PL-400 · CHEAT SHEET · CERTIFYDECK

Microsoft Power Platform Developer

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

Nivel: Intermediate (Associate) Duración: 100 min Nota de corte: 700/1000 Fuente: guía de estudio oficial de Microsoft Learn

1. Create a technical design

10–15%

Design 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.

2. Build Power Platform solutions

10–15%

Configure and troubleshoot Microsoft Power Platform

- Cuando un usuario o un componente recibe un error "insufficient privileges" o un plug-in falla en producción por permisos, el diagnóstico pasa por revisar roles de seguridad asignados, la unidad de negocio del registro, las reglas de uso compartido (sharing) y si el componente se ejecuta con el contexto del usuario o con SYSTEM. Las herramientas de auditoría y el Plug-in Trace Log ayudan a aislar la causa.
- Un componente de código nunca debería ejecutarse con más privilegios de los estrictamente necesarios. Esto implica crear roles de seguridad dedicados (no reutilizar el de System Administrator), limitar el acceso a las tablas y niveles de acceso (usuario, unidad de negocio, organización) que realmente necesita el plug-in, conector o flujo.
- El ciclo de vida de un desarrollador pasa por entornos separados (desarrollo, pruebas, producción), cada uno con su propia instancia de Dataverse. Gestionar entornos incluye crear entornos de tipo Sandbox/Developer, controlar quién tiene acceso, y decidir cuándo copiar datos de producción a un entorno inferior para pruebas realistas.

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.

3. Implement Power Apps improvements

10–15%

Implement advanced canvas apps features

- Más allá de fórmulas básicas, un desarrollador PL-400 debe dominar funciones de Power Fx para manipular colecciones (`Filter`, `LookUp`, `AddColumns`), variables de contexto/globales, y funciones de comportamiento encadenadas para lógica condicional compleja dentro de apps canvas, manteniendo la fórmula legible y con buen rendimiento.
- Una component library agrupa componentes canvas reutilizables (controles, lógica, estilos) en un artefacto independiente que se puede importar en múltiples apps canvas, evitando duplicar el mismo control en cada aplicación y facilitando actualizarlo de forma centralizada.
- Cuando una app canvas necesita operaciones que exceden Power Fx (llamadas a APIs externas, procesos de larga duración, lógica compartida con otros clientes), delega esa lógica a un flujo de Power Automate invocado directamente desde la app, pasando parámetros y recibiendo una respuesta estructurada.

Optimize and troubleshoot apps

- Monitor captura en tiempo real las llamadas de red, evaluaciones de fórmulas y errores de una app canvas o model-driven mientras se ejecuta, permitiendo identificar exactamente qué fórmula falla o qué llamada a Dataverse tarda demasiado, complementado con las herramientas de desarrollador del navegador para inspeccionar red y consola.
- La delegación permite que Dataverse (u otro origen de datos) procese el filtrado/ordenación en el servidor en lugar de traer todos los registros al dispositivo; superar el límite de delegación produce resultados incompletos silenciosamente. Precargar datos críticos al abrir la app (en lugar de bajo demanda) reduce la latencia percibida por el usuario.
- Formularios con demasiados campos, sub-grids o reglas de negocio síncronas ralentizan la carga; vistas con columnas ordenadas por campos no indexados degradan el rendimiento de las búsquedas. Optimizar implica simplificar formularios, cargar componentes bajo demanda y revisar qué scripts se ejecutan en el evento `OnLoad`.

4. Extend the user experience

10–15%

Apply business logic in model-driven apps 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, definidas en Power Apps 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 enfocados a tareas que el formulario estándar no puede ofrecer.

Create a Power Apps component framework (PCF) code component

- 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 del control (p. ej. `WebAPI`) que el componente PCF necesita. 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.

5. Extend the platform

30–35%

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 prácticamente 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 de un usuario suplantado si es necesario).

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 integrado. 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.

Use 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`/`$expand`, POST para crear, PATCH para actualizar y DELETE para eliminar, usando el nombre del set de entidades en plural.
- El Organization service es el SDK .NET (SOAP internamente) para interactuar con Dataverse desde código de servidor, plug-ins, consolas o Azure Functions. Ofrece los mismos mensajes que la Web API pero con tipado fuerte en C# a través de `IOrganizationService`.
- Dataverse aplica límites de solicitudes por usuario/aplicación en una ventana de tiempo; al superarlos, devuelve un error 429 con una cabecera `Retry-After`. Un cliente robusto debe capturar ese código, esperar el tiempo indicado y reintentar con backoff exponencial en lugar de fallar directamente.
- Para grandes volúmenes conviene usar `ExecuteMultipleRequest` o `$batch` (Web API) en lugar de llamadas individuales, agrupar operaciones relacionadas en una misma transacción cuando deben tener éxito o fallar juntas, y controlar la concurrencia optimista con ETags para evitar sobrescrituras accidentales.
- 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). Se envía en la cabecera `Authorization: Bearer` en cada solicitud.

Process workloads by using 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 Power Platform cuando termina.
- 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 de forma desacoplada.
- Una managed identity permite que una Azure Function se autentique contra 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

- El conector de Dataverse ofrece triggers ("cuando se crea/actualiza/elimina una fila", con filtro opcional por columnas) y acciones (crear, actualizar, listar filas, ejecutar acción/custom API). Configurar bien el filtro del trigger evita que el flujo se dispare innecesariamente o entre en bucles con sus propias actualizaciones.
- El lenguaje de expresiones de Power Automate (basado en funciones tipo `if()`, `coalesce()`, `formatDateTime()`, referencias a entradas/salidas de pasos anteriores) permite lógica condicional y transformación de datos dentro de un paso sin necesidad de una acción adicional, aunque en exceso dificulta el mantenimiento del flujo.
- Los parámetros marcados como "seguros" (`secureInputs`/`secureOutputs`) en un paso ocultan su contenido en el historial de ejecución del flujo, evitando que contraseñas, tokens o datos personales queden expuestos en los logs para cualquiera con acceso de lectura al flujo.
- En lugar de guardar secretos directamente en el flujo, Azure Key Vault los almacena de forma centralizada y cifrada; el flujo los recupera en tiempo de ejecución mediante el conector de Key Vault, facilitando rotar credenciales sin tener que editar cada flujo que las usa.
- Los ámbitos (`Scope`) junto con la configuración "ejecutar después" (run after: has failed, has timed out, is 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 simplemente falle sin control.

6. 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 data synchronization with Dataverse

- 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.