## Introduction to developing with Microsoft Power Platform

Five main components: Power Apps, Power Automate, Power BI, Power Pages, and Power Virtual Agents

Graphical user interface, application

Description automatically generated

1. Power Apps Learning Path:

<https://learn.microsoft.com/en-us/training/paths/create-powerapps/>

<https://learn.microsoft.com/en-us/training/paths/create-app-models-business-processes/>

1. Power Automate Learning Path:

<https://learn.microsoft.com/en-us/training/paths/automate-process-using-flow/>

<https://learn.microsoft.com/en-us/training/paths/pad-get-started/>

1. Power BI Learning Path

<https://learn.microsoft.com/en-us/training/paths/create-use-analytics-reports-power-bi/>

1. Power Pages Learning Path

<https://learn.microsoft.com/en-us/training/paths/power-pages-get-started/>

1. Power Virtual Agents Learning Path

<https://learn.microsoft.com/en-us/training/paths/work-power-virtual-agents/>

Connector - is a proxy or a wrapper around an API that allows the underlying service to talk to Microsoft Power Automate, Microsoft Power Apps, and Azure Logic Apps. It provides a way for users to connect their accounts and use a set of prebuilt actions and triggers to build their apps and workflows.

Dataverse – cloud-scale data store allows RBAC on tables and at rows level.

Common Data Model - an open-sourced standard definition of entities that represent commonly used concepts and activities. Tips: Use the Common Data Model Library to work with data stored in Azure Data Lake. Refer to documentation [here](https://learn.microsoft.com/en-us/common-data-model/schema/core/applicationCommon/overview/).

Graphical user interface, application

Description automatically generated

**Common Data Model** is influenced by data schemas that are present in Dynamics 365, covering a range of business areas.

The Common Data Model is maintained in [GitHub](https://github.com/microsoft/CDM/). Schemas are maintained as JSON files. An entire entity reference can be found at [Overview of applicationCommon](https://learn.microsoft.com/en-us/common-data-model/schema/core/applicationCommon/overview/). Microsoft also provides a [Visual Entity Navigator](https://microsoft.github.io/CDM/) that allows users to visually navigate Common Data Model entities.

Extensibility - any code-based customization activities are referred to as "extending" an application. To extend the user experience of a Power Apps app, you can use *Power Apps Component Framework (PCF)* to create code components that are used across all Power App types. Model-driven Power Apps also exposes a JavaScript client API that allows you to interact with application pages to implement business logic on the application forms.

While canvas apps don't offer scripting capabilities, their logic is built using [Microsoft Power Fx](https://learn.microsoft.com/en-us/power-platform/power-fx/overview/). Power Fx is the low-code language that is used across Microsoft Power Platform. It's a general-purpose, strong-typed, declarative, and functional programming language.

Extending Dataverse - need for some level of automation or behaviour not supported by the existing features. These types of extensibility points run as server-side code through a construct called a plug-in. When requirements for synchronous operations exist within an application, plug-ins are still required. Synchronous processing by plug-ins allows any Dataverse modifications to be treated as a single unit of work in a transaction.

Dataverse Solutions - Solutions are the mechanism for implementing application lifecycle management (ALM) in the Power Platform. For detailed information about the solution concepts, see [Solutions in Power Apps](https://learn.microsoft.com/en-us/power-apps/maker/data-platform/solutions-overview/) in the Dataverse documentation. Power Platform build tools can be used with Azure Dev Ops or GitHub to automate Power Platform ALM operations. For more information how solutions are used for application lifecycle management, see [Overview of ALM with Microsoft Power Platform](https://learn.microsoft.com/en-us/power-platform/alm/overview-alm/) in Microsoft Power Platform ALM guide.

### Extending Power Platform with Azure

Azure cloud services provide a wealth of functionality and enable Microsoft Power Platform developers to harness its capabilities through various extensibility points.

Azure Functions - is a supported extensibility endpoint for Power Platform. Use it to implement business logic outside of apps. Build APIs that are exposed to Microsoft Power Platform as a **custom connector**. As a custom connector the Azure Function business logic can easily be invoked from apps and flows.

API Management - manage APIs across clouds and on-premises. In addition, API Management can export API definitions directly to Microsoft Power Platform. When exported, the API is configured as a **custom connector** available in Power Apps and Power Automate.

Service Bus - reliable [messaging-as-a-service](https://azure.microsoft.com/services/service-bus/) (MaaS) framework that enables real-time, asynchronous messaging across systems. Developers can configure Microsoft Dataverse to publish events to Azure Service Bus queues and topics. Events can be published automatically on Dataverse data modifications or on demand from developer's custom logic. Service bus can store the message until the consuming party is ready to receive the messages allowing you to architect solutions that are less dependent.

Event Grid - fully managed [single service for managing routing for all events](https://azure.microsoft.com/services/event-grid/) from any source for any destination. It simplifies the development of event-based applications and the creation of serverless workflows. Event Grid can be used to route events between Microsoft Power Platform and other Azure services like Azure Functions.

