AB-400 · CHEAT SHEET · CERTIFYDECK

Microsoft Certified: Power Platform Developer Associate

Dominios, pesos oficiales y habilidades clave del temario de AB-400, para repasar antes del examen.

Nivel: Intermediate (Associate) Nota de corte: 700/1000 Fuente: guía de estudio oficial de Microsoft Learn

1. Build Microsoft Power Platform solutions

15–20%

Design the technical architecture

- Antes de escribir código, un desarrollador PL-400 debe traducir requisitos de negocio en componentes técnicos concretos: ¿esto es un plug-in, un flujo de Power Automate, un componente PCF o una integración externa? La arquitectura técnica documenta cada pieza, sus dependencias y el motivo de la elección, evitando sobre-ingeniería o soluciones frágiles.
- Cada componente (plug-in, conector personalizado, Azure Function, app) necesita una estrategia de identidad: usuarios interactivos vía OAuth/Microsoft Entra ID, o procesos desatendidos vía entidades de servicio o managed identities. La autorización se combina con roles de seguridad de Dataverse para limitar qué puede leer o escribir cada componente.
- El principio "low-code primero" del PL-400: antes de programar, evalúa si reglas de negocio, flujos de Power Automate, vistas o columnas calculadas ya resuelven el requisito sin código. Escribir un plug-in o componente personalizado solo se justifica cuando la plataforma no ofrece esa capacidad de forma nativa.
- La misma lógica de negocio se puede implementar en distintas capas: reglas de negocio (declarativas, ligeras), scripting cliente (validaciones de UI), plug-ins server-side (síncronos, transaccionales) o Power Automate (asíncrono, orquestación). Elegir la capa correcta afecta rendimiento, capacidad de mantenimiento y si la lógica se aplica también fuera de la interfaz.
- Standard tables almacenan datos dentro de Dataverse; virtual tables exponen datos externos como si fueran nativos sin duplicarlos; elastic tables están pensadas para volúmenes altos y datos semi-estructurados (telemetría, logs); los conectores acceden a datos externos bajo demanda sin modelarlos como tabla. La elección depende de volumen, latencia y necesidad de relaciones.

Design solution components

- Los componentes canvas (bloques reutilizables dentro del editor low-code) y los componentes PCF (code components, escritos en TypeScript) resuelven necesidades distintas: los primeros para lógica/UI reutilizable entre apps canvas, los segundos cuando se necesita control total del DOM, rendimiento o integración con librerías JS externas, incluso en apps model-driven.
- Un conector personalizado envuelve una API REST (propia o de terceros) en una definición OpenAPI para que Power Apps y Power Automate puedan consumirla como cualquier conector nativo. Diseñarlo implica decidir autenticación, operaciones expuestas y cómo se agrupan para no chocar con políticas DLP.
- Las funciones de Power Fx en Dataverse permiten extender columnas calculadas con lógica reutilizable; los plug-ins ejecutan código C# síncrono ante eventos de la plataforma; las custom APIs definen operaciones a medida invocables como mensajes de la Web API, con contrato de entrada/salida propio.
- Diseñar un flujo en lugar de codificarlo tiene sentido cuando la lógica es principalmente orquestación entre sistemas (esperar, aprobar, notificar) más que cómputo intensivo. El diseño debe considerar manejo de errores, concurrencia y qué parte de la lógica conviene delegar a un plug-in o Azure Function en vez de al propio flujo.
- Las integraciones salientes (outbound) suelen usar Service Endpoints/Webhooks o Azure Service Bus para notificar a sistemas externos cuando cambia un registro; las entrantes (inbound) usan la Web API, custom APIs o conectores para que sistemas externos escriban en Dataverse. La elección depende de si necesitas respuesta síncrona o desacoplamiento.

Implement application lifecycle management (ALM)

