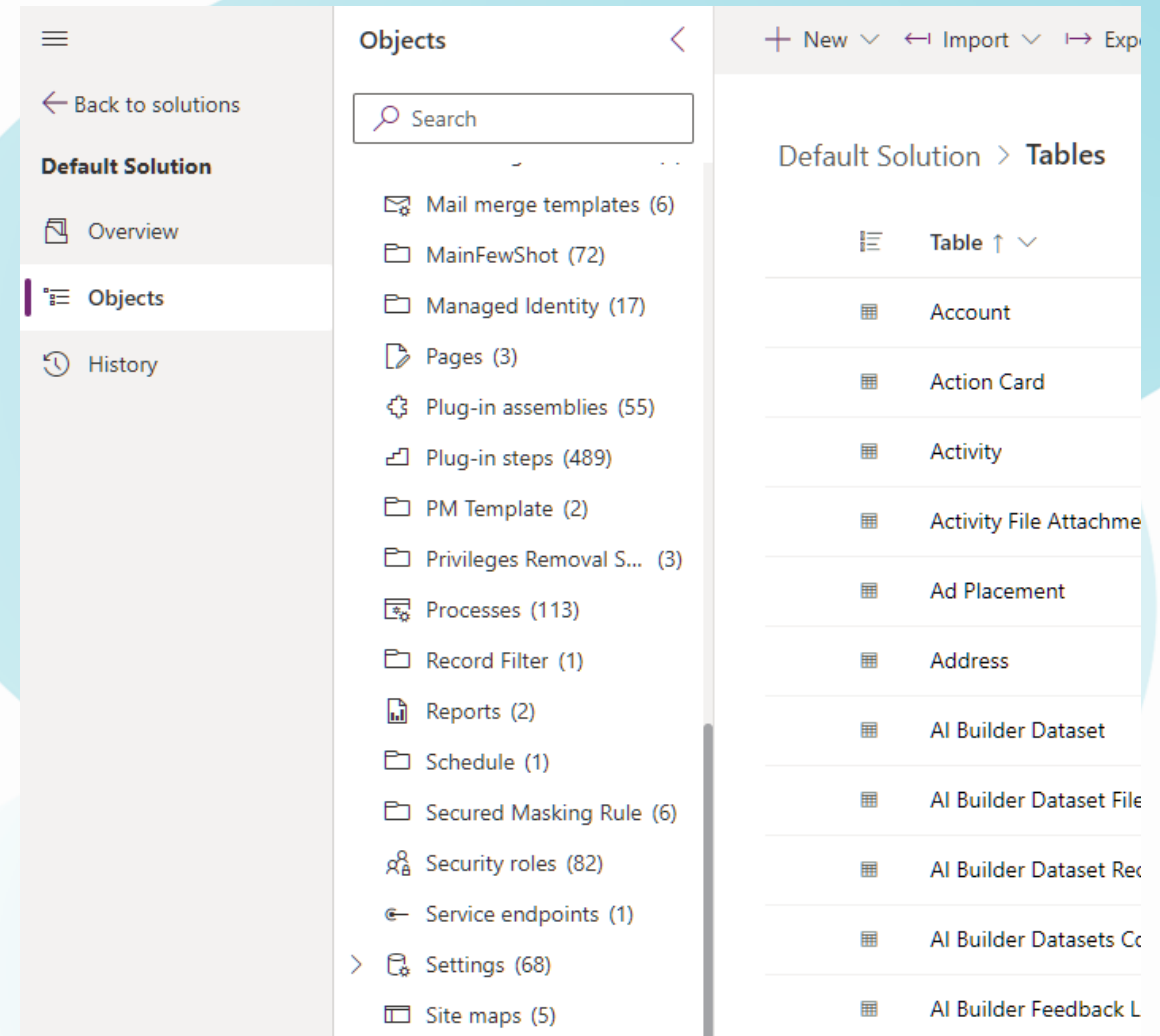
Show how components interact, highlighting data flow and integration points using diagrams or swimlanes.

## Mapping Requirements to Design

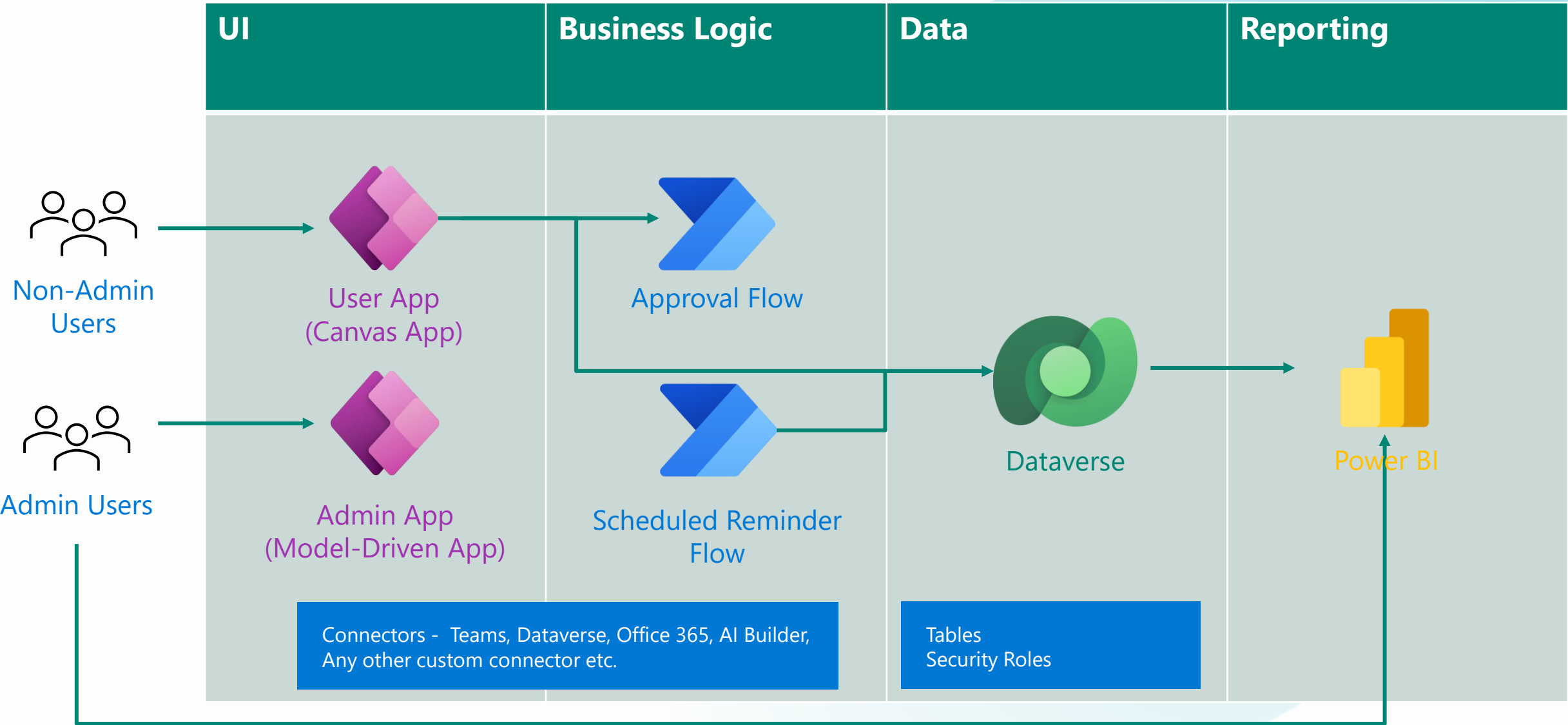
Ensure every requirement is addressed by specific components within a layered conceptual architecture model.

## Key Design Decisions

Document critical design choices such as integration methods and environment setups for future clarity.



# Sample App Components Architecture



# Deciding between Model-driven App and Canvas App

---

## Model-Driven Apps

---

- Data-first approach; UI generated from Dataverse.
  - Best for structured, complex Dataverse data.
  - Uses predefined templates with limited customization.
  - Single-page structure with guided navigation.
  - Faster to build but may need more user training.
  - Built-in responsiveness and accessibility.
- 

## Canvas Apps

---

- UI-first approach; UI designed freely before connecting data.
  - Works with many data sources.
  - Full design flexibility on a blank canvas.
  - Multi-page, highly flexible layouts.
  - More effort and time required to design and build.
  - Must be designed manually for responsiveness and accessibility.
-

# Data Modeling and Security in Power Platform



# Designing Data Models in Dataverse: Entities, Relationships, and Field Types

## Entity Identification

Identify core business objects as entities with relevant columns based on solution requirements.

## Normalization and Relationships

Normalize data to reduce redundancy using 1:N, N:1, and N:N relationships between tables.

## Field Types and Choices

Select appropriate Dataverse field types like Text, Lookup, Choice, and Date based on data needs.

## Naming and Documentation

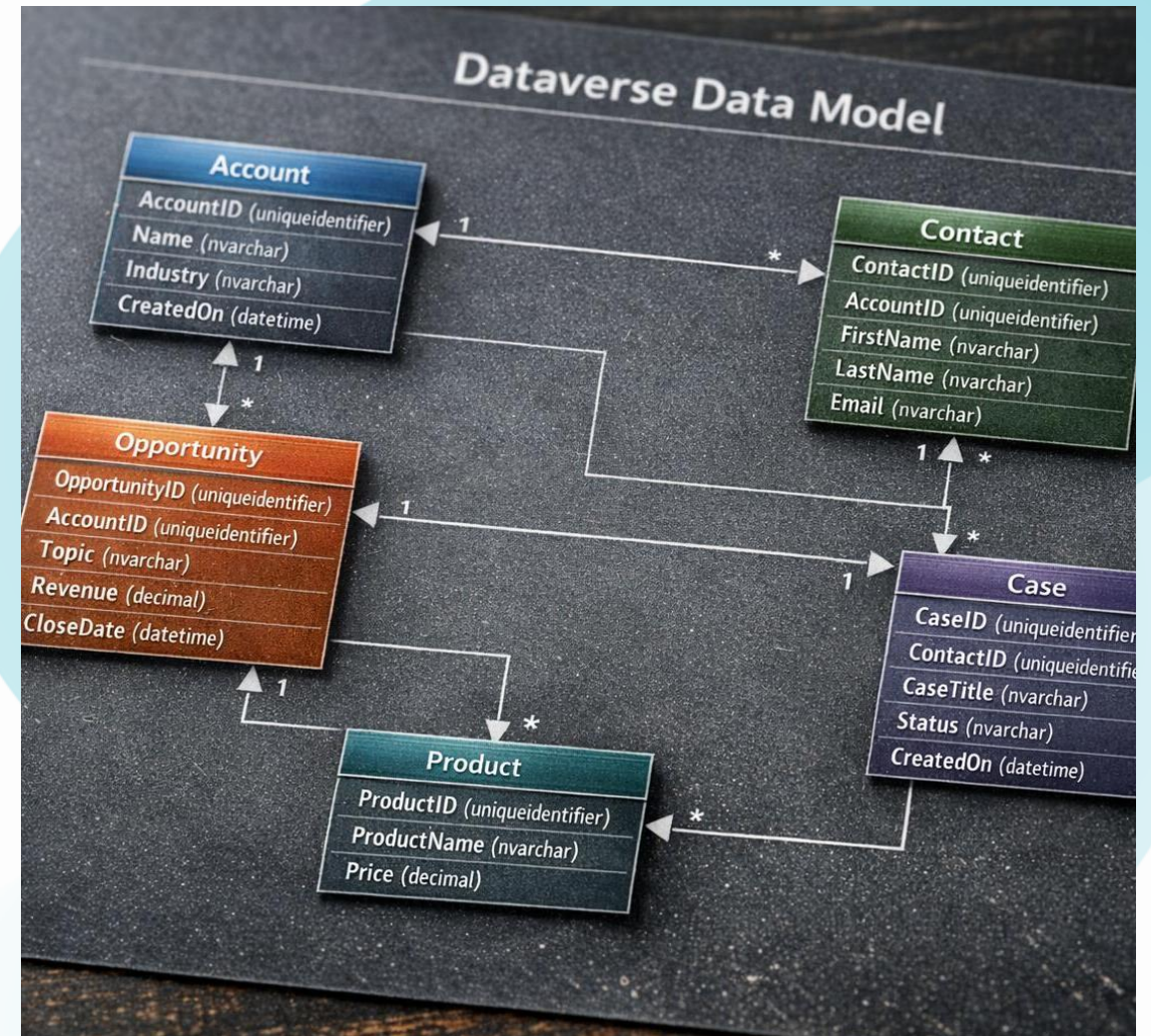
Clear, consistent naming and thorough documentation improve understanding and maintenance.

## Iterate with Sample Data

Populating sample records reveals gaps and validates the data model design effectively.

## Business Rules and Performance

Use business rules, calculated and rollup fields to enforce logic; optimize for performance and delegation.



# Implementing Security: Environment, Roles, Teams, and Field-Level Security

## Environment-Level Security

Power Platform environments use roles like Admin, Maker, and User to control app creation and access permissions.

## Dataverse Roles and Business Units

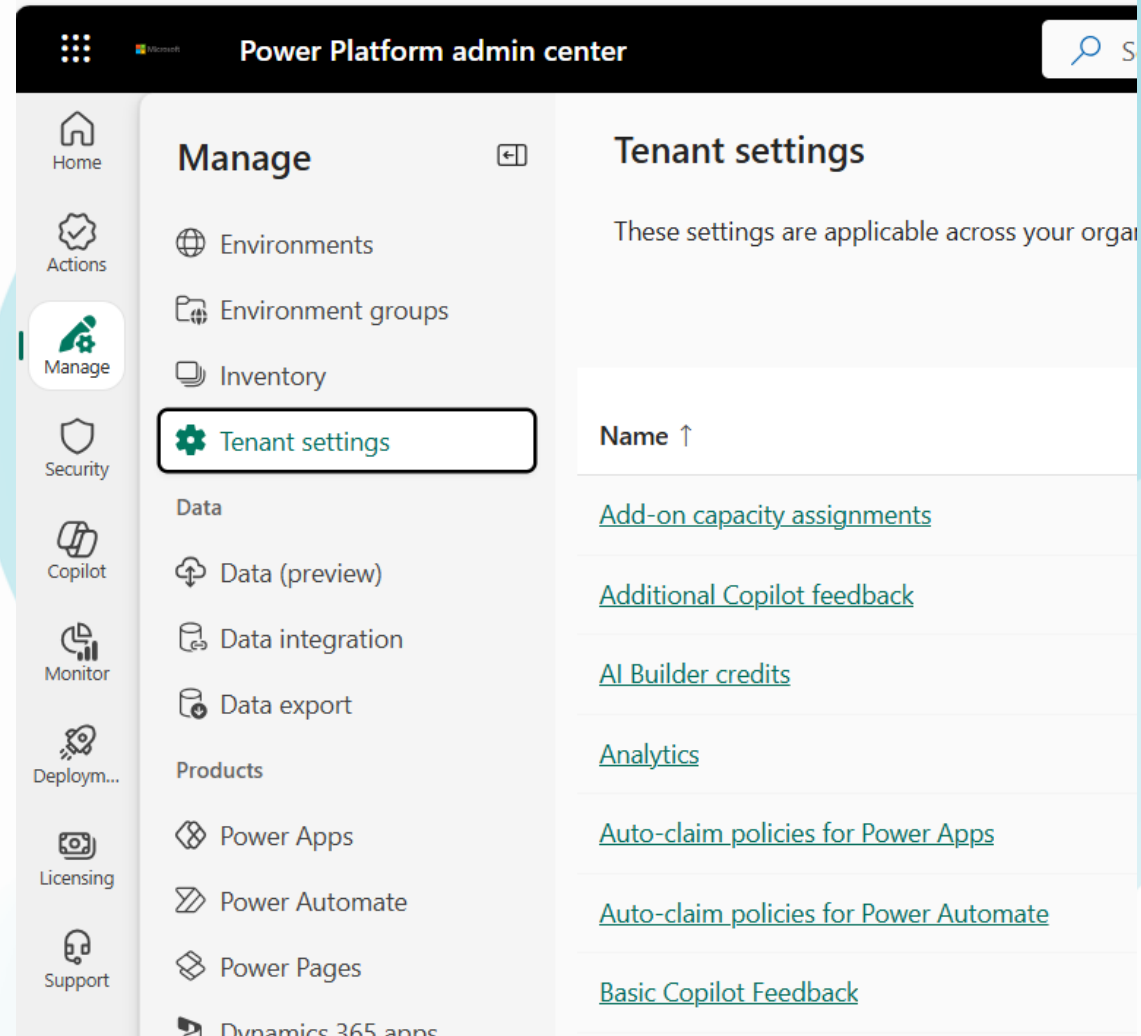
Dataverse roles define permissions with scopes, while business units segregate data for organizational hierarchy and security.

