Multi-Cloud Engineering Program

Parte 03 — Azure: plataforma esencial

12 clases · 52 horas

Cuaderno de una sola parte: su introducción, sus clases completas y sus evaluaciones.

Extracto del manual integral, generado desde las mismas fuentes versionadas.

Alcance de este cuaderno

Componente	Cobertura
Parte	03 de 23
Clases	037–048
Evaluaciones	12
Horas estimadas	52

Índice

Parte 03 — Azure: plataforma esencial

← Parte 02 · Índice completo · Parte 04 →

Descargar: Esta parte en PDF · Recorrido de Azure en PDF · Manual integral

Nivel: intermedio · Clases: 12 · Duración sugerida: 6–8 semanas

Diseñar y operar una carga empresarial en Azure desde su jerarquía de gobierno.

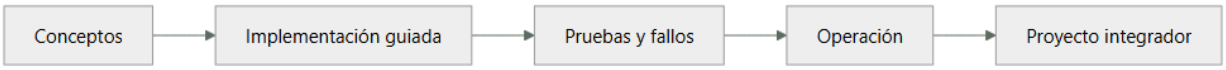
Resultados de la parte

Al completar esta parte podrás explicar el dominio con lenguaje independiente del proveedor, implementar sus mecanismos esenciales, diagnosticar fallos y defender decisiones mediante evidencia, costo, seguridad y objetivos de confiabilidad.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Secuencia



Clases

ID	Clase	Laboratorio	Horas
037	Tenant, management groups, suscripciones y resource groups	governance	4
038	Microsoft Entra ID, RBAC, managed identities y PIM	iam	4
039	Virtual Network, subredes, NSG, UDR, peering y Private Link	network	4
040	Virtual Machines, Scale Sets y Load Balancer	compute	4
041	Blob Storage, Files, redundancia y lifecycle	storage	4
042	Azure SQL, Cosmos DB y Azure Cache for Redis	data	4
043	App Service, Functions y Container Apps	serverless	4
044	Service Bus, Event Grid y Event Hubs	messaging	4
045	Azure Monitor, Log Analytics y Application Insights	observability	4
046	Key Vault, Defender for Cloud y Azure Policy	security	4
047	Bicep, plantillas y despliegues por alcance	iac	4
048	Proyecto: aplicación de tres capas en Azure	capstone	8

Evaluación de la parte

- 35 % laboratorios y evidencia reproducible.
- 25 % retos verificables.
- 20 % decisiones y ADR.
- 10 % seguridad, confiabilidad y costo.
- 10 % proyecto integrador de la parte.

Se aprueba con 80 % de clases en nivel B o superior y el proyecto integrador reproducible.

Pauta bibliográfica

- Azure Architecture Center — Microsoft.
- Azure for Architects — Ritesh Modi.
- Exam Ref AZ-305 — Microsoft Press.

037 — Tenant, management groups, suscripciones y resource groups

Parte: 03 — Azure: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: governance · Estado: EXECUTABLECORE

Propósito

Construir la jerarquía de Azure y descubrir en qué se parece y en qué no a la de AWS. El ejercicio central de esta parte no es aprender nombres nuevos: es distinguir qué decisiones de la parte 02 eran de arquitectura —y reaparecen— y cuáles eran de proveedor, que es la pregunta con la que cerró la clase 036.

Resultados de aprendizaje

Al finalizar podrás:

1. Traducir la jerarquía de Azure a los conceptos neutrales de la clase 017 y señalar dónde no hay equivalencia.
2. Explicar por qué el grupo de recursos no es una frontera de aislamiento y qué sí lo es.
3. Escribir una directiva que restrinja regiones sin bloquear los recursos globales.
4. Distinguir cuota de suscripción de límite de servicio y anticipar cuál se agota antes.
5. Diseñar grupos de administración por régimen de gobierno, no por organigrama.