- Las soluciones de Dataverse registran automáticamente dependencias entre componentes (por ejemplo, un plug-in que referencia una columna concreta). Estas dependencias impiden eliminar o modificar un componente si otro depende de él, y determinan qué se debe incluir al exportar una solución para que funcione en otro entorno.
- Las variables de entorno externalizan valores que cambian entre entornos (URLs de API, claves, IDs de conexión) para que el mismo componente funcione en desarrollo, pruebas y producción sin modificar código. Se definen en la solución y se les asigna un valor distinto por entorno, evitando "hardcodear" configuración.
- Cuando varias soluciones (no gestionadas y gestionadas, de distintos proveedores) modifican el mismo componente, Dataverse las apila en capas y solo la capa superior activa (top layer) determina el comportamiento visible. Entender el apilado de capas es clave para depurar por qué un cambio "no se ve reflejado" o entra en conflicto con otra solución.
- Power Platform Pipelines automatiza el despliegue de soluciones entre entornos (dev ’ test ’ prod) directamente desde la interfaz de Power Platform, sin configurar Azure DevOps manualmente. Se pueden extender con etapas o comprobaciones personalizadas cuando el flujo de despliegue estándar no cubre necesidades específicas del proyecto.
- Power Platform Build Tools son tareas para Azure DevOps (y equivalentes en GitHub Actions) que exportan, empaquetan, versionan e importan soluciones automáticamente como parte de un pipeline de CI/CD, permitiendo aplicar controles de calidad (checker de soluciones, pruebas automatizadas) antes de llegar a producción.

2. Extend the user experience

25–30%

Apply business logic in model-driven apps by using client scripting

- El Client API expone objetos como `formContext` y `Xrm` para leer/escribir atributos, controlar visibilidad de campos, mostrar notificaciones o navegar en formularios model-driven. Escribir contra este modelo de objetos (en vez de manipular el DOM directamente) es la forma soportada y estable de scripting de cliente.
- Los manejadores de eventos (`OnLoad`, `OnChange`, `OnSave`) se pueden registrar de forma estática en el diseñador de formularios o de forma dinámica con `addOnChange`/`addOnSave` en tiempo de ejecución. La elección afecta si el evento sobrevive a la reapertura del formulario y qué tan fácil es depurarlo o eliminarlo.
- Desde JavaScript de cliente se pueden hacer llamadas asíncronas a la Web API (`Xrm.WebApi`) para leer o modificar registros relacionados sin recargar el formulario, por ejemplo validar un dato contra otra tabla antes de guardar. Es distinto de manipular solo los campos del formulario actual.
- El ribbon command bar de una app model-driven se personaliza con reglas de habilitación y acciones de comando, hoy configurables con Power Fx de forma declarativa o con JavaScript para lógica más compleja, controlando cuándo aparece un botón y qué ejecuta al pulsarlo.
- Las páginas personalizadas (custom pages) son lienzos canvas embebidos dentro de una app model-driven. Se navega a ellas programáticamente con `Xrm.Navigation.navigateTo`, pasando parámetros de entrada, útil para flujos guiados o interfaces a medida que el formulario estándar no puede ofrecer.

Create PCF code components

- Un componente PCF pasa por eventos de ciclo de vida bien definidos: `init` (una vez, al cargar), `updateView` (cada vez que cambian los datos), `getOutputs` (para devolver valores al framework) y `destroy` (limpieza). Implementarlos correctamente evita fugas de memoria y comportamientos inconsistentes.
- El archivo `ControlManifest.Input.xml` declara las propiedades (bound/unbound), recursos (scripts, CSS, imágenes) y características de la plataforma (Device, Utility, WebAPI) que necesita un componente PCF; es lo primero que Power Apps lee para saber cómo renderizarlo y qué datos pasarle.
- Un componente PCF implementa la interfaz `ComponentFramework.StandardControl` (o `ReactControl`), con métodos obligatorios como `init` y `updateView`, además de tipar correctamente los parámetros de entrada/salida definidos en el manifest para que el framework pueda comunicarse con el componente en tiempo de ejecución.
- Un componente PCF se empaqueta como solución (con `pac pcf push` o incluido en una solución de Dataverse), se despliega al entorno, y luego se "consume" añadiéndolo a un campo, formulario o app canvas concreta desde el diseñador correspondiente.
- El objeto de contexto (`context`) de un PCF expone capacidades adicionales si se declaran en el manifest: `context.device` (cámara, geolocalización, micrófono), `context.utils` (formateo, diálogos) y `context.webAPI` (lectura/escritura en Dataverse), ampliando lo que un componente puede hacer más allá de renderizar datos.