## Teams and Record Ownership

Teams, mapped to Azure AD groups, own records to facilitate collaboration and access management in Dataverse.

## Field-Level Security and Governance

Sensitive fields are protected by field security profiles, with governance covering compliance, DLP, and audit logging.





# Security Best Practices

## Least Privilege Principle

Grant users only the minimum access necessary to perform their tasks to enhance security and reduce risks.

## Business Unit Segmentation

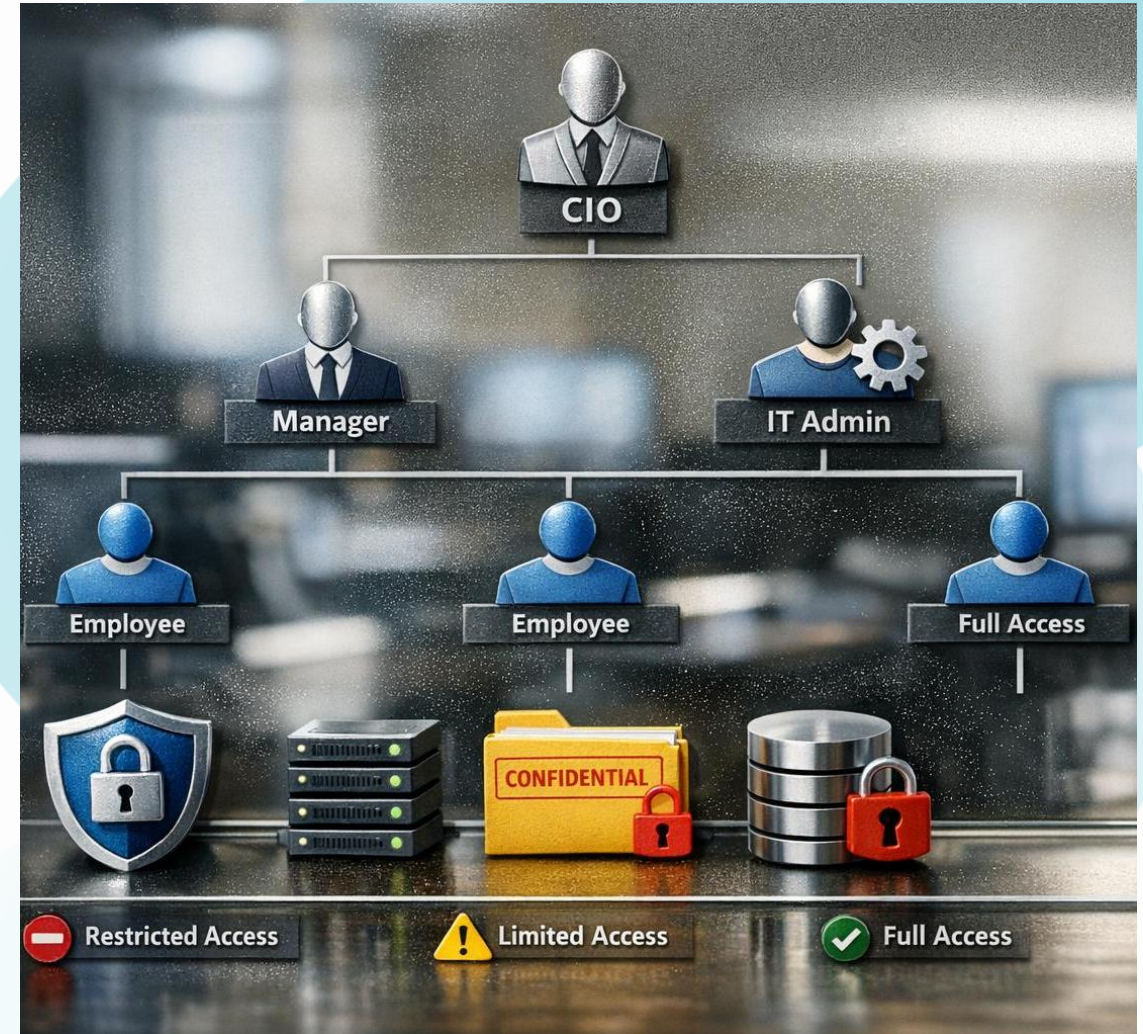
Plan business unit hierarchy early to segregate data and control access across departments and environments.

## Default Security Roles

Use default security roles as templates and customize them to fit job functions and data access needs.

## Data Loss Prevention Policies

Implement DLP policies to restrict data flow between business and non-business connectors, preventing leaks.



# Designing scalable navigation & app structure

# Screen Planning and Navigation Patterns

## Logical Screen Planning

Design screens around logical steps with focused content for easier use and maintenance.

## Smooth Navigation Patterns

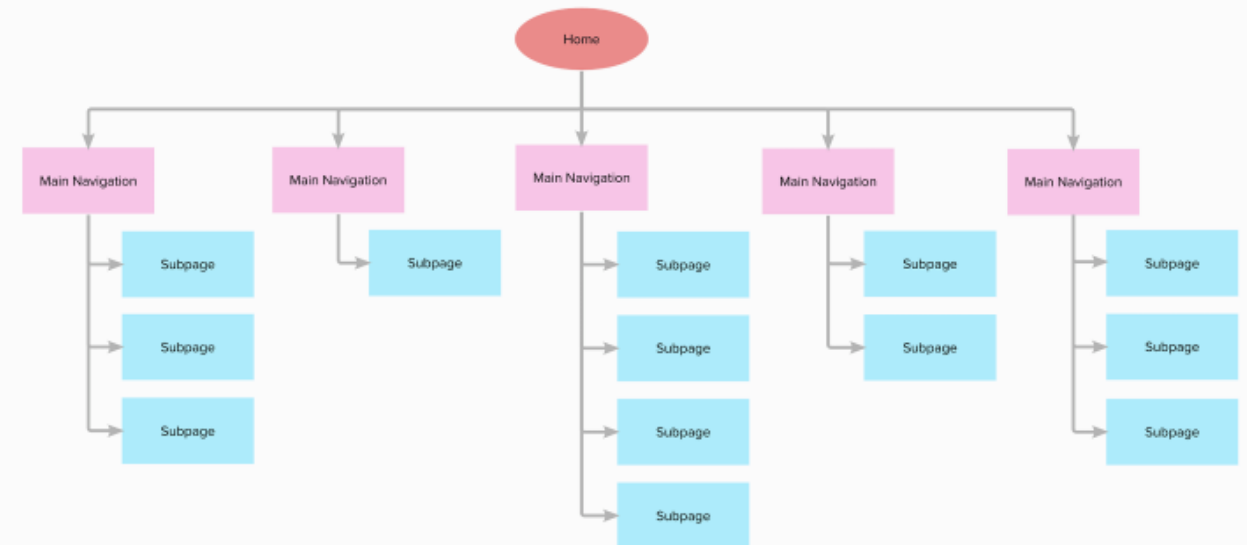
Use consistent navigation bars, back buttons, and step indicators for user-friendly flow.

## Consistent Design & Components

Maintain consistent controls and use components for uniform navigation across screens.

## Performance and Deep Linking

Load data intelligently and enable deep linking for direct access to screens with parameters, for e.g. sending a link to view the request directly in app





# User Experience, Feedback, and Common Mistakes

## User Journey Feedback

Clear guidance on start, progress indicators during processes, and success messages improve user experience.

## Error Handling and Empty States

Graceful error messages and empty state notifications prevent user confusion during data issues.

## Responsive Design for Devices

Design adapts for phones with vertical stacking and larger buttons; tablets use multi-column layouts.

## Common UX Mistakes

Avoid information overload, unclear buttons, missing confirmations, and poor layout balance for better usability.



# Error Handling in Canvas App

# Error Handling Techniques in Power Apps

Use **IfError()** Function to wrap risky operations and provide alternate actions or friendly error messages.

*IfError( Expression, ValueIfError [, FallbackValue ] )*

Example Below:

`IfError( 10 / 0, "Error: Cannot divide by zero" )`



# Error Handling Techniques in Power Apps

Use **IsError()** or **IsBlankOrError()** Function to detect errors from data sources or values for conditional handling.

## *IsError( Expression)*

Examples Below:

### During Data Retrieval

```
With(
    { ExistingRecord: LookUp('App Devices Assets', ID = 123) },
    If(
        IsError(ExistingRecord),
        Notify("Error: Unable to fetch the record", NotificationType.Error),
        If(
            IsBlank(ExistingRecord),
            Notify("No record found", NotificationType.Information),
            Notify("Record fetched successfully", NotificationType.Success)
        )
    )
)
```

### During Running a Flow

```
If(
    IsError( GetAllCustomerInvoices.Run("C0001023") ),

    // On failure
    Notify( "Error: could not retrieve customer invoices",
    NotificationType.Error ),

    // On success
    Navigate('Success Screen')
)
```

# Error Handling Techniques in Power Apps

Use **OnFailure** and **OnSuccess** properties in forms to provide user feedback after submission attempts.

## ***OnSelect property of the form's submit button***

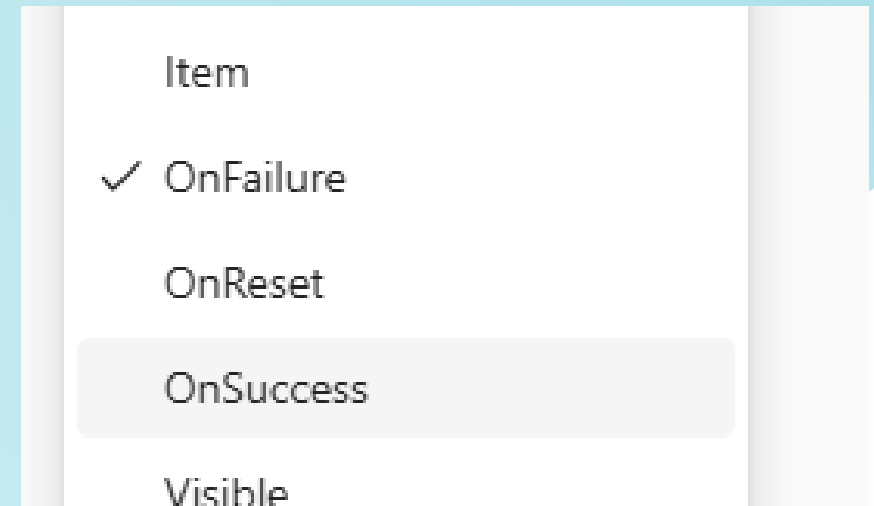
`SubmitForm(frm_Invoice);`

## ***OnSuccess property of the form***

`Navigate('Success Screen');`

## ***OnFailure property of the form***

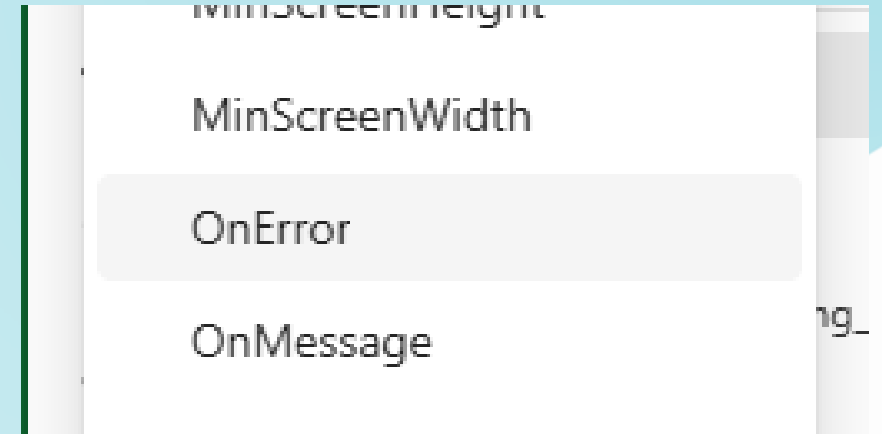
`Notify( "Error: the invoice could not be created", NotificationType.Error );`



# Error Handling Techniques in Power Apps

Use **App.OnError** for global error handling and Notify function to inform users with friendly messages.

```
Notify(  
    Concatenate(  
        "Error: ", FirstError.Message,  
        "Kind: ", FirstError.Kind,  
        ", Observed: ", FirstError.Observed,  
        ", Source: ", FirstError.Source  
    ),  
    NotificationType.Information  
);
```



# Complex Power Fx formulas

# Using With() to Simplify Complex Logic

## When to use With()

- When a formula is too long or unreadable
- When you repeat the same expression multiple times
- When a calculation should happen only once
- When you want “inline variables” rather than global ones

### Before

```
If(
    TextInput_Quantity.Text * TextInput_Price.Text > 500,
    TextInput_Quantity.Text * TextInput_Price.Text * 0.9,
    TextInput_Quantity.Text * TextInput_Price.Text
)
```

### After

```
With(
    {
        total: Value(TextInput_Quantity.Text) *
Value(TextInput_Price.Text)
    },
    If(
        total > 500,
        total * 0.9,
        total
    )
)
```

# Using Named Formulas (App.Formulas)

## Why use Named Formulas?

- Always stay up-to-date (no manual Reset or Update)
- Reduce duplicate logic across screens
- Great for reusable expressions like:
  - URLs
  - Permission checks
  - User context
  - Role logic
  - Current environment variables
  - Business rules

### Examples

Named Formula

*CurrentUser = User()*

Use it anywhere in the app

*CurrentUser.FullName*

Named Formula

*StandardTaxRate = 0.08*  
*PreferredCustomerDiscount = 0.15*

Use it anywhere in the app

*TotalAmount \* (1 + StandardTaxRate)*

Named Formula

*IsAdminUser = "Admin" in VarUserRoles*

Use it anywhere in the app

*IsAdminUser*

# Modular patterns (1)

Modular patterns break big formulas into **small, reusable components**.

## Pattern 1: Input → Process → Output structure

```
With(  
    {  
        qty: Value(txtQty.Text),  
        price: Value(txtPrice.Text),  
        tax: 0.08  
    },  
    With(  
        {  
            subtotal: qty * price  
        },  
        subtotal + (subtotal * tax)  
    )  
)
```

## Pattern 2: Reusable component formulas

Inside a component:

```
With(  
    {  
        cleanedText: Trim(Self.InputText),  
        isValid: Len(Trim(Self.InputText)) > 0  
    },  
    {  
        CleanValue: cleanedText,  
        Valid: isValid  
    }  
)
```

Parent screen:

```
MyTextValidator.Valid
```

# Modular patterns (2)

Modular patterns break big formulas into **small, reusable components**.

## Pattern 3: Centralized business rules

In Named Formula:

HighValueThreshold = 10000

Parent screen:

```
If(
    ThisItem.Amount > HighValueThreshold,
    "Director Approval Required",
    "Manager Approval Required"
)
```

## Pattern 4: State + Behavior separation

State Layer (Named Formulas):

CurrentInvoiceAmount = Sum(gallInvoiceLines.AllItems,  
LineTotal)

IsLargeInvoice = CurrentInvoiceAmount >  
HighValueThreshold

UI Layer (Button.Visible):

IsLargeInvoice



# Simplifying and Structuring Formulas

## Choosing Appropriate Functions

Use built-in functions like Switch, ForAll, and Collect to simplify logic and improve formula efficiency.

## Maintaining Clean Formulas

Use comments and break complex logic into smaller parts to make formulas easier to maintain and debug.

## Testing and Debugging

Test formulas in isolation using temporary labels and debugging tools like Monitor for reliable app performance.