Conceptos centrales

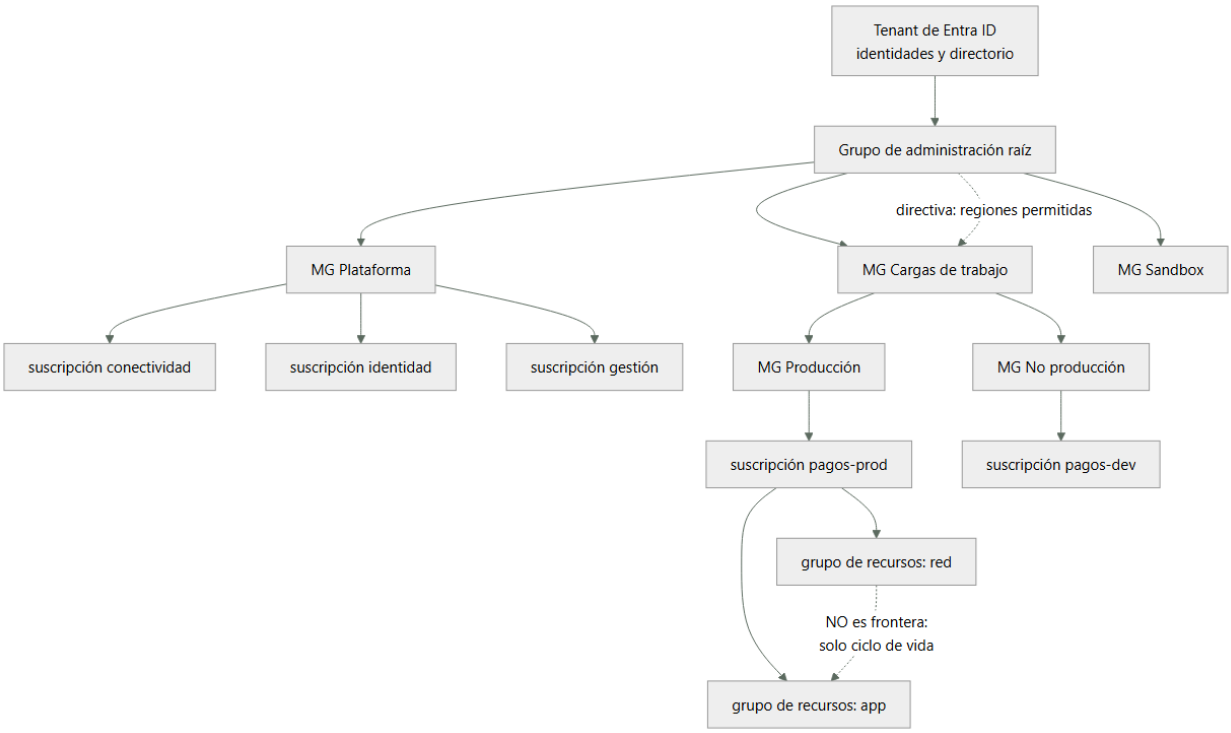
Concepto	Comprensión verificable
tenant	Instancia de Microsoft Entra ID que contiene identidades y suscripciones. Es la raíz de la jerarquía y el equivalente conceptual de la organización, con una diferencia importante: la identidad vive ahí, no en la suscripción.
suscripción	Frontera de aislamiento, facturación y cuota. Es el equivalente funcional de la cuenta de AWS y del proyecto de Google Cloud.
grupo de recursos	Contenedor de recursos con ciclo de vida común dentro de una suscripción. No aísla nada: es unidad de despliegue y de borrado, y confundirlo con frontera de seguridad es el error más frecuente al llegar de AWS.
directiva	Regla que evalúa recursos y puede auditar, denegar, modificar o desplegar. A diferencia de una SCP, puede corregir además de prohibir, y se aplica también a lo que ya existe.
grupo de administración	Agrupador de suscripciones sobre el que se heredan directivas y asignaciones de rol. Admite hasta seis niveles de profundidad bajo la raíz.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Aplicado a esta clase, separa siempre cuatro planos: intención (qué necesita el usuario), configuración (qué declaramos), estado observado (qué existe de verdad) y evidencia (cómo sabemos que cumple). Confundirlos produce diseños que se ven correctos en un diagrama pero fallan al operar.

■ Flujo de razonamiento



Desarrollo

1. El mapa de equivalencias, y dónde se rompe

Concepto neutral (clase 017)	AWS	Azure	Google Cloud
Raíz	Organización	Tenant	Organización
Agrupador	Unidad organizativa	Grupo de administración	Carpeta
Frontera de aislamiento	Cuenta	Suscripción	Proyecto
Agrupador interno	—	Grupo de recursos	—
Política heredada	SCP	Azure Policy	Restricción de organización

Dos diferencias cambian el diseño y no son cosméticas:

La identidad vive en el tenant, no en la suscripción. En AWS cada cuenta tiene su propio IAM y el acceso entre cuentas exige asumir roles. En Azure, un usuario del tenant puede tener asignaciones de rol en varias suscripciones sin ningún salto: la frontera de identidad y la de aislamiento no coinciden. Eso simplifica la operación y hace que una identidad comprometida alcance potencialmente más superficie, así que la separación por asignaciones de rol importa más que en AWS.

Existe un nivel intermedio que AWS no tiene. El grupo de recursos agrupa recursos con ciclo de vida común. Su propiedad más útil es que borrarlo borra todo lo que contiene, lo que lo hace excelente para entornos efímeros y peligroso si se confunde con una frontera de seguridad.

Y una asimetría de las directivas frente a las SCP: Azure Policy puede modificar y desplegar, no solo denegar. Una directiva con efecto DeployIfNotExists puede añadir el diagnóstico que falta a un recurso recién creado. Es más potente y exige más cuidado: una directiva de corrección mal escrita cambia recursos en producción sin que nadie lo pida.