Logic Apps - a [cloud service](https://azure.microsoft.com/services/logic-apps/) that helps you schedule, automate, and orchestrate tasks, business processes, and workflows when you need to integrate apps, data, systems, and services across enterprises or organizations.

AI Services - a family of AI and cognitive APIs to help build intelligent apps. Azure AI Services can be categorized into five main pillars: vision, speech, language, web search, and decision.

Azure Data Lake and Azure Synapse analytics - Microsoft Power Platform data seamlessly supports [Azure Data Lake storage](https://azure.microsoft.com/services/storage/data-lake-storage/) with its Microsoft Dataverse and Common Data Model framework. Any data or metadata changes in Dataverse are automatically pushed to the Azure Synapse metastore and Azure Data Lake, depending on the configuration, without any additional action. This is a push, rather than pull, operation. Changes are pushed to the destination without you needing to set up refresh intervals.

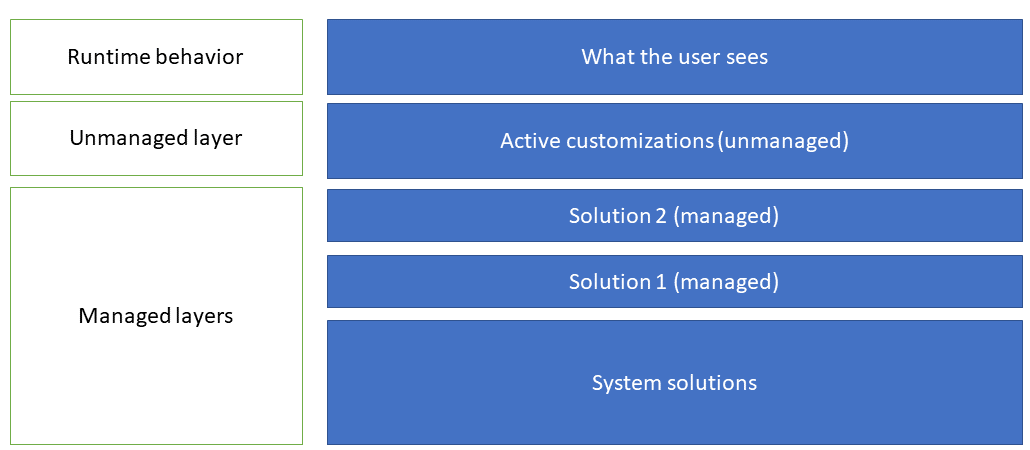
Azure SQL Database - Dataverse itself is built on top of Azure SQL Database engine. Power Apps and Power Automate can work with data in Azure SQL via the Azure SQL connector.

## Use developer tools to extend Power Platform

To get hands on 🡪 <https://learn.microsoft.com/en-us/training/modules/power-apps-canvas-app-online-workshop/>

To learn how to create PCF component using VS Code with Power Platform CLI and push to the environment as a solution, refer <https://learn.microsoft.com/en-us/training/modules/developer-tools-extend/exercise>

* Solution can be layered - Multiple solutions can co-exist within an environment. All unmanaged solutions are tracked as a single layer.
* a form for a Power Apps model-driven app, might be modified by three managed solutions, each changing different sections of the form.
* Below, the merged result would be a combined form, including changes from all three managed solutions.
* When multiple managed solutions are installed, the last one that's installed is above the managed solution that was previously installed. Essentially, the second solution that's installed can customize the one that was installed before it.
* When two managed solutions have conflicting definitions, the runtime behaviour is "last one wins" or a merge logic is implemented.
* If you uninstall a managed solution, the managed solution below it will take effect.
* If you uninstall all managed solutions, the default behaviour that's defined within the system solution will be applied.
* The unmanaged layer, however, is always at the top, and a change in the unmanaged layer will override any change from managed solutions in an environment. Each layer is merged to produce the runtime behaviour as opposed to the "last layer wins" behaviour.
* Ultimately, a user sees a combination of all the solution layers in an environment.



Solution components often depend on other solution components. The solution runtime tracks these dependencies to ensure that solutions can't be removed if they are depended on by another solution.

SolutionPackager Tool - distributed as part of the NuGet package. Can use to conduct the following actions:

* Extract: Extract solution .zip file to a folder [pac solution pack –zipfile xxx]
* Pack: Pack a folder into a .zip file [pac solution unpack –zipfile xxx]

Refer SolutionPackager docs [here](https://learn.microsoft.com/en-us/power-platform/alm/solution-packager-tool). SolutionPackager can be manually run standalone but can make it automated build process for both *Microsoft Power Platform Azure DevOps* builds tasks and *GitHub Actions*.

Tips: *Solution Packer* decomposes Dataverse solutions into multiple XML files and other files. Solution unpack reversibly decomposes a solution .zip file by dividing it into a logical folder structure. Resulting output can be more readily maintained by a source control system.

Package Deployer - lets administrators deploy packages into Microsoft Dataverse environments. Microsoft provides a Visual Studio template for creating packages. Package includes one or more solutions and data.