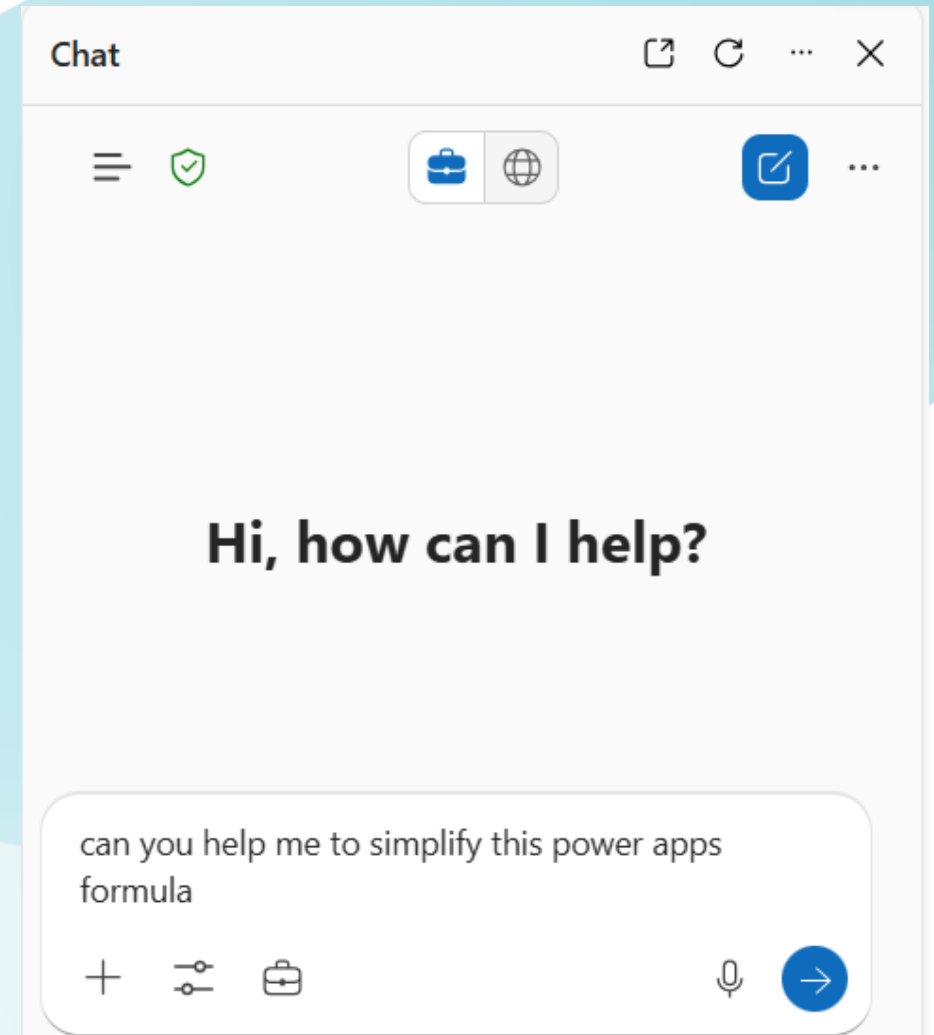
## Delegation in Power Apps

Power Apps delegates operations to data sources to improve performance, but some complex formulas are not delegable.

Aggregate functions like Sum, CountIf, and Average simplify data computations and are often delegable to data sources.

## Avoiding Common Pitfalls

Avoid nested ifs, repeated sub-expressions, and unclear operator precedence to reduce errors in app logic.



# Building and Using Components

# Reusable UI Components

## Reusable UI Components

Build custom controls once and reuse them across multiple screens or apps to ensure consistency and ease maintenance.

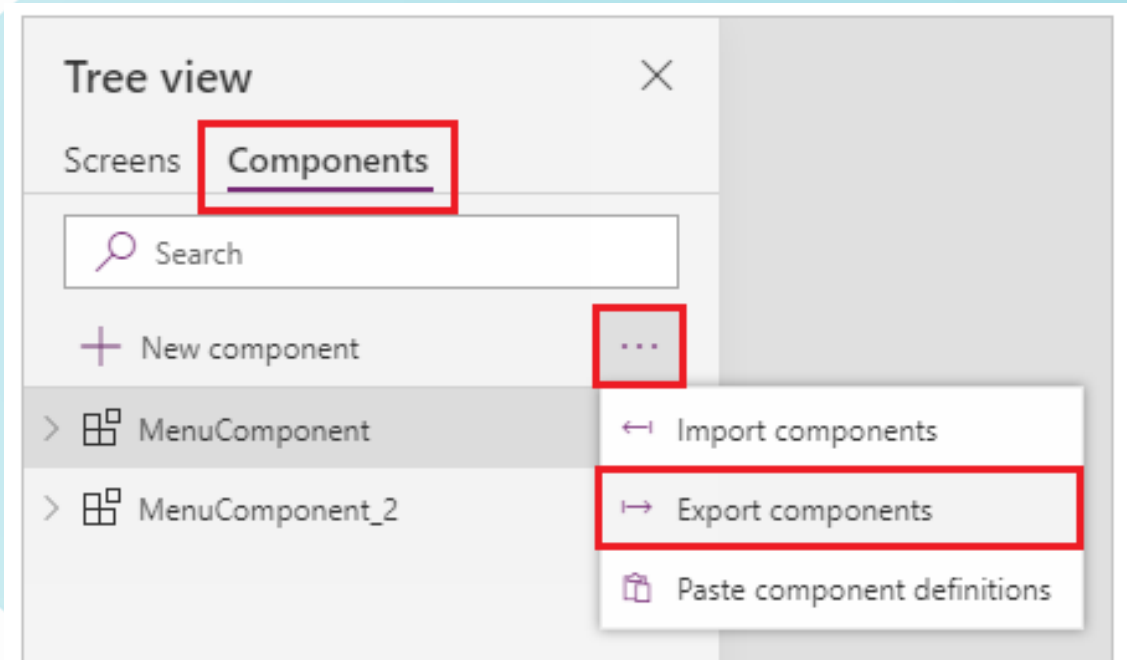
## Custom Input and Output Properties

Components support input properties for data and output properties for events, enabling dynamic interaction and navigation.

## Common Use Cases

Typical components include

- app headers,
- navigation menus,
- form sections,
- custom controls, and
- pop-up dialog boxes.



# Component Custom Properties

- **Input** Properties: Pass data into a component
- **Output** Properties: Return data or computed values
- Enable real component reusability and flexibility
- Used for dynamic behavior and data passing
- Different Property types for different purposes

The image shows two overlapping windows from the Axure RP software. The background window is the 'Properties' panel for a component named 'Component\_DateInput'. It has tabs for 'Display' and 'Advanced'. Under 'Display', there is a 'Description' field with 'No value', an 'Access app scope' toggle set to 'On', a 'Size' section with 'Width' set to 500 and 'Height' set to 'No value', and a 'Fill' color picker. The foreground window is a 'Property type' dropdown menu. It shows a list of property types: 'Data' (selected), 'Function', 'Event', and 'Action'. Each type has a brief description of its purpose.

**Properties** [X]

COMPONENT ?

**Component\_DateInput**

Display Advanced

Description: No value

Access app scope: On [Toggle]

Size: Width: 500, Height: No value

Fill: [Color Picker]

**Custom properties** ⓘ

+ New custom property

Editable	Boolean	...
Label	Text	...
ScreenSize	Number	...
Cal Input Default Da...	Text	...
Cal Input Hint Text	Text	...
Cal Input Current D...	Output property	
Reset	Output property	

**Property type** ⓘ

Data [Dropdown Arrow]

**Data**  
This type of property can send or receive values between the app and the component.

**Function**  
This type of property can be called as a function with parameters.

**Event**  
This type of property allows you to create an event that the component can trigger, and then be handled by the app.

**Action**  
This type of property can be called as a function with parameters, and can contain logic that changes state (side effects).

# Component Libraries, Performance, and Collaboration

## Component Libraries Benefits

Component libraries enable reuse across multiple apps and centralized updates for consistent organization-wide components.

## Build Once, Update Everywhere

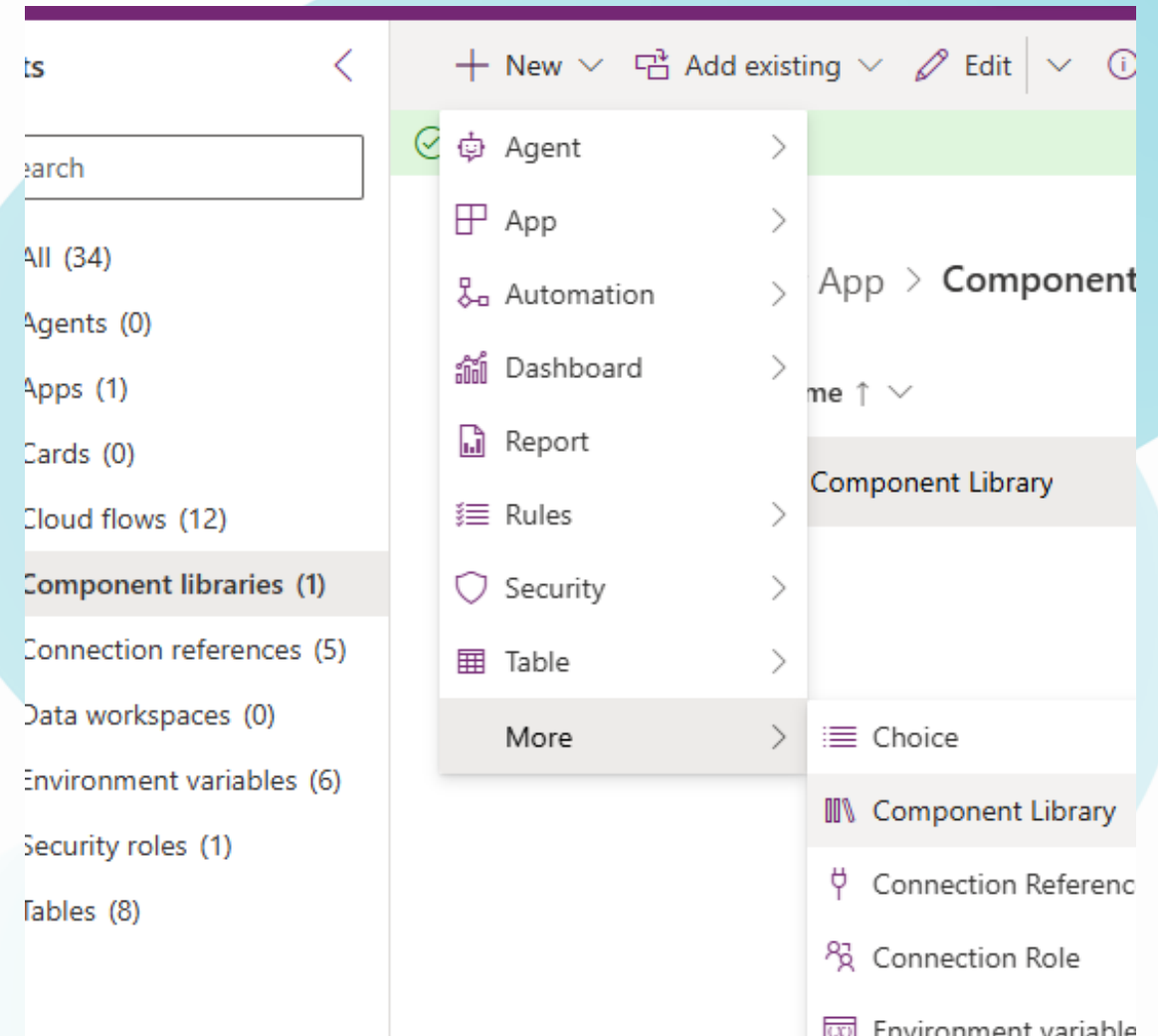
Editing a component updates all instances instantly, saving time and ensuring uniformity across app screens.

## Performance Enhancements

Components encapsulate complex logic and reduce control trees, improving app performance and maintainability.

## Collaboration and Best Practices

Components facilitate teamwork by modularizing work and require clear naming and regular updates for effective use. Define governance to avoid breaking changes



# Power Platform Expert Bootcamp – Day 2

# Session Schedule (Day 2)

Time	Activity Title
09 00-10 15hrs	Power Automate Architecture & Resilient Automation
10 15-10 30hrs	Break
10 30-12 00hrs	Hands-on Troubleshooting Flow
12 00-13 00hrs	Lunch
13 00-15 30hrs	Hands On Lab Invoice Approval Flow

# Agenda Overview

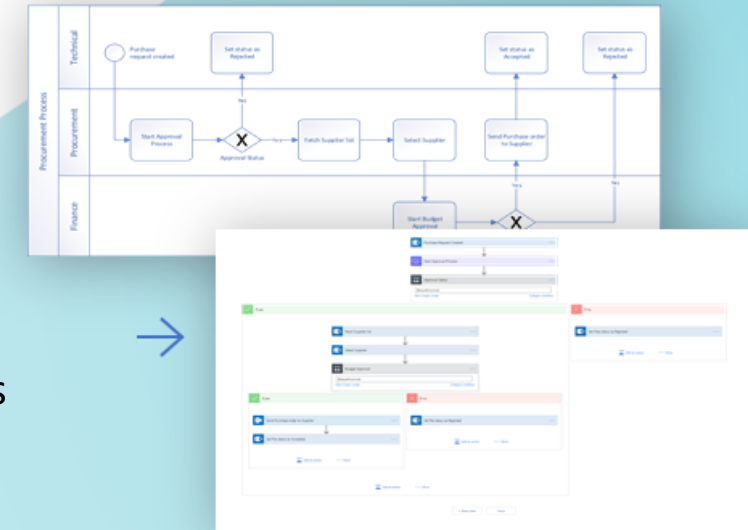
- Scenario Evaluation and Automation Patterns
- Modular Automation with Child Flows
- Error handling in Power Automate
- Advanced Power Automate Action Settings
- Interactive Experiences with Adaptive Cards



# Scenario Evaluation and Automation Patterns

# Common Automation Scenarios

- Automated approvals across data stores and services
- Recurring processing – e.g., expire all records over 7 days
- Process guidance – interactive step by step completion of common business processes
- Automated triggering from data or service events e.g., record or file created
- Mobile user clicks a button to kick off automation including context
- Power App app user clicks a button to launch a flow in the background



## Automation Suitability Checklist

Question	Why it matters
Is the process repeatable & rules-based?	Automation requires clear, deterministic steps
Does it involve high volume or frequency?	High ROI for automation
Are the systems involved API-friendly?	Determines cloud flow vs. RPA
Are there approvals, conditions, or branching?	Helps select the flow structure
Are humans still required for decisions?	Determines hybrid automation
Are compliance & audit logs required?	Influences flow type + environment

# Decision Matrix for Selecting an Automation Pattern

Scenario	Recommended Pattern	Why
<b>Trigger-based (record created/updated)</b>	Automated Flow	Best event-driven model
<b>Human decision required</b>	Approval Flow (Human in the loop)	Provides audit + Teams integration
<b>Complex business stages</b>	Business Process Flow	Structured step-by-step flow
<b>On-demand user action</b>	Instant/Button Flow	Manual execution by users
<b>Time-based recurring process</b>	Scheduled Flow	Runs automatically on schedule
<b>Legacy desktop app</b>	Desktop Flow (RPA)	Simulates user actions
<b>Large-scale data processing</b>	Automated or Scheduled + Parallelism	Best performance

## Additional Evaluation Criteria

### ✓ Complexity

- Use automated flows for simple rules; RPA only when absolutely necessary.

### ✓ Resilience

- For long-running workflows:
- Use Scopes + RunAfter error handling
- Use separate child flows
- Add retry policies

### ✓ Data Sensitivity

- Processes involving finance/HR → prefer Dataverse connectors + solution-aware flows.

### ✓ Performance

- If processing > 5,000 items:
- Use pagination
- Use parallel branches
- Prefer SQL/Dataverse over SharePoint

# Modular Automation with Child Flows

# Concept and Benefits of Child Flows

## Definition of Child Flows

Child flows are subroutines invoked by parent flows to break down complex processes into smaller reusable units.