2. El grupo de recursos no aísla

Es el error de traducción más común al llegar de AWS. Un grupo de recursos parece una cuenta pequeña y no lo es:

```
grupo de recursos
  ✓ ciclo de vida común: se borra entero
  ✓ ámbito de asignación de roles
  ✓ ámbito de aplicación de directivas
```

```
✗ NO tiene cuotas propias
✗ NO es frontera de facturación
✗ NO impide que un recurso hable con otro de otro grupo
```

Las dos primeras exclusiones son las que producen incidentes. Las cuotas son de la suscripción, así que dos grupos de recursos en la misma suscripción compiten por los mismos núcleos disponibles — exactamente el problema de la clase 017 con otro nombre:

```
$ az vm list-usage --location brazilsouth --query "[?contains(name.value,'cores')].{n:name.value,u:
currentValue,l:limit}" -o table
n                       u     l
----------------------  ----  ----
cores                   186   200
standardDSv3Family      142   150
```

186 de 200 núcleos consumidos en la suscripción, sin importar en cuántos grupos de recursos estén repartidos. Un experimento en el grupo «desarrollo» deja sin capacidad al grupo «producción» si comparten suscripción.

La tercera exclusión también sorprende: no hay aislamiento de red entre grupos de recursos. Dos máquinas en la misma red virtual se hablan aunque estén en grupos distintos; el aislamiento de red lo dan la red virtual y los grupos de seguridad, no la organización de recursos.

La regla de traducción: donde en AWS separarías por cuenta, en Azure separa por suscripción. El grupo de recursos es la unidad de despliegue, no de aislamiento.

3. Directivas: cuatro efectos y cuándo usar cada uno

Azure Policy va más allá de prohibir:

Efecto	Qué hace	Cuándo
Audit	Marca no conforme, no impide	Siempre primero: medir antes de bloquear
Deny	Impide la creación o el cambio	Controles duros, tras medir
Modify	Añade o cambia propiedades	Etiquetas obligatorias
DeployIfNotExists	Despliega un recurso asociado	Diagnóstico, agentes, copias

La disciplina es la misma que con el WAF de la clase 035: empezar en Audit. Una directiva Deny aplicada sin medir bloquea despliegues legítimos, y el equipo la desactiva en el primer incidente en vez de corregirla.

```
$ az policy assignment create --name regiones-permitidas \
    --scope /providers/Microsoft.Management/managementGroups/mg-cargas \
    --policy "e56962a6-4747-49cd-b67b-bf8b01975c4c" \
    --params '{"listOfAllowedLocations":{"value":["brazilsouth","eastus"]}}' \
    --enforcement-mode DoNotEnforce      # equivale a Audit
```

Y la restricción de regiones tiene aquí el mismo problema que en la clase 025, con distinta solución. Varios tipos de recurso son globales y su ubicación declarada es global:

```
Microsoft.Network/trafficManagerProfiles
Microsoft.Network/dnsZones
Microsoft.Cdn/profiles
Microsoft.ManagedIdentity/userAssignedIdentities  (en algunos casos)
```

La directiva integrada de ubicaciones permitidas ya excluye los recursos globales, pero una escrita a mano no. Comprobarlo antes de aplicarla evita bloquear la creación de zonas DNS sin entender por qué.

Y un detalle operativo: las directivas evalúan lo existente, no solo lo nuevo. Tras asignar una, aparece un informe de conformidad de todo lo que ya estaba. Es una ventaja sobre las SCP —que solo actúan sobre peticiones nuevas— y también una fuente de ruido inicial que hay que triar.

4. Cuotas: dos límites que se confunden

Azure tiene dos clases de límite y se agotan por motivos distintos:

```
cuota de suscripción    núcleos por familia y región, IP públicas, redes virtuales
                        → ampliable con solicitud, tarda horas o días

límite de servicio      reglas por grupo de seguridad, tamaño de plantilla,
                        recursos por grupo de recursos
```

```
                          → normalmente NO ampliable
```

La primera es la que aparece en el autoescalado y la segunda en el despliegue. Un caso frecuente del segundo tipo:

```
límite de reglas por grupo de seguridad de red: 1.000
un equipo genera una regla por cliente autorizado
→ el despliegue falla al llegar a 1.000 y NO se puede ampliar
→ la corrección exige rediseñar: grupos de seguridad de aplicación
  o listas de prefijos en vez de reglas individuales
```

La consulta de cuotas debe formar parte del plan de capacidad, no de la respuesta a incidentes:

```
$ az vm list-usage --location brazilsouth -o json \
  | jq -r '.[] | select((.currentValue / .limit) > 0.8)
           | "\(.name.value): \(.currentValue)/\(.limit)"'
standardDSv3Family: 142/150
```

El 80 % como umbral de alerta, igual que en la clase 017. Ampliar una cuota tarda horas o días, así que descubrirla durante un pico significa que ya no es una palanca disponible.

Y una diferencia con AWS que conviene saber: en Azure las cuotas de núcleos son por familia de máquina virtual además de por total. Tener 60 núcleos libres del total no sirve si la familia concreta que necesitas está al límite, y el mensaje de error no siempre lo deja claro.

5. Grupos de administración por gobierno, no por organigrama

El mismo criterio de la clase 017: agrupar por qué directiva se aplica, porque los organigramas se reorganizan y mover suscripciones entre grupos cambia las directivas heredadas.

La estructura de referencia de Microsoft para escala empresarial:

```
raíz
■■■ Plataforma           directivas estrictas, la opera el equipo central
■   ■■■ conectividad     redes virtuales de concentrador, DNS privado
■   ■■■ identidad        controladores de dominio, servicios de identidad
■