Build Power Apps code apps

- Para conectar una code app a un conector se usa la CLI: `pa connector list`, `pa connection create` y `pa app add data-source` (con `--table`/`--dataset` si es tabular, como SQL o SharePoint), que genera un modelo y servicio TypeScript. Para portabilidad entre entornos se usa una referencia de conexión (`--connection-ref`/`--solution-id`); no todos los conectores están soportados, como Excel Online.
- Para usar Dataverse como origen de datos se ejecuta `pa app add data-source --connector dataverse --table <nombre-lógico>` en un entorno con Dataverse habilitado, generando un modelo y servicio TypeScript. El servicio admite crear, leer, actualizar y eliminar, con delegación de filtro, orden y `top`, y paginación. No soportados: lookups polimórficos, CRUD de metadatos de entidad, FetchXML y claves alternativas.
- El servicio de tabla expone `create`, `get`, `getAll`, `update` y `delete`; `getAll` acepta `IGetAllOptions` (select, filter, orderBy, top, skipToken) y conviene limitar columnas con `select`. Al actualizar solo se incluyen las propiedades que cambian. `getMetadata` recupera metadatos de entidad, columnas y relaciones, pero no propiedades de tipos derivados (opciones de un choice), los lookups se asocian con `@odata.bind`.
- Para llamar a una acción o función de Dataverse, se busca con `pa app find-dataverse-api --search <nombre>` (coincidencia de subcadena sobre el nombre) y se añade con `pa app add dataverse-api --api-name <nombre>`, que genera un modelo y servicio TypeScript. En una acción enlazada (p. ej. `AddToQueue`), el primer argumento del servicio es el ID del registro sobre el que actúa; las funciones no enlazadas, como `WhoAmI`, no lo requieren.
- Las code apps solo pueden invocar flujos de nube instantáneos, dentro de una solución, que usan el desencadenador de Power Apps; los flujos programados o automatizados no son compatibles. Se añaden con `pa app add flow --flow-id <id>`, que genera un servicio TypeScript con un método `Run`. El creador necesita acceso al flujo y sus conexiones; los usuarios finales necesitan permisos de Dataverse, como App Opener.

Deploy and manage Power Apps code apps

- La Content Security Policy (CSP) de un code app se configura a nivel de entorno en el Power Platform admin center (Settings > Product > Privacy + Security > Content security policy > App), o mediante la Power Platform API. Por defecto connect-src es 'none', bloqueando llamadas salientes no permitidas. Conviene activar solo Enable reporting y probar antes de forzar la aplicación con Enforce content security policy.
- Para monitorizar un code app con Azure Application Insights se instala `@microsoft/applicationinsights-web` y se registra un logger con `setConfig` de `@microsoft/power-apps/app`, que recibe métricas como sessionLoadSummary y networkRequest. Si la CSP está activa, hay que añadir el endpoint a connect-src. Application Insights solo captura telemetría tras la carga correcta; los fallos de arranque solo se ven en Monitor.
- Un code app se compila con `npm run build` y se publica con `pa app push` (Power Apps CLI). Un administrador debe activar antes la función en el entorno (Settings > Product > Features > Enable code apps), y los usuarios finales necesitan licencia Power Apps Premium. `pa app push` añade la app a una solución de Dataverse para el ALM entre entornos, y `pa app share` concede acceso play o edit.
- Para depurar un code app se usan las herramientas del navegador (F12), filtrando Network por apihub.net (conectores) o dynamics.com (Dataverse). El área Monitor del admin center ofrece métricas como Time to interactive y Data request success rate/latency. Application Insights solo registra telemetría tras la carga correcta; los fallos de arranque solo se ven en Monitor.
- El rendimiento de un code app se sigue en Monitor con las métricas Data request latency y Data request success rate; ante muchos errores 429 se recomienda reducir las llamadas a datos. Sus llamadas a la Web API de Dataverse comparten los límites de protección del servicio (solicitudes, tiempo de ejecución y solicitudes simultáneas en cinco minutos), y los batches muy grandes aumentan el riesgo de superar el límite de tiempo de ejecución.

3. Extend Microsoft Power Platform

35–40%

Create a Dataverse plug-in

- El pipeline de eventos de Dataverse tiene 4 etapas: PreValidation (10, antes de la transacción, incluso antes del bloqueo de seguridad), PreOperation (20, dentro de la transacción, ideal para modificar datos antes de guardar), MainOperation (30, reservada a la plataforma) y PostOperation (40, después de guardar, para efectos secundarios).
- El `IPluginExecutionContext` da acceso a la entidad objetivo, el mensaje (Create, Update...), el usuario que ejecuta la acción, la profundidad de la ejecución (para evitar bucles infinitos) y la colección de parámetros de entrada/salida, siendo el punto de partida de cualquier plug-in.
- Las Pre Images capturan el estado de un registro *antes* de la operación (útil en Update/Delete para comparar valores anteriores) y las Post Images el estado *después*. Se configuran al registrar el paso del plug-in y evitan tener que hacer una consulta adicional a Dataverse para obtener esos datos.
- Dentro de un plug-in, `IOrganizationService` (obtenido vía `IOrganizationServiceFactory`) permite hacer Create, Update, Delete o Retrieve contra Dataverse, respetando automáticamente el contexto de seguridad del usuario que disparó el evento (o el de un usuario suplantado si es necesario).