## Benefits of Child Flows

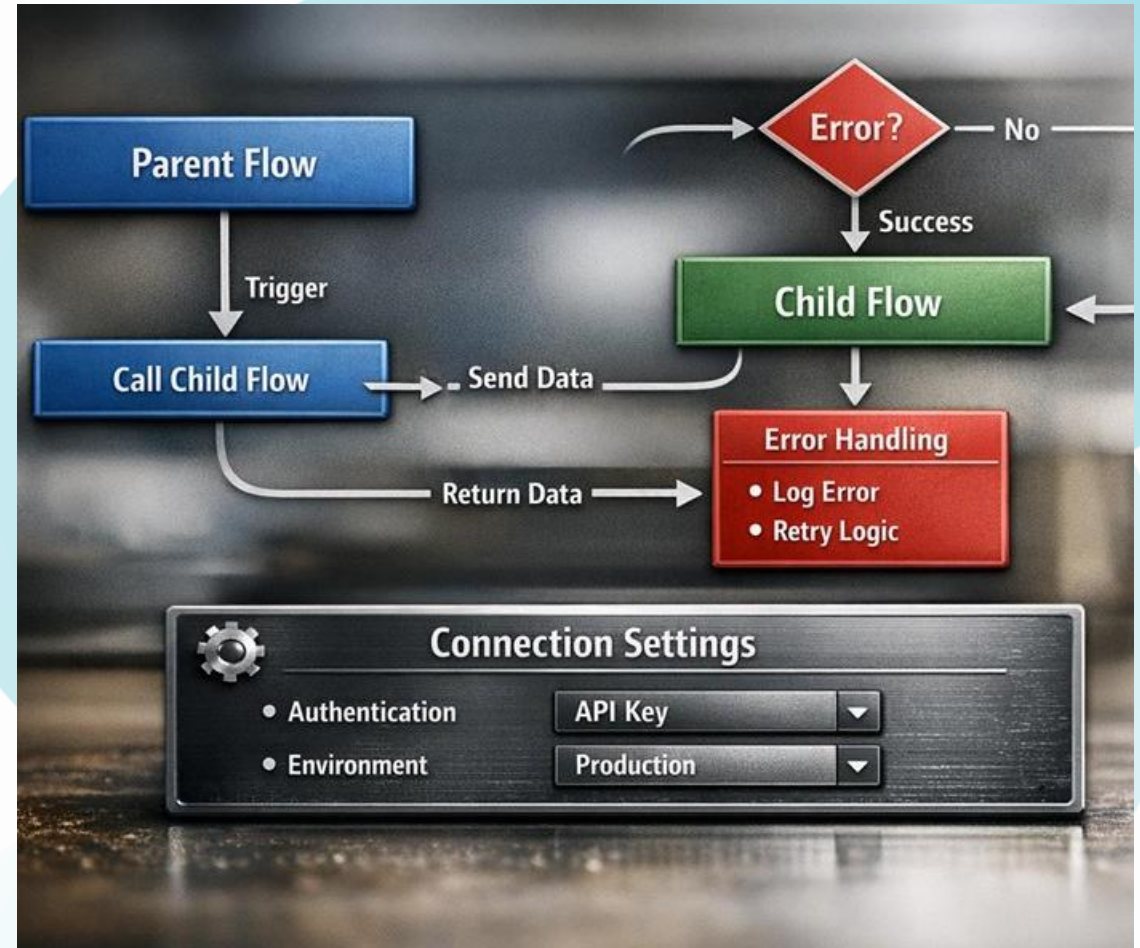
They reduce complexity, avoid duplication, and simplify maintenance by centralizing common process steps.

## Reusability Across Processes

Child flows can be used by multiple parent flows, creating a reusable library of process logic.

## Best Practices for Child Flows

Name child flows clearly, document inputs/outputs, and add conditions to prevent unintended standalone runs.



# Setup, Usage, and Error Handling

## Setup Requirements

Child flows must be inside solutions and triggered manually with defined inputs and outputs for communication.

## Connection Considerations

Child flows run with owner's connections; **configure run-only users** to avoid permission errors in parent flow.

## Execution Limits

Parent flow waits up to **30 days** for child completion; child flows are limited to cloud flows only with no nesting.

## Error Handling Strategies

Handle child flow failures **using run-after conditions** or by designing child to return success/failure status outputs.



# Environment Variables



# Environment Variables

Environment variables enable one-click deployment using correct configuration values for each environment, avoiding missing config errors.

## Best Practices Tips

Create variables for values changing by environment, set default and current values, and update during solution import.

## Common Mistakes to Avoid

Avoid hardcoding settings and remember to update env variables after import; use Dataverse-backed environments only.



**New environment variable** ✕

Environment variables can have different values when re-used, enter information about this variable so that future users can understand its purpose. [Learn more](#)

**Display name \***

Test

**Name \*** ⓘ

cr1bb\_ Test

**Description**

**Data Type \***

Choose an option ▾

- ✕ Decimal number
- { }** JSON
- Abc** Text
- ≡** Yes/No
- 📄** Data source
- 🔑** Secret



# Error handling in Power Automate

# Error Handling Strategies in Power Automate

## Configure Run After

Set actions to run after failure or skip to create alternative logic paths in flows.

## Scopes for Try/Catch

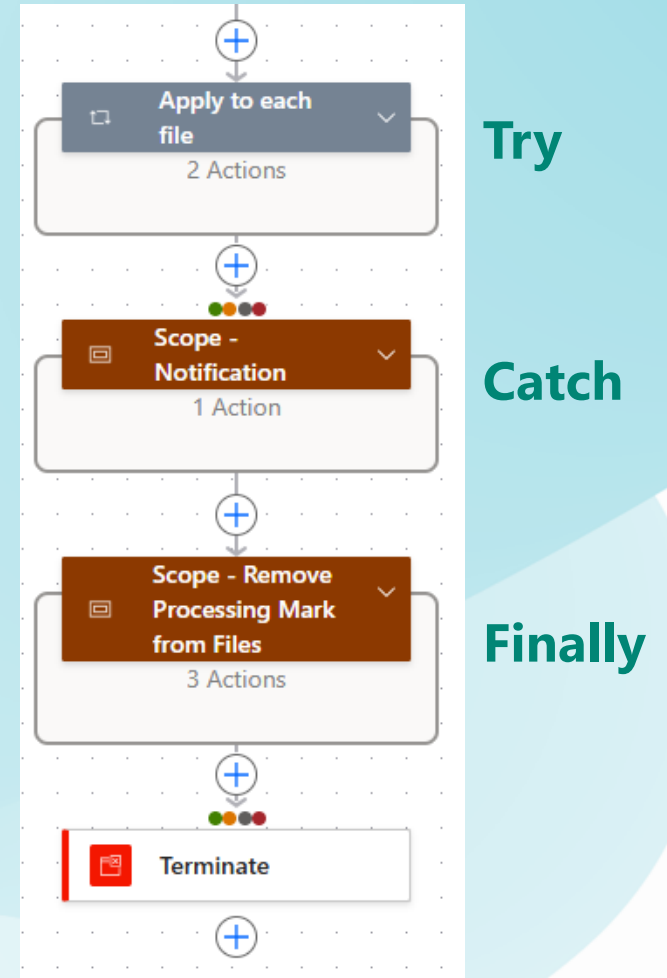
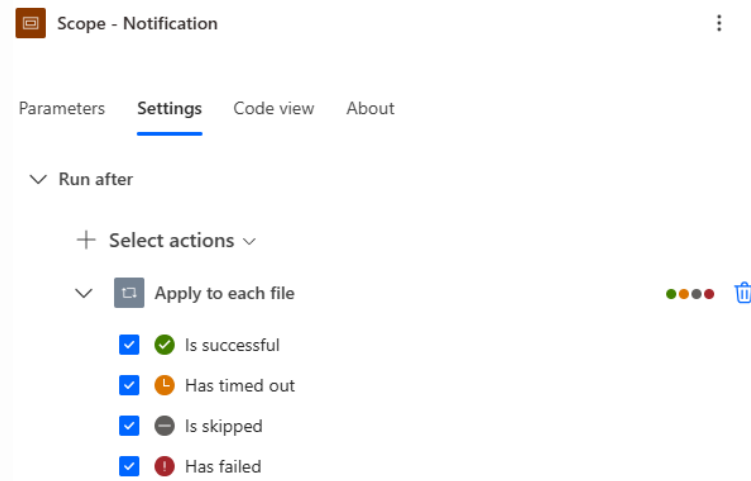
Use try and catch scopes to handle errors by grouping actions and executing catch logic on failures.

## Terminate Action Usage

Terminate flows on errors with a failure status and custom message to highlight issues clearly.

## Retry Policy and Monitoring

Adjust retry settings for transient errors and configure failure notifications for effective monitoring.



# Advanced Power Automate Action Settings

# Error Handling with Retries

## Retry Policy

A Retry Policy defines how an action or trigger retries a request when the original request times out or fails. The retry policy works with the HTTP action and handles the following HTTP status codes.

Automatic retries on failures help handle transient errors; configurable policies include none, fixed, or exponential delays based on action safety.

Sr No	Error Code	Error Meaning
1	408	Request Timeout
2	429	Too many requests
3	5xx	xx refers to any number like 500 – Internal server error, 503 – Service Unavailable, 522 – Connection timed out etc

**Get Uploaded Files**

Parameters **Settings** Code view Testing About

☒ On  
Threshold  
1000

Retry policy ⓘ  
A retry policy applies to intermittent failures, characterized as HTTP status codes 408, 429, and 5xx, in addition to any connectivity exceptions. The default is an exponential interval policy and the number of retries is based on your performance profile. [Learn more](#)

Default  
Default  
None  
Exponential interval  
Fixed interval

minimize previous refresh delete

# Pagination and Timeouts

## Pagination and Timeouts

Pagination fetches large datasets in chunks to improve flow performance.

## Timeouts

Timeout prevent long-running actions from hanging indefinitely. Duration in ISO-8601 format, for e.g.

- PT1H – 1 hour
- PT30S – 30 Seconds
- P1D – 1 day

## 30-Day Flow Duration Limit

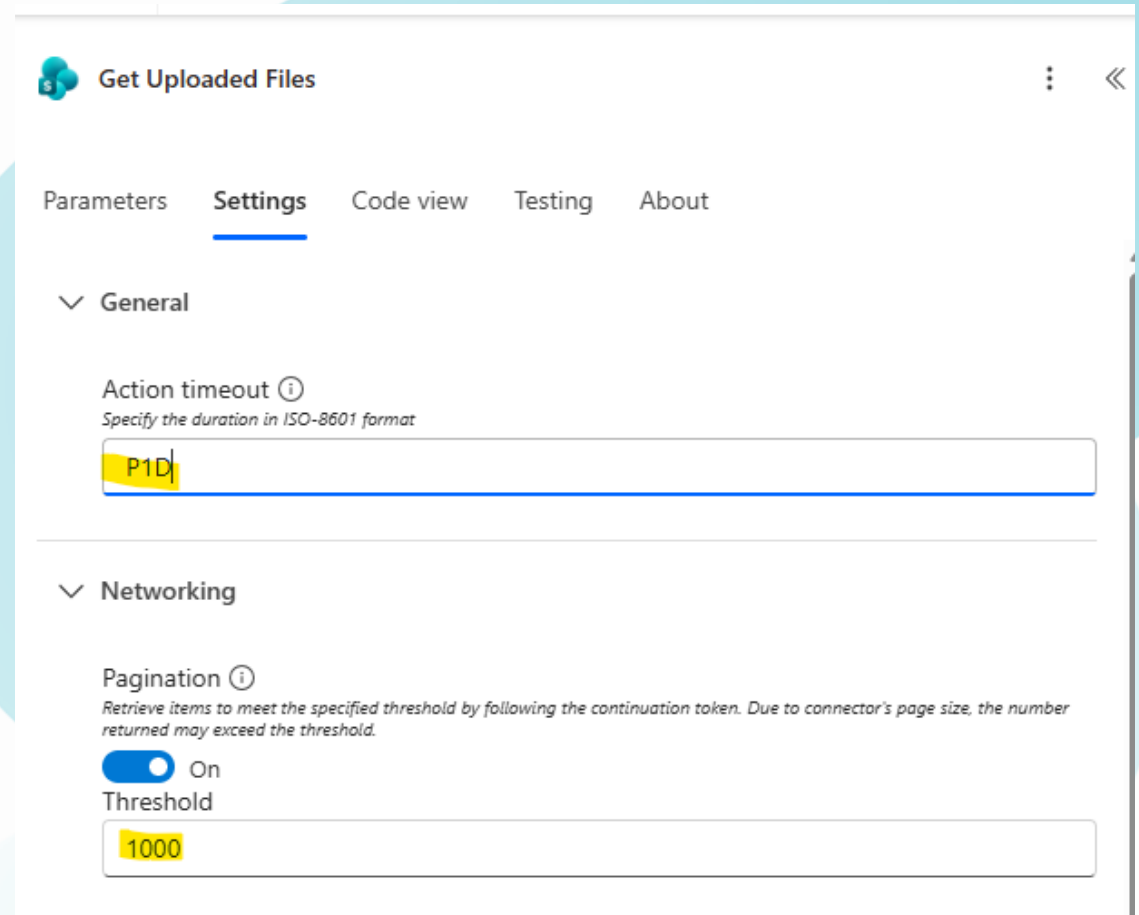
Cloud flows can run up to 30 days including all waiting times before they automatically time out.

## Auto-Expiration

Any pending steps like approvals expire after 30 days if no response is received, stopping the flow.

## Implications for Workflow Design

Workflows cannot indefinitely wait for user actions; design must consider the 30-day timeout limit.



The screenshot shows the 'Get Uploaded Files' connector settings in Power Automate. The 'Settings' tab is selected, showing two sections: 'General' and 'Networking'. In the 'General' section, the 'Action timeout' is set to 'P1D' (1 day). In the 'Networking' section, the 'Pagination' toggle is turned 'On' and the 'Threshold' is set to '1000'.

**Get Uploaded Files**

Parameters Settings Code view Testing About

General

Action timeout ⓘ  
Specify the duration in ISO-8601 format

P1D

Networking

Pagination ⓘ  
Retrieve items to meet the specified threshold by following the continuation token. Due to connector's page size, the number returned may exceed the threshold.

On

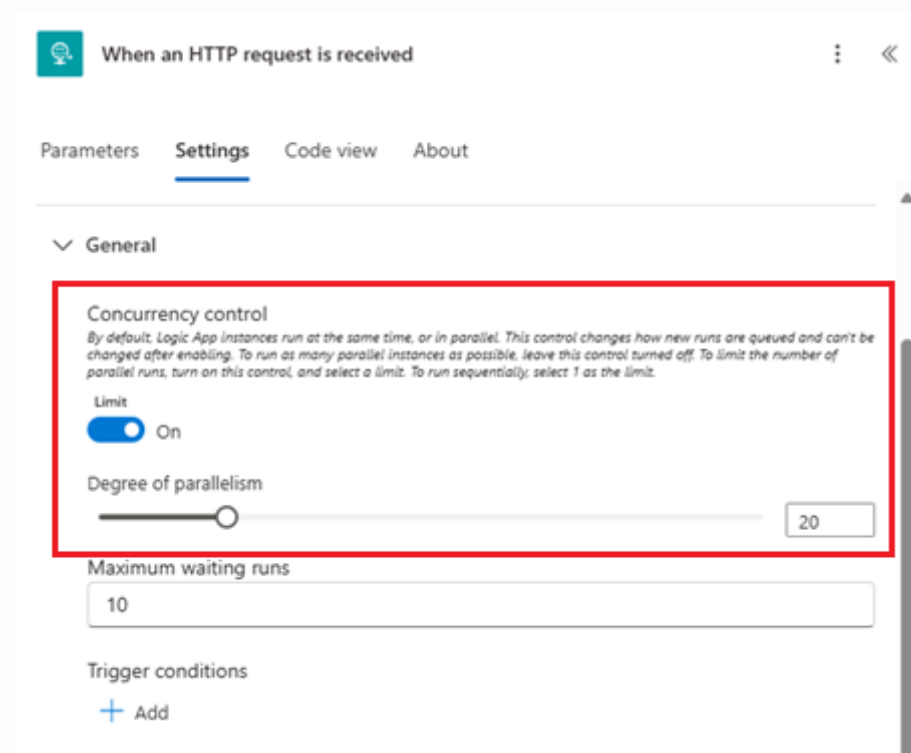
Threshold

1000

# Optimizing Performance with Concurrency Control

## Trigger Concurrency

Triggers can be set to allow multiple concurrent runs or restrict to sequential execution to manage workflow overlap and resource use.



When an HTTP request is received

Parameters Settings Code view About

General

**Concurrency control**  
By default, Logic App instances run at the same time, or in parallel. This control changes how new runs are queued and can't be changed after enabling. To run as many parallel instances as possible, leave this control turned off. To limit the number of parallel runs, turn on this control, and select a limit. To run sequentially, select 1 as the limit.

Limit  
☒ On

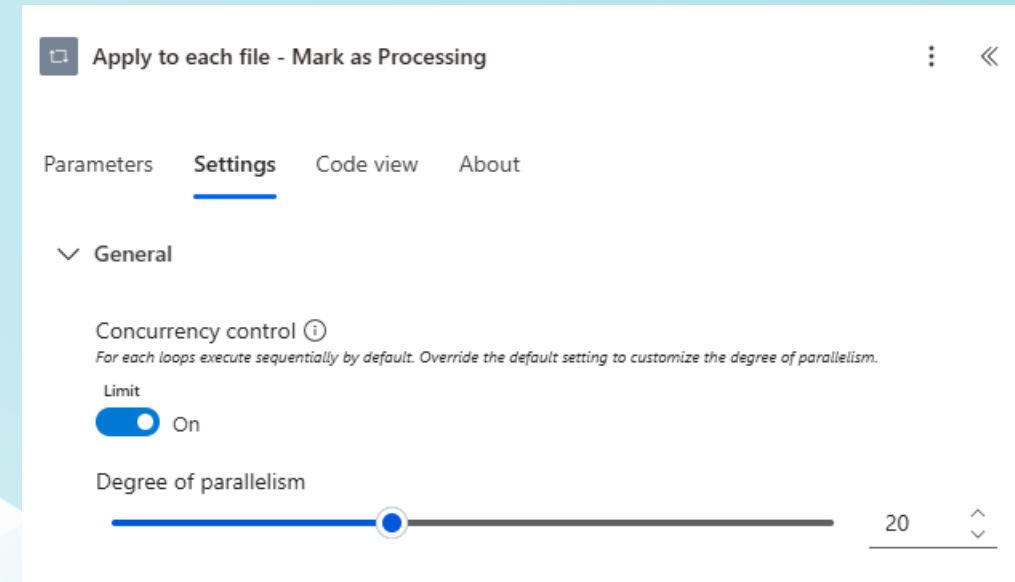
Degree of parallelism  
20

Maximum waiting runs  
10

Trigger conditions  
[+ Add](#)

## Iteration Concurrency

Enabling concurrency allows multiple loop iterations to run simultaneously, improving speed for independent tasks. Use moderate parallelism to avoid conflicts or throttling.



Apply to each file - Mark as Processing

Parameters Settings Code view About

General

**Concurrency control** ⓘ  
For each loops execute sequentially by default. Override the default setting to customize the degree of parallelism.

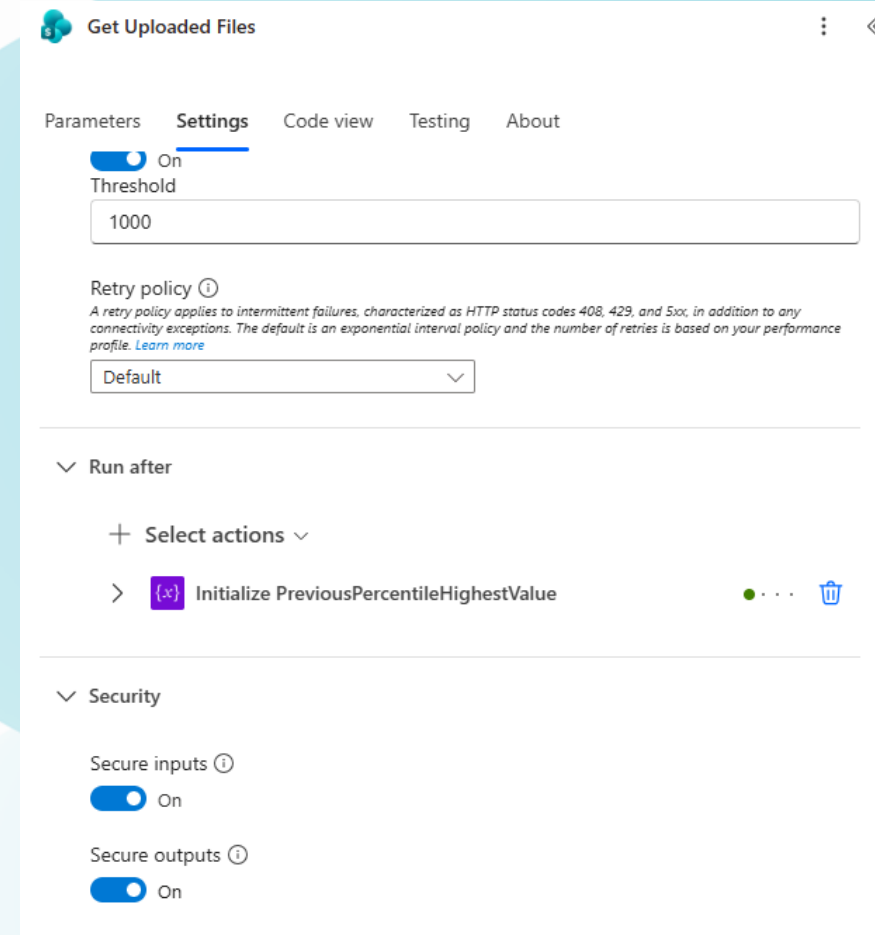
Limit  
☒ On

Degree of parallelism  
20

# Security, Branching, and Best Practices

## Secure Inputs and Outputs

Mask sensitive data in flow logs by enabling secure inputs and outputs to protect API keys and secrets from exposure.



# Interactive Experiences with Adaptive Cards



# Adaptive Cards in Teams and Outlook

## Adaptive Cards Overview

Adaptive Cards are JSON-defined UI snippets that render interactive cards in apps like Teams and Outlook.

## Use Cases

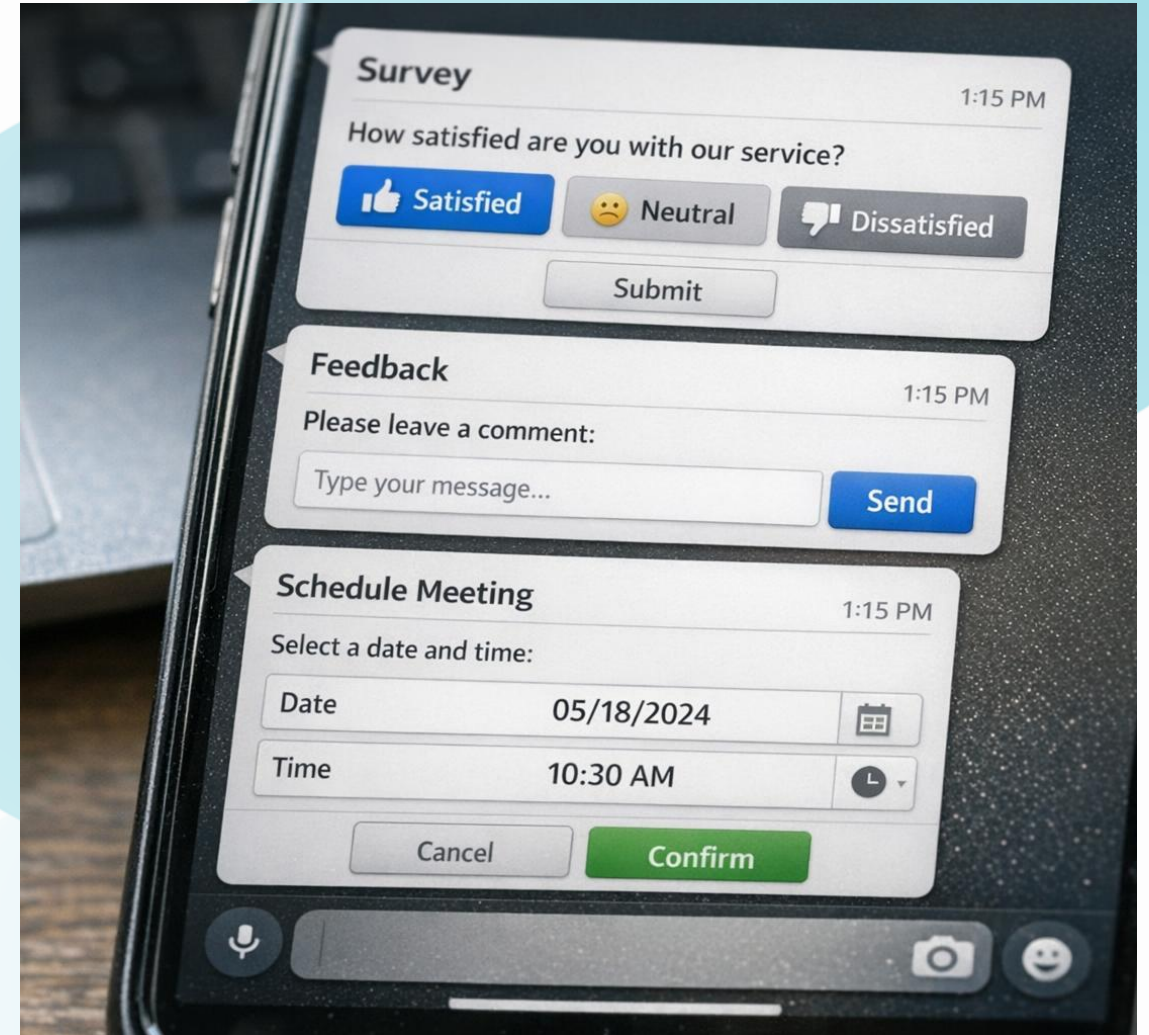
Use cases include approvals, surveys, data collection, decision forms, and quizzes within Teams chat.

## Power Automate Integration

Power Automate posts adaptive cards to Teams users or channels, optionally waiting for user responses.

## Outlook Actionable Messages

Adaptive Cards can be sent as actionable emails in Outlook, though response handling requires additional setup.



# Design and Limitations of Adaptive Cards

## Wait for Response Pattern

This pattern pauses the flow until the user submits the interactive card, resuming with their inputs.

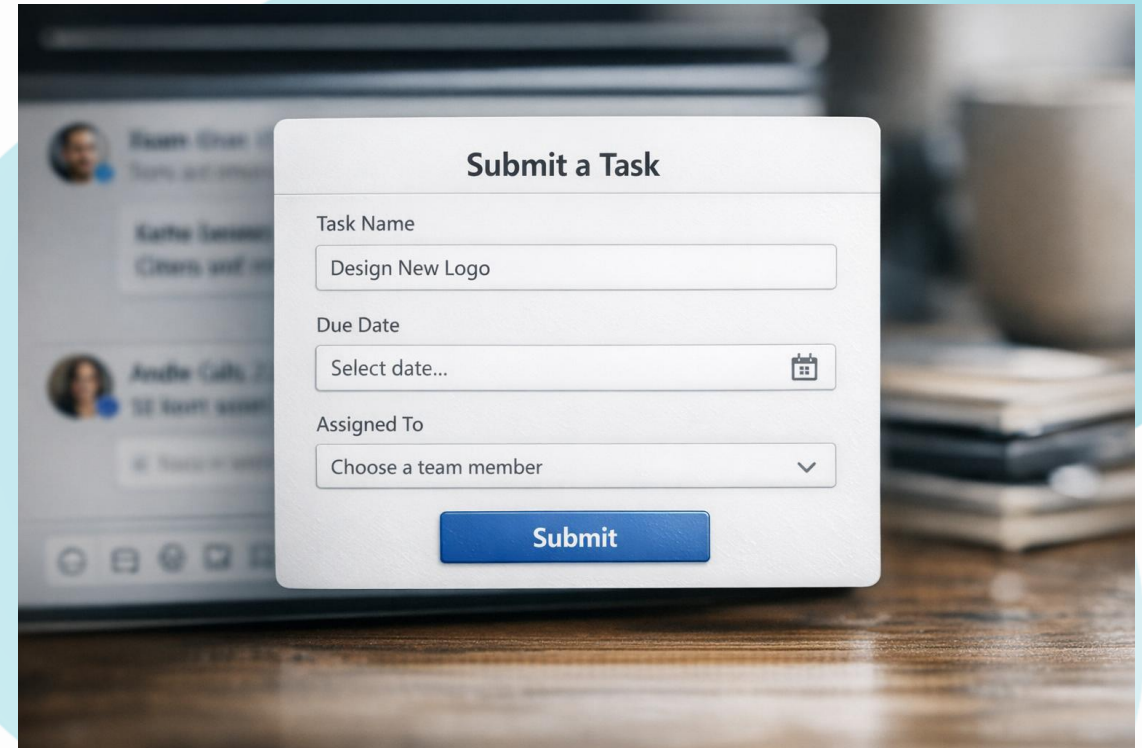
## Designing Adaptive Cards

Cards are designed visually with Adaptive Card Designer or Teams Card editor using JSON for customization.

<https://adaptivecards.microsoft.com/designer>

## Limitations and Security

Cards accept responses only from targeted users or channels, with some UI elements unsupported in Teams.



# Power Platform Expert Bootcamp – Day 3

# Session Schedule (Day 3)

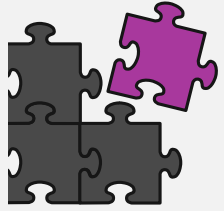
Time	Activity Title
09 00-10 15hrs	ALM/DevOps
10 15-10 30hrs	Break
10 30-12 00hrs	Power BI Integration & Solution Documentation
12 00-13 00hrs	Lunch
13 00-15 30hrs	Hands On Lab Invoice Approval Power BI dashboard

# Agenda Overview

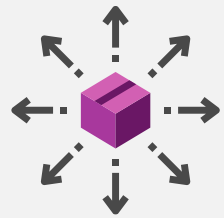
- Solutions
- Application Lifecycle Management (ALM) and DevOps in Power Platform
- Power Platform Pipelines
- Power Platform Solution Documentation
- Integrating Power Apps and Power BI Reports

Solutions

# What is a solution?



***Solutions*** are ***authored, packaged,*** and ***maintained*** by a ***publisher*** allowing customizations to components that extend your Power Platform organization



***Solutions*** allow ***distribution*** of business functionality across Power Platform environments

# Deployment Using Solutions

- In Power Platform, solutions are leveraged to transport apps and components from one environment to another or to apply a set of customizations to existing apps.
- A solution can contain one or more apps and flows as well as other components such as site maps, tables, processes, web resources, option sets, and more.
- Some components are nested within other components, for example, a table contains forms, views, charts, fields, entity relationships, messages, and business rules.
- Solutions are stored within Dataverse and so need environments with Dataverse installed to be used.



# Solution Components

Schema

User  
Interface

Environment  
Variables

Process  
& Code

Connection  
References

Security

Settings



## Power Apps

Low-code application  
development



## Power Virtual Agents

Powerful chatbots

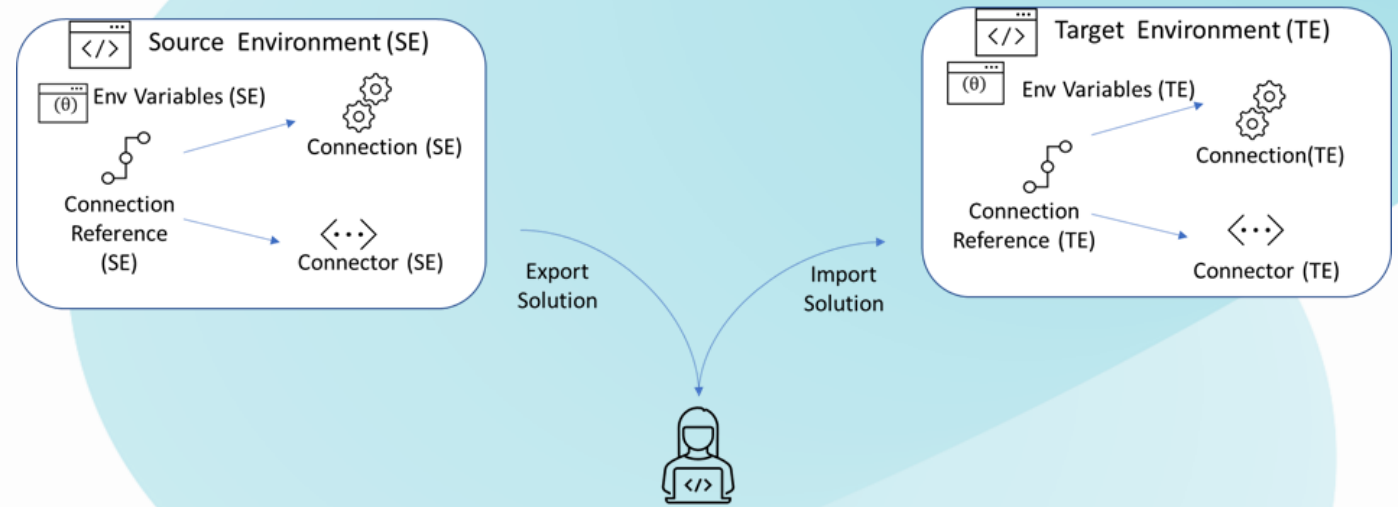


## Power Automate

Intelligent process  
automation

# Environment Variables and Connection References

- Environment variables allow you to transport application configuration data with solutions and optionally manipulate the values in your Application Lifecycle Management (ALM) pipeline.
- Connection references ensure that connections can participate in ALM by removing hard-coded identifiers within components and allowing new connections to be associated as you migrate solutions across environments.