Sample command:  
pac package init --outputDirectory MyPackage

pac package add-solution --path ..\MySolution1\_1\_0\_0\_2\_managed.zip

pac package add-solution --path ..\MySolution2\_2\_0\_0\_0\_managed.zip

Deploying a Package - can be deployed to Dataverse environments using one of the following methods.

CLI - Power Platform CLI to authenticate to the target environment and then run the pac package command.

Interactively - wizard-like experience, uses the Package Deployment Tool windows application.

Windows PowerShell - appropriate for packages deployed as part of an automated process.

AppSource - used when an ISV offer is selected for install from the AppSource portal.

Developer Tools – following commands open each developer tools.

pac tool cmt [The Configuration Manager Tool should install and launch]

pac tool pd [The Package deployer tool should install and launch]

pac tool prt [The Plugin Registration should install and launch]

Extensibility Points – Power Platform allows various points to write custom code when low code is not feasible.

Graphical user interface, application

Description automatically generated

Figure: Each Extensibility Points will be tagged as EP1, EP2, etc.

UX Extensibility – create a PCF code component (EP2) or implement some client form scripting (EP3)

To learn how to create PCF component using VS Code with Power Platform CLI and push to the environment as a solution, refer <https://learn.microsoft.com/en-us/training/modules/developer-tools-extend/exercise>

To learn how to use client scripting in Power Apps model driven apps, refer <https://learn.microsoft.com/en-us/power-apps/developer/model-driven-apps/clientapi/walkthrough-write-your-first-client-script>

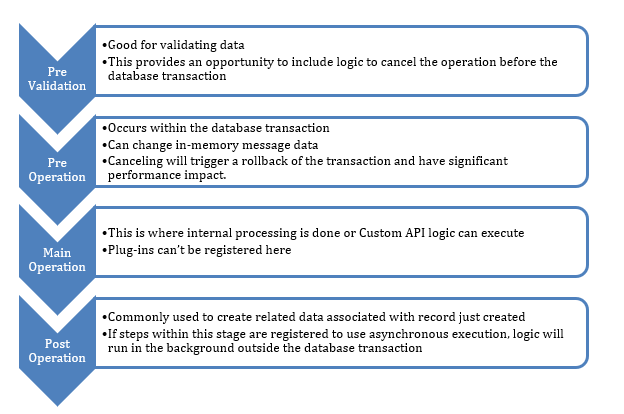
Tips: Client scripting isn't intended to be visual but intended to implement business rules in a programmatic way. Client scripting only applies to model-driven apps.

Dataverse Extensibility – provides two styles of APIs (EP7) for developers: Web API and Organization Service.

[Web API](https://learn.microsoft.com/en-us/power-apps/developer/data-platform/webapi/overview/) is available at an OData v4 RESTful endpoint. Use this for any programming language that supports HTTP requests and authentication using OAuth 2.0.

[Organization Service](https://learn.microsoft.com/en-us/power-apps/developer/data-platform/org-service/overview/) is a .NET SDK with .NET assemblies provided by Microsoft along with typed class generators for table classes. The APIs support their own approaches to building data queries in addition to supporting [FetchXML](https://learn.microsoft.com/en-us/power-apps/developer/data-platform/use-fetchxml-construct-query/). The Power Automate Dataverse connector also supports FetchXML. Organization service is the only option when using plug-ins to extend Dataverse.

Event Pipeline – EP8, when you perform an action like create a record in an app or create a record using the API, a Create message is processed by Dataverse. The message is processed in the event pipeline that provides a consistent set of stages that the message goes through. Each stage except for the main operation can have a plugin attached to it to run custom logic.



For more details on Event Framework, refer to [documentation](https://learn.microsoft.com/en-us/power-apps/developer/data-platform/event-framework).

Building Plugins – EP9, plugins are .NET classes that implement a **IPlugin** interface provided by the Dataverse SDK assemblies. That interface requires you implement only one method named *Execute*. Understanding of the execution context is fundamental for a plugin developer. Learn more about the [execution context](https://learn.microsoft.com/en-us/power-apps/developer/data-platform/understand-the-data-context/).

Tips: To have a plugin execute, it must be registered to run for a specific message. This can be accomplished using the Plugin Registration tool.

Custom APIs – EP10, operations in Dataverse are defined as messages.

Custom APIs offer a code-first way to define new messages that you can extend Dataverse web services. These messages can then be invoked, just like the system messages, but to execute custom business logic.

To define a new custom API, you start by creating a custom API record.

To implement the logic of the custom API, you create a plugin and register it on the Main Operation stage of the pipeline. Custom API implementation is the only scenario where a plugin can be registered on the Main Operation stage.

Once implemented you can use the custom message from the Dataverse APIs, Power Apps and Power Automate. To create a custom API and using it from a Power Automate flow, refer to <https://learn.microsoft.com/en-us/training/modules/introduction-power-platform-extend/exercise>

Configure vs Code – when to take advantage of Power Platform and when to write code. Things to consider includes:

1. Business rules vs Client Script
2. Workflows vs Power Automate flows vs Plugins
3. Use of PCF
4. Integration with External systems (Power Automate with custom connectors)
5. Portals vs Custom Sites

Graphical user interface, text, application, email

Description automatically generated