Perform operations by using platform APIs

- La Web API es una interfaz RESTful/OData para operar sobre Dataverse desde fuera de la plataforma (o desde JavaScript de cliente): GET para consultar con `$select`/`$filter`/`$expand`, POST para crear, PATCH para actualizar y DELETE para eliminar, usando URLs del tipo `/api/data/v9.2/entidades(id)`.
- El SDK .NET (`ServiceClient`) es la forma recomendada para interactuar con Dataverse desde código de servidor: plug-ins, consolas o Azure Functions. Ofrece los mismos mensajes que la Web API pero con tipado fuerte y gestión de conexión más cómoda.
- Dataverse aplica límites de protección de servicio por usuario/aplicación en una ventana de tiempo; al superarlos, devuelve un error 429 con una cabecera `Retry-After`. Un cliente robusto debe capturar ese código, esperar el tiempo indicado y reintentar con backoff exponencial en lugar de fallar inmediatamente.
- Para grandes volúmenes conviene usar `ExecuteMultiple` o `$batch` (Web API) en lugar de llamadas individuales, agrupar operaciones relacionadas en una misma transacción cuando deben tener éxito o fallar juntas, y acceder preferentemente a Dataverse (vía Organization Service) desde el mismo entorno para reducir latencia.
- El acceso programático a Dataverse (Web API u Organization Service) requiere un token OAuth 2.0 emitido por Microsoft Entra ID, obtenido mediante flujo de credenciales de cliente (procesos desatendidos) o flujo interactivo (usuario final), y enviado como cabecera `Authorization: Bearer` en cada solicitud.

Process workloads by using Microsoft Azure Functions

- Cuando una operación excede los límites de tiempo de un plug-in síncrono (2 minutos) o de un flujo, se delega a una Azure Function, que puede ejecutar procesos más largos de forma asíncrona (o con el patrón durable) y notificar el resultado de vuelta a Dataverse o al flujo llamante.
- Además de invocarse vía HTTP desde un flujo o plug-in, una Azure Function puede dispararse por un temporizador (Timer trigger, para tareas programadas como limpiezas nocturnas) o por eventos de otros servicios de Azure (Service Bus, Event Grid...), integrándose con Power Platform a través de conectores.
- Para acceder a Dataverse sin almacenar secretos ni claves en el código: Azure gestiona las credenciales automáticamente y solo hay que conceder a esa identidad los permisos necesarios como usuario de aplicación en Power Platform.

Configure Power Automate cloud flows and Copilot Studio workflows

- El lenguaje de expresiones de Power Automate (basado en funciones tipo `if()`, `coalesce()`, `formatDateTime()`, referencias a `triggerBody()` u outputs de pasos anteriores) permite lógica condicional y transformación de datos dentro de los pasos, sin necesidad de una acción aparte.
- Los ámbitos (Scope) junto con la configuración "ejecutar después" (run after) de los pasos (failed, timed out, skipped) permiten construir bloques try/catch: si un paso falla, otro ámbito paralelo captura el error, lo registra o notifica, en lugar de dejar que el flujo termine en fallo.
- El filtro de disparador (trigger condition) en Dataverse evita ejecuciones del flujo cuando el cambio no cumple una condición concreta, ahorrando ejecuciones innecesarias; la política de reintento de cada acción define cuántas veces y con qué intervalo se reintenta en caso de fallo transitorio.
- Para reutilizar lógica entre flujos se crea un flujo "hijo" (child flow) que se llama desde el padre con "Run a Child Flow", pasando entradas y recibiendo salidas; es el equivalente a una función o subrutina.
- Para que un flujo se autentique contra una API o servicio de Azure sin usar la identidad de un usuario concreto, se registra una entidad de servicio (service principal) en Microsoft Entra ID con permisos específicos, y el flujo usa sus credenciales (client ID/secret o certificado) para las llamadas HTTP.