# Unmanaged vs Managed solutions



## Unmanaged Solution

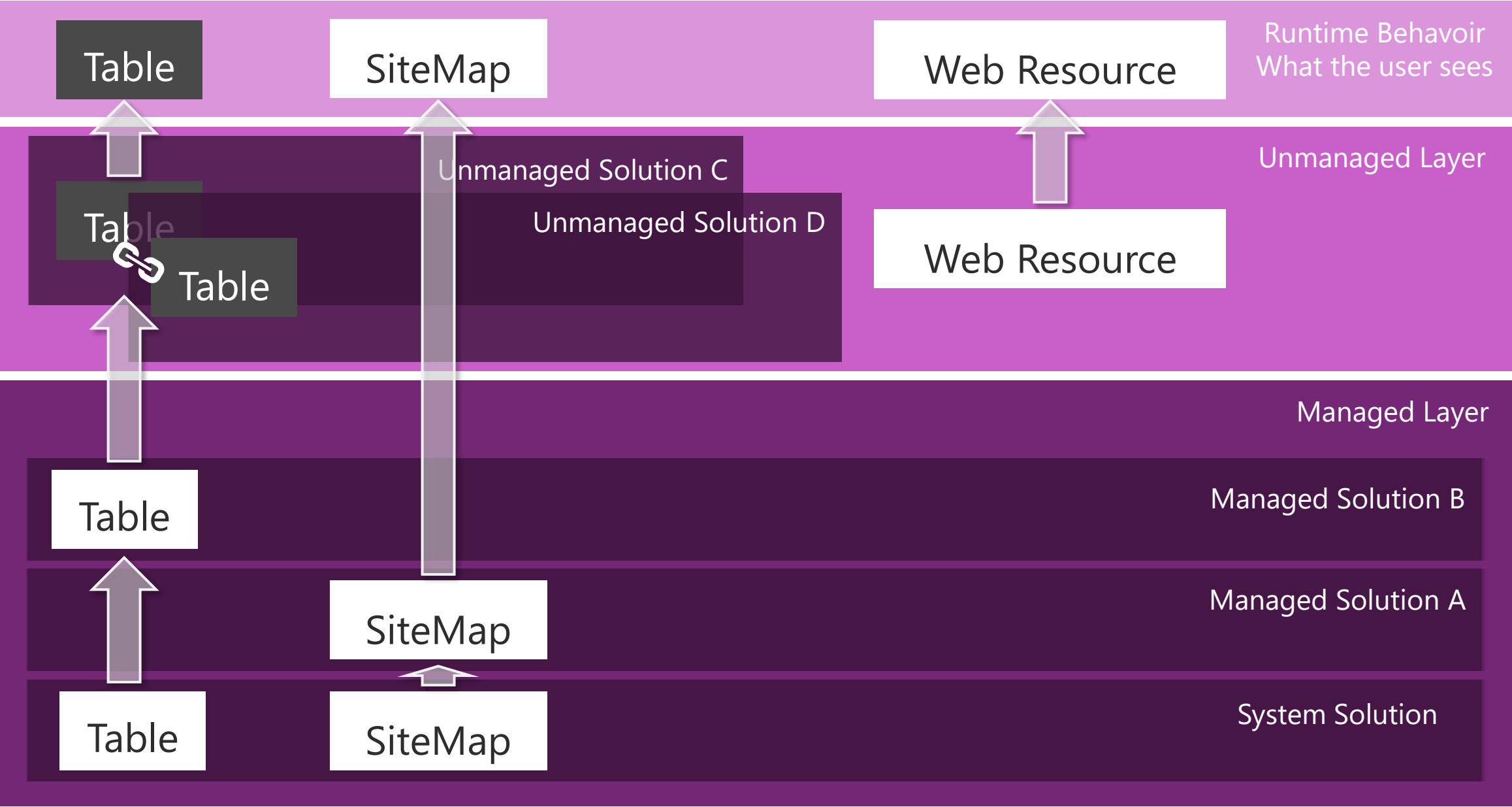
Collection of references to components  
No restrictions on what can be added, removed, or modified  
Recommended only during development of the solution



## Managed Solution

Sealed "binary"  
Components can't be added or removed  
Cannot be exported  
Recommended when a solution is not actively being customized\*

# Solution layering - basics



# Connection References, Versioning, and Governance

## Connection References & Environment Variables

Connection references act as placeholders for connections, enabling easy mapping in different environments. Environment variables carry config specific to each environment.

## Managed vs Unmanaged Solutions

Development uses unmanaged solutions for editing; managed solutions deployed to production ensure stability and prevent direct edits.

## Versioning and Deployment Lifecycle

Increment solution versions for releases; pipelines deploy tested versions to test and production ensuring controlled upgrades and rollbacks.

## Governance and ALM Process

Governance enforces authorized deployments and approvals to prevent unapproved production changes, maintaining source of truth integrity.

### New Connection Reference

[Learn about](#) connections and connection references

Display name \*

Name \* ⓘ

Add a description

Connector \*

Connection \* ⓘ Refresh

CreateClose

### Settings

- General
- Display
- Upcoming features

### Support

Environment ⓘ  
Contoso (default)

**Authoring version**  
3.21102.25

Edit

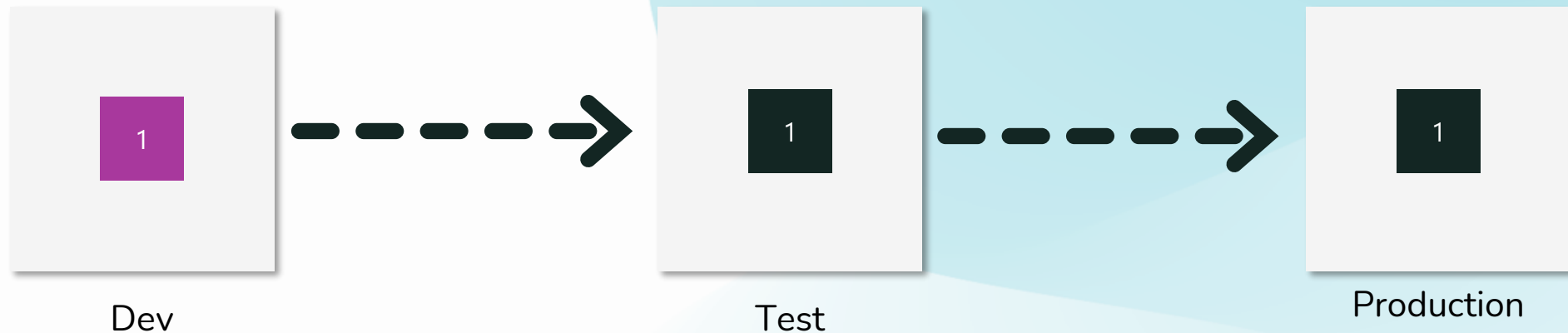
Session ID  
328053be-4dc8-483b-a0b0-47ceb151ed45

Session details

# ALM and DevOps in Power Platform

# Application Lifecycle Management

- Application Lifecycle Management of a software includes plan and track, develop, build and test, deploy, operate, monitor, and learn from discovery.
- For a healthy ALM, there should be separate Dev, Test and Prod environments.
- Source control should be your source of truth for storing and collaborating on your components.
- Automated deployment mechanisms are preferred over manual.



# DevOps Principles: Source Control, CI/CD, and Team Collaboration

## Source Control and Versioning

Power Platform solutions can be exported and tracked using source control systems like Git for version management.

## Environment and Branch Strategy

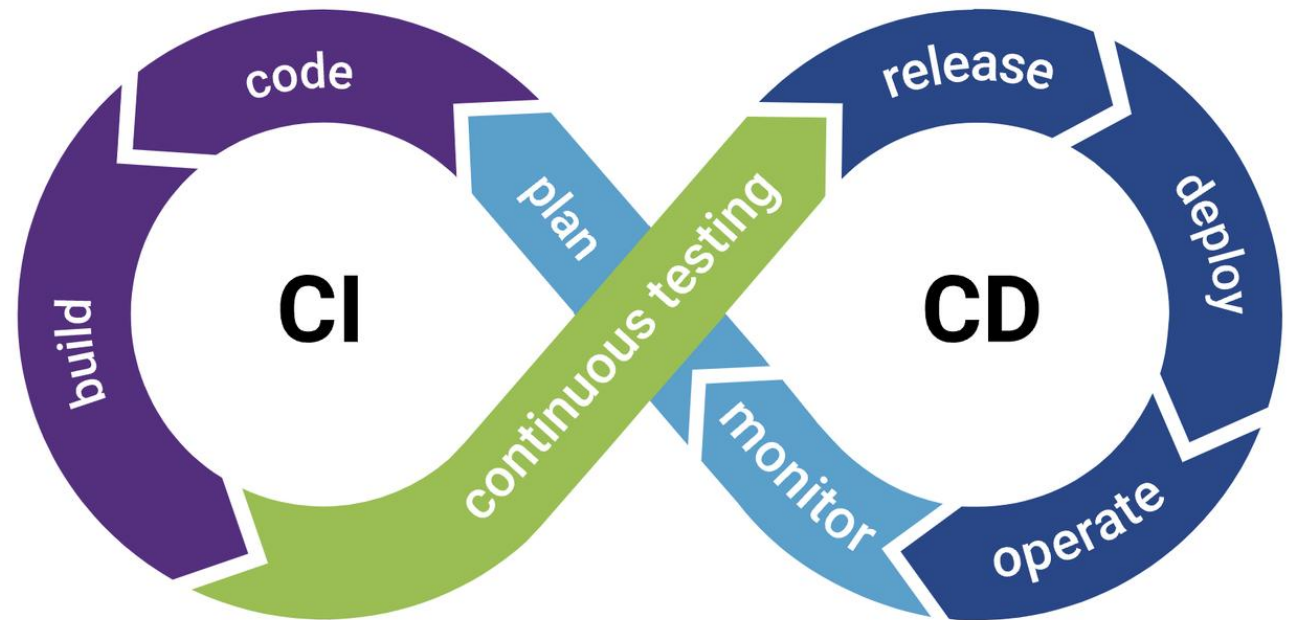
Use Dev, Test, and Prod environments as branches to safely build and deploy solutions without affecting production.

## Build and Deployment Automation

Automate solution export, import, and quality checks using Azure DevOps or GitHub Actions for reliable deployments.

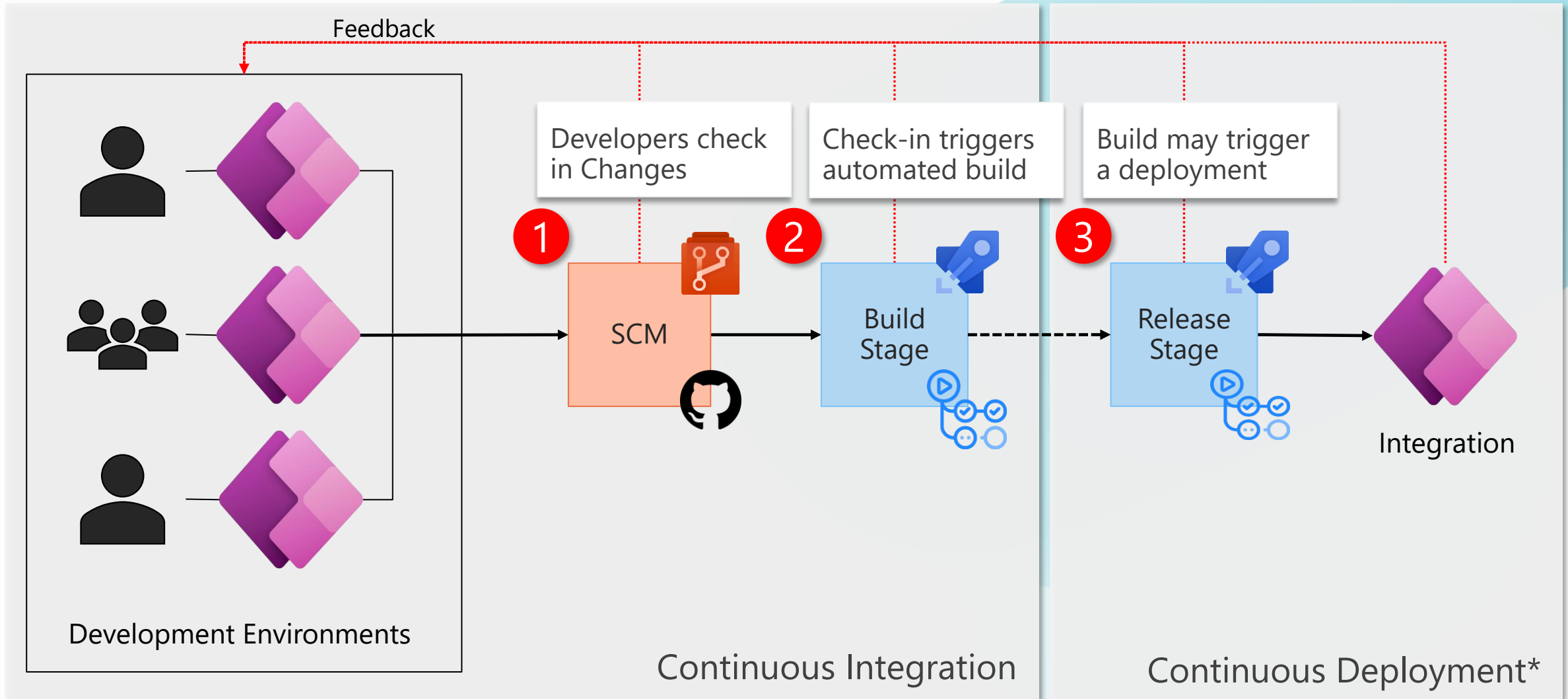
## Team Collaboration and Testing

Coordinate multiple makers through solution ownership, peer review, and quality gates like Solution Checker tests.





# Team development Overview with DevOps



# DevOps Options in Power Platform

Easy to Difficult

Capability	Pipelines	ALM Accelerator	DevOps (ADO)/GitHub
IT / Developer involvement	Not required	Up-front set up	Required for every project
Source Code integration	No, but planned	Yes	Yes
Maker requires elevated privileges in target environment	Yes, but service principal support planned	No, service principal supported	No, service principal supported
Quality Control	Minimal	Best practices	Unlimited
Democratized for Citizen Development	Yes	Yes	No
In-Product Experience	Yes	Canvas app provided, but not in maker experience	No
Support	Microsoft supported	Power CAT support via GitHub Issues and Discussions	Microsoft supported and GitHub Issues
Customization	No	Yes	Yes
Code-first development	No	Yes	Yes

# Power Platform Pipelines

# Power Platform Pipelines- In-Product ALM with Managed Environments

Pipelines in Power Platform aim to democratize application lifecycle management (ALM) for Power Platform and Dynamics 365 customers by bringing ALM automation and continuous integration and continuous delivery (CI/CD) capabilities into the service in a manner that's more approachable for all makers, admins, and developers.

The screenshot displays the Power Apps Pipelines interface. At the top, the header shows "Power Apps" and the environment "Workshop-Dev". The main section is titled "Pipelines" and includes a dropdown menu for "Pipeline" set to "Demo". Below this, there are two tabs: "Details" (selected) and "Run history".

The interface shows three environments in a sequence, connected by arrows:

- Development**: A box with a yellow header. It contains the text: "Securely test and verify in isolation. After you've got everything working properly, it's time to deploy. [Learn more](#)". Below this, it shows "Solution version 1.0.0.1".
- Dev-Test**: A box with a yellow header. It includes a "Go to this environment" link. Below the header, it shows "Last successfully installed version 1.0.0.1" and "Last deployed Sep 27, 2023 1:54 PM (a day ago)". A message at the bottom states: "Deployments may be pending until an associated background process succeeds. This process is managed by your admin. [Learn more](#)". A "Deploy here" button is at the bottom right.
- Test-Prod**: A box with a yellow header. It includes a "Go to this environment" link. Below the header, it shows "Last successfully installed version 1.0.0.1" and "Last deployed Sep 27, 2023 3:22 PM (a day ago)". A message at the bottom states: "Deployments may be pending until an associated background process succeeds. This process is managed by your admin. [Learn more](#)". A "Deploy here" button is at the bottom right.

# Why Power Platform Pipelines

Centrally view all deployment activity across my team, organization, or tenant

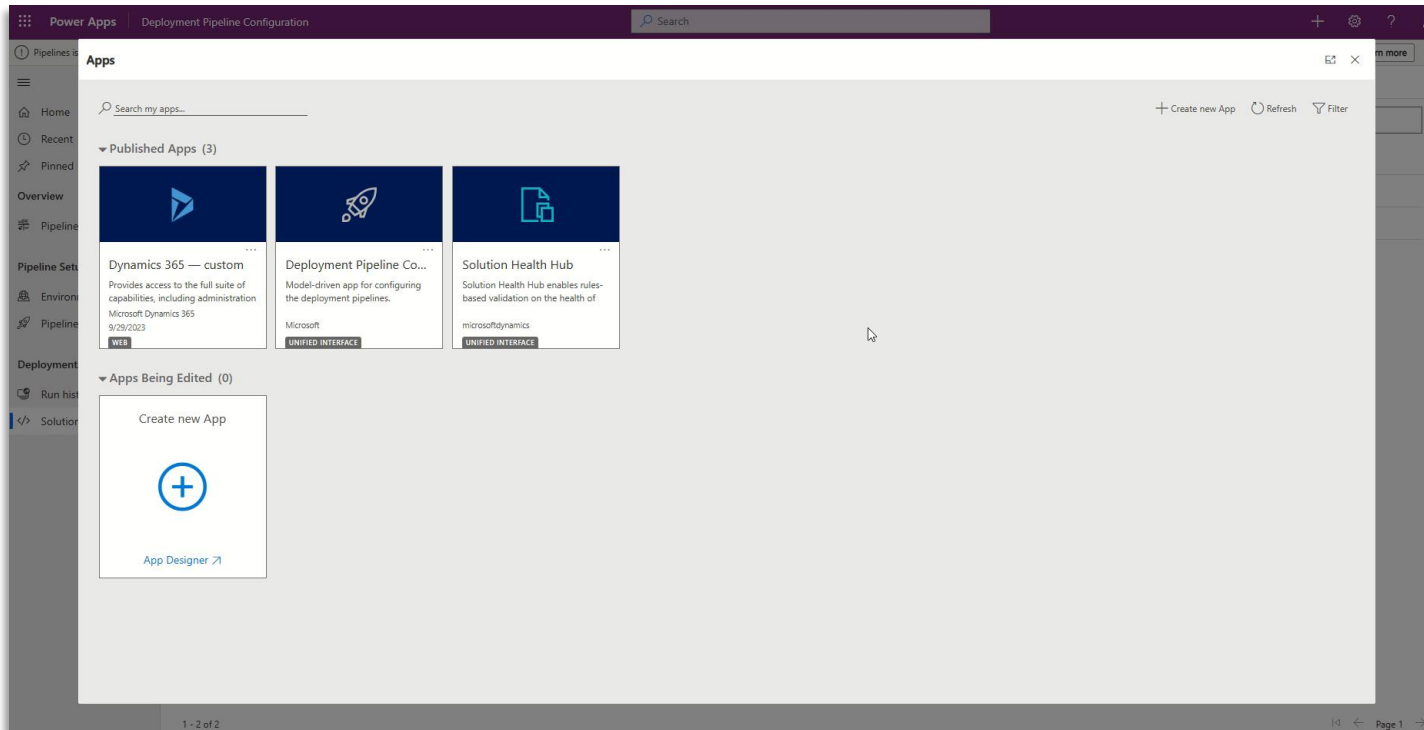
“Host” Dataverse DB is the Pipelines control center

Centrally manage pipelines & security

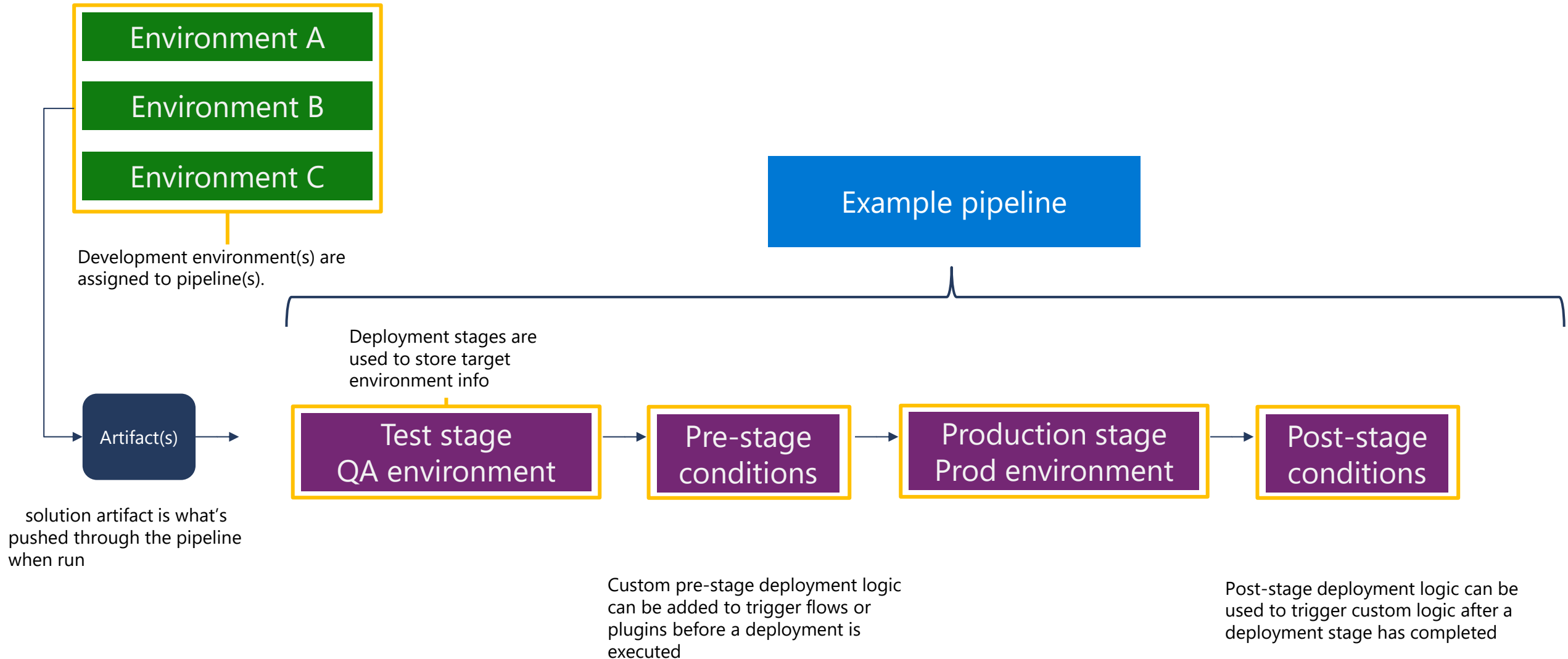
Every solution, every version, every error log is automatically retained in “host” DB

Access to Pipelines and resources are controlled by security roles

- Deployment Pipeline Admin
- Deployment Pipeline User



# Pipelines in Power Platform Key Concepts



# Deep dive: Pipeline extensibility

The screenshot displays the Power Automate interface with the 'OnPreDeploymentStartedSample' flow selected. The flow consists of two main steps:

- OnPreDeploymentStarted**: This step is configured with the following details:
  - Trigger condition**: Set the trigger condition in the settings as "@equals(triggerOutputs()?['body/OutputParameters/DeploymentPipelineName'], 'Northwind Traders Pipeline')". This ensures that flow will only execute when the pipeline name is 'Northwind Traders Pipeline'.
  - Catalog**: Microsoft Dataverse Common
  - Category**: Power Platform Pipelines
  - Table name**: (none)
  - Action name**: OnPreDeploymentStarted
- start and wait for an approval**: This step is configured with the following details:
  - Approval type**: Approve/Reject - First to respond
  - Title**: Please review the Deployment request for [ActionOutputs ...]
  - Assigned to**: Radha@gccssenv.onmicrosoft.com
  - Details**:
    - # Please review the Deployment details
    - 1. #### PipeLineName: [OutputParamet...]
    - 1. #### StageName: [OutputParamet...]
    - 1. #### Artifact Name: [OutputParamet...]
    - 1. #### Artifact Version: [OutputParamet...]
    - 1. #### [Item Link: Stage Run Link] [ActionOutputs ...]
    - 1. #### Deploy Requested by: [ActionOutputs ...]
    - 1. #### Scheduled Time: [if(...)]
    - 1. #### Deployment Notes: [ActionOutputs ...]
  - Item link**: [ActionOutputs ...]
  - Item link description**: Solution Artifact Download

## Pre-deployment

- Produces event to trigger cloud flows and/or plugins
- Deployment is pending (blocked) until flow or plugin updates deployment request to approved/rejected
- Executes after system pre-validates deployment & stores artifact in host

## Post-deployment

- Produces non-blocking event
- Ex: send notification when deployment completes
- Note: pre-deployment events are always fired but won't block deployment unless set as required.

# Configuring Approvals and Governance Before Production Deployment

## Governance and Change Management

Organizations require formal approvals before deploying to production to ensure accountability and alignment with change management policies.

## Approval Gates in Pipelines

Azure DevOps and Power Platform pipelines can enforce approval gates, requiring designated approvers before production deployment.

## Approver Roles and Responsibilities

Approvers like project sponsors or IT managers review changes, test results, and grant deployment permission, ensuring oversight.

## Automating Approvals with Power Automate

Power Automate can streamline release approvals by triggering deployment automatically once the approval is granted.





# Data Migration and Common ALM Issues

## Metadata vs Business Data

Solutions include metadata like tables and apps but exclude business data records, requiring separate migration tools.

## Missing Dependencies

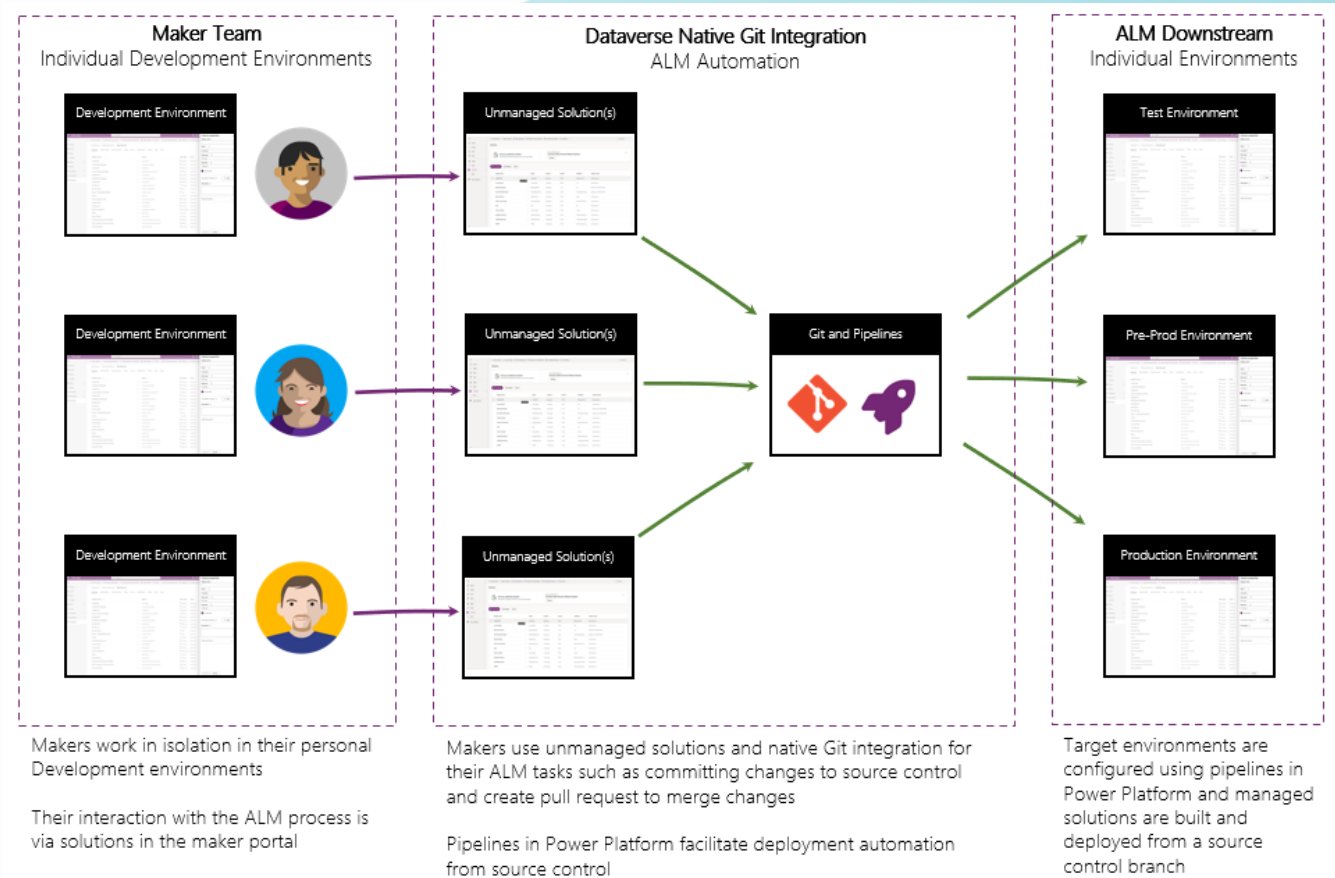
Forgetting components such as connection references can cause import failures or malfunctioning flows.

## Syncing Multiple Developers

Coordinating changes among multiple developers is challenging; use a single source of truth or source control.

## Environment Differences

Target environments need proper configuration like user accounts and roles; solutions don't handle user sharing.