Build Microsoft Foundry agents by using code that integrates with the Microsoft Power Platform

- Dataverse actúa como servidor MCP remoto en https://{org}.crm.dynamics.com/api/mcp, con herramientas como read_query, search_data, describe, create_record y upsert_skill. Copilot Studio accede por defecto; otros clientes deben habilitarse en el centro de administración de Power Platform, y el punto de conexión remoto directo necesita una app de Entra con el permiso mcp.tools. El servidor respeta los roles de seguridad de Dataverse.
- Un agente de Copilot Studio puede conectarse a otro agente de Copilot Studio del mismo entorno, o a agentes externos como Microsoft Foundry, Microsoft 365 Agents SDK o agentes A2A. Conectar un agente de Foundry exige tener habilitado el endpoint de protocolo Activity; conectar uno del Agents SDK requiere su URL de mensajería. Un agente principal con agentes conectados propios no puede a su vez ser agente conectado de otro agente principal.
- Work IQ es la capa de inteligencia que da a Microsoft 365 Copilot y a los agentes contexto compartido sobre archivos, reuniones, mensajes y sistemas de negocio. Dataverse intelligence se apoya en esa base para que los agentes actúen sobre datos de Dataverse mediante contexto reutilizable. El catálogo Work IQ MCP incluye un servidor Dataverse and Dynamics 365 con operaciones CRUD, gobernado desde el centro de administración de Microsoft 365.
- El cliente de Copilot Studio del Microsoft 365 Agents SDK invoca desde código, en .NET, JavaScript o Python, un agente que sigue ejecutándose de forma remota. La conexión se configura con una connection string de la página Channels, o con Environment ID, Tenant ID y Schema name, más una app de Entra con el permiso Copilot Studio.Copilots.Invoke. Si la app llamante usa un token de entidad de servicio, debe usarse Direct Line en su lugar.

4. Develop integrations

10–15%

Publish and consume Dataverse events

- Desde un plug-in registrado en un service endpoint, `IServiceEndpointNotificationService` permite reenviar el mensaje del evento actual (con su contexto de ejecución) hacia Azure Service Bus o un webhook de forma explícita en código, en vez de depender únicamente de la configuración declarativa del paso.
- La forma más común de publicar eventos es registrar un "Service Endpoint" en la Plug-in Registration Tool (apuntando a un webhook, Service Bus o Event Hub) y luego un paso que lo asocia a un mensaje/entidad concreta, sin necesidad de escribir código de publicación manualmente.
- Cada tipo de endpoint tiene un caso de uso distinto: webhooks para notificar directamente a un único receptor HTTP, Service Bus para mensajería confiable punto a punto o con colas, y Event Hub cuando se necesita ingesta de eventos a gran escala (telemetría, streaming).
- Para reaccionar a cambios en Dataverse desde fuera de la plataforma hay varias opciones según el escenario: service endpoints/webhooks para notificación push en tiempo real, Power Automate como capa de orquestación de bajo código, o sondeo con change tracking cuando no se necesita reacción inmediata.

Implement synchronization for Dataverse data

- El change tracking permite a un sistema externo pedir solo los registros que cambiaron desde la última sincronización (usando un token de continuación) en lugar de descargar toda la tabla cada vez, siendo la base de escenarios de sincronización incremental eficiente con Dataverse.
- Las claves alternativas permiten identificar y referenciar un registro de Dataverse mediante un identificador de negocio (por ejemplo, un código de producto) en lugar del GUID interno, muy útil en integraciones donde el sistema externo no conoce ese GUID.
- `UpsertRequest` crea el registro si no existe o lo actualiza si ya existe (identificándolo normalmente por una clave alternativa), evitando tener que hacer primero un Retrieve para decidir manualmente entre Create y Update en escenarios de sincronización con sistemas externos.

Create custom connectors

- Un conector personalizado soporta varios esquemas de autenticación: API Key, Basic, OAuth 2.0 (el más común para APIs empresariales) o Azure AD/Entra ID. Configurarlo correctamente implica registrar la app cliente en el proveedor de identidad y mapear los parámetros de token en la definición del conector.
- Las policy templates son transformaciones que se aplican en tiempo de ejecución sin tocar el backend: añadir cabeceras, reescribir la URL, establecer valores por defecto o convertir el formato de fecha/hora, útiles para adaptar una API existente a las expectativas de Power Platform.
- En lugar de escribir la definición desde cero, se puede importar directamente desde un archivo OpenAPI, desde un recurso de Azure (como una Function App con Swagger habilitado) o desde un repositorio de conectores certificados en GitHub, acelerando la creación del conector.
- Servicios de Azure como Functions, Logic Apps o API Management pueden exponerse directamente como conector personalizado desde el propio portal de Azure o importando su definición OpenAPI, manteniendo la autenticación gestionada (managed identity o claves de Azure) integrada con Power Platform.
- Una Azure Function puede actuar como backend de un conector personalizado: recibe la llamada del conector, ejecuta lógica (transformación, agregación, llamada a otro sistema) y devuelve una respuesta JSON. Debe exponer su propia definición OpenAPI para que el asistente de creación de conectores la reconozca.