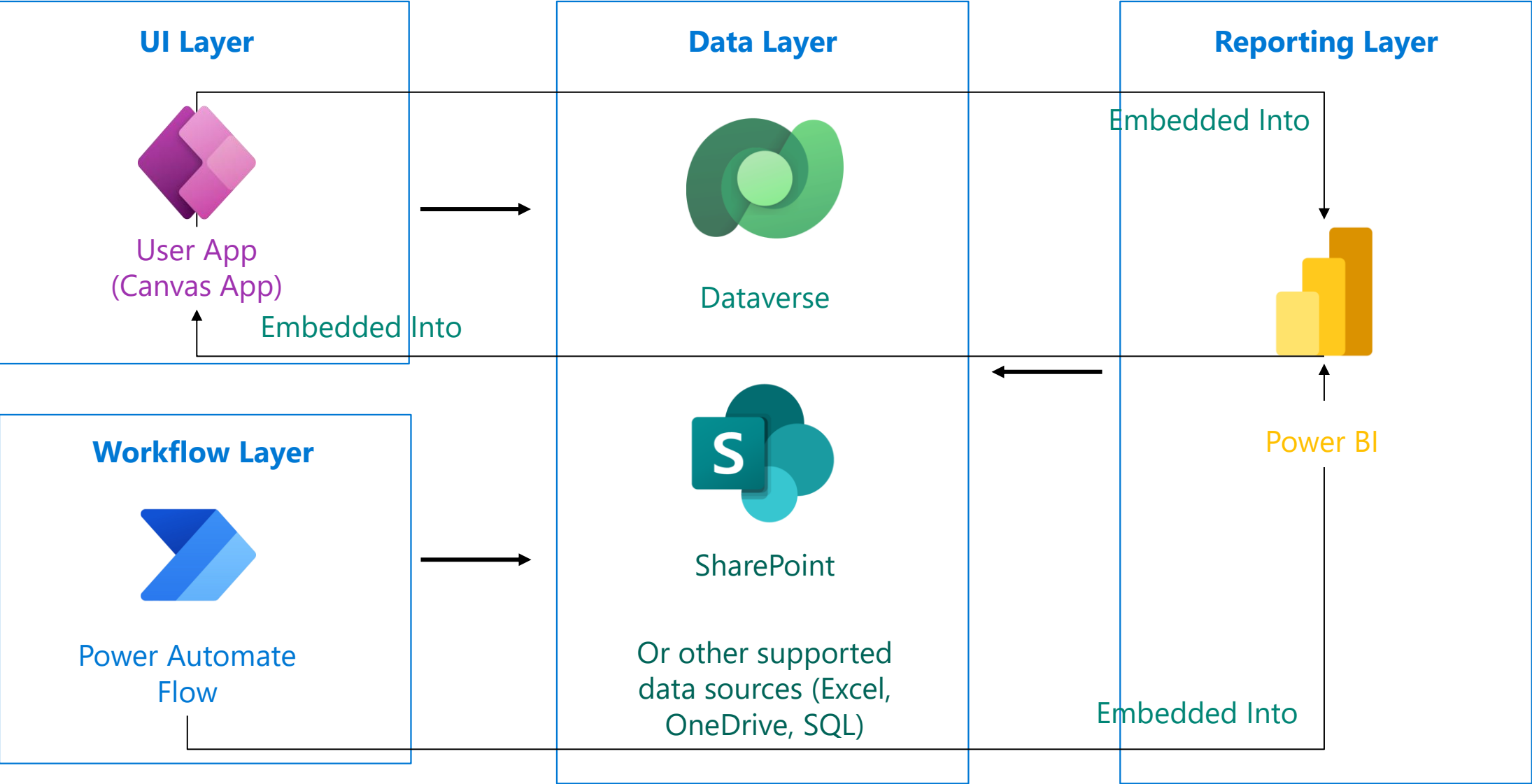


You cannot delete while the following objects depend on it.

Display name ↑	Name	Object type	Dependent table	Managed solution	Dependency type
Active Loans	Active Loans	View	Loan		Published
Information	Information	System Form	Loan		Published
Loan main form	Loan main form	System Form	Loan		Published

# Integrating Power Apps and Power BI Reports

# Power BI and Power Apps/Automate Integration Possibilities



# Integrating Power BI Dashboard in Power Apps

## Power BI Dashboard

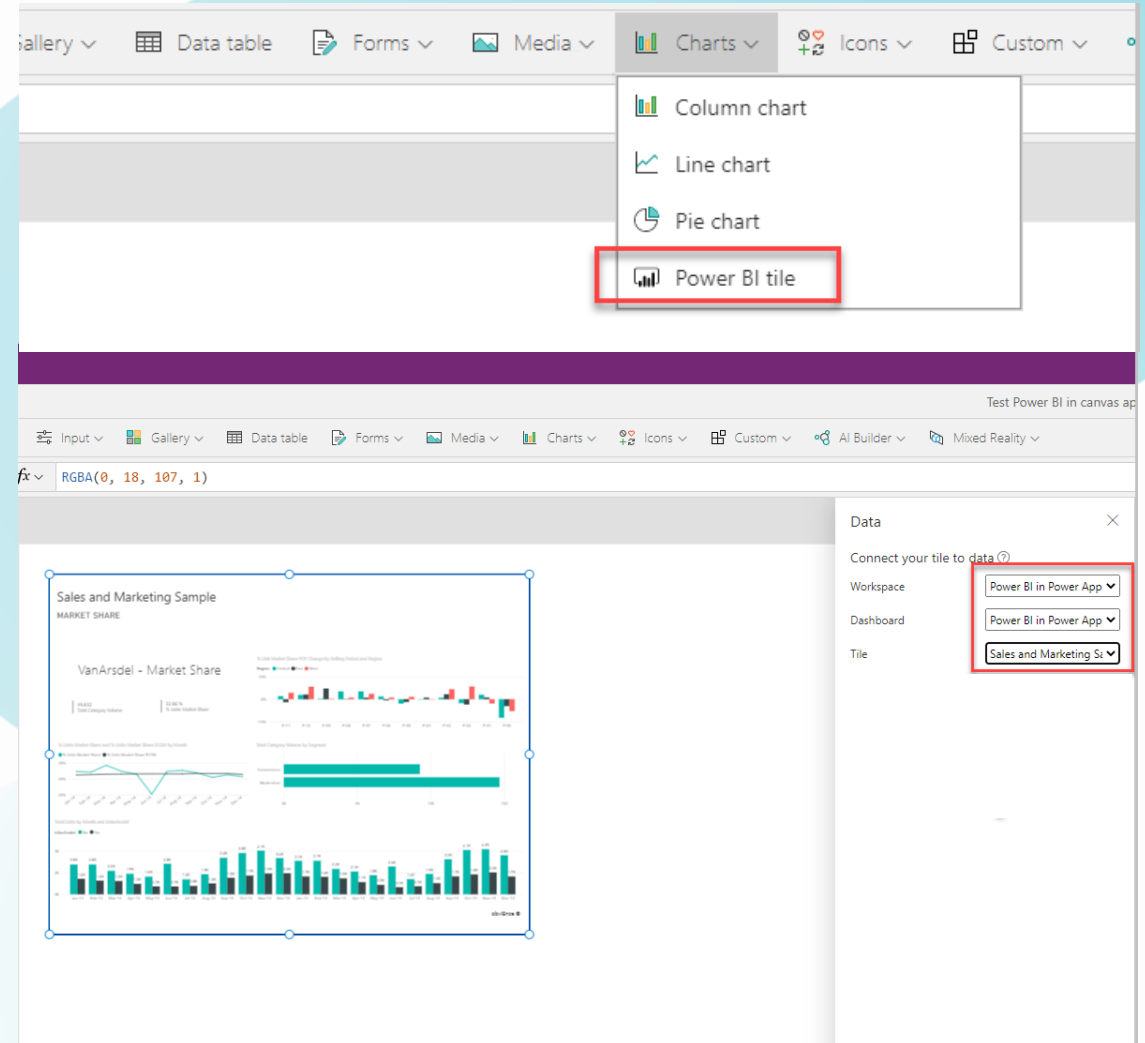
Power BI reports will have to be pinned to a dashboard for Power Apps to be able to view the information. Configure reports via Power BI portal to dynamically display visuals for the report.

## Dynamically Update Visuals

Updating the Power BI Dashboard will automatically update all apps that are using the Tile.

## Power BI Report Access Control

Embedded Power BI report also needs to be shared with app users so that they can see the report from inside the app.



# Integrating Power Apps in Power BI

## Embedding Power Apps in Power BI Reporting

Build custom pages to enhance reporting dashboard. Enable interactions to perform additional actions (edit, new, delete) to backend database.

## Custom Input and Output Properties

Embedded canvas apps support input properties for data and output properties to display customized information, enabling additional way to interact.

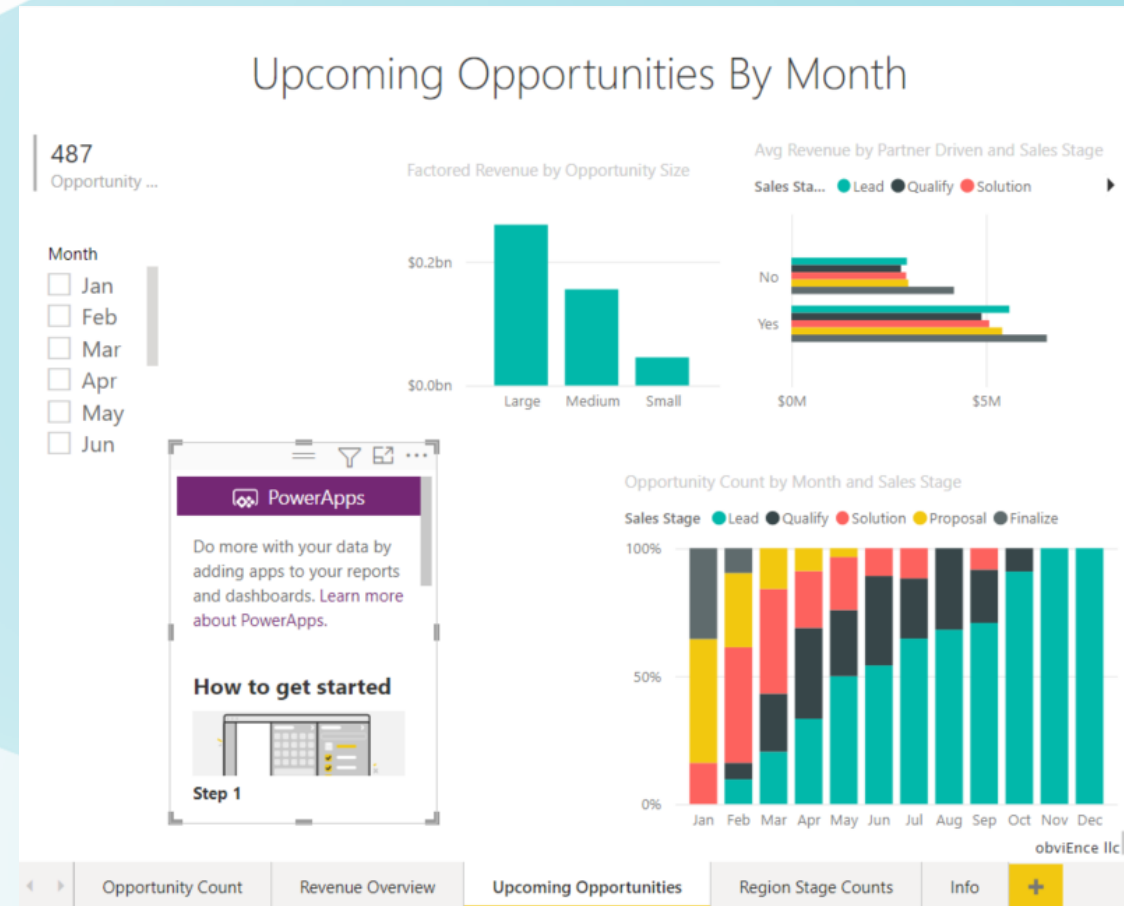
## Common Use Cases

Typical embedded canvas apps are used to collect additional information, perform calculations, to collect information used for follow-up to the report

## Data Refresh

*Note: By default, the data on the report will only refresh as part of the scheduled/manual refresh. Not upon update via embedded canvas app.*

*If the data needs to be refreshed, canvas app must trigger a data refresh to Power BI using **PowerBIIntegration.Refresh**.*



# Embedding Power Automate Flow in Power BI

## Embedding Power Automate in Power BI Reports

Embed Power Automate flows directly within Power BI to extend report interactivity. Workflow initiations — directly from within a report, without leaving the Power BI experience.

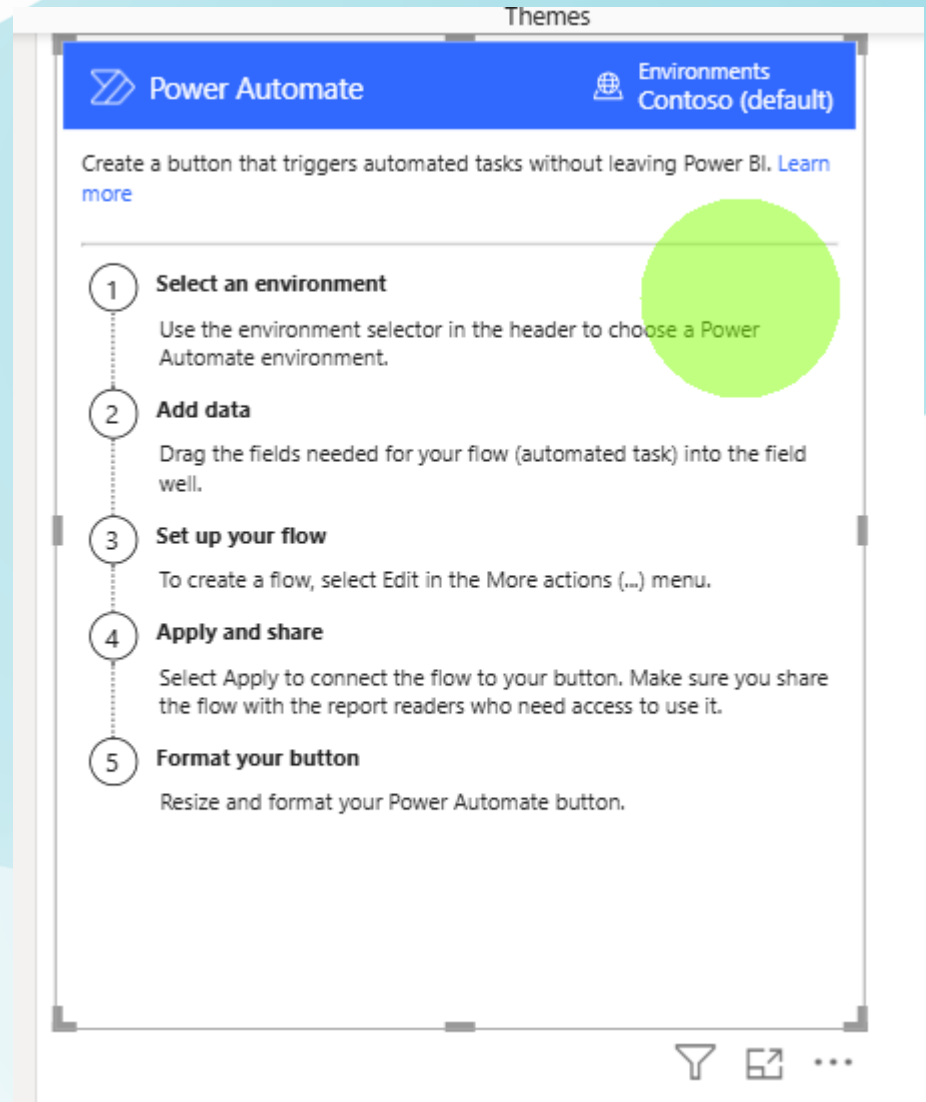
## Custom Inputs and Flow Parameters

Power Automate visual supports sending contextual data from Power BI (such as selected rows, filters, or metrics) as input parameters to a flow.

## Common Use Cases

- Sending alerts or reminders based on selected report data
- Triggering approval workflows from operational dashboards
- Creating tasks or tickets for items highlighted in analytics

*Note: Flows trigger instantly, but **report data itself will not refresh as a result of a triggered flow**. Visuals update only during scheduled/automatic refresh or when performed manually.*



# Power Platform Solution Documentation

# Documentation Best Practices

## ✓ Use consistent naming conventions

Clear, camelCase naming improves clarity and maintainability.

## ✓ Document “just enough,” not everything

Avoid documenting every action; focus on purpose, triggers, and main sequences.

## ✓ Use inline comments in Power Apps formulas

Comments help future maintainers quickly understand logic.

## ✓ Keep architecture diagrams updated

ERDs, integration diagrams, and data flows must reflect current solution state.

## ✓ Leverage AI & automation tools

AI Doc, PowerDocu and GitHub Copilot help generate solution summaries and markdown docs.

[Power-CAT-Tools/AI\\_DOCUMENTATION.md at main · microsoft/Power-CAT-Tools](#)

<https://pnp.github.io/cli-microsoft365/cmd/flow/flow-get>

[modery/PowerDocu: Generate technical documentation from your existing Power Automate Flows and Power Apps canvas apps](#)

## ✓ Keep documentation accessible (SharePoint, OneDrive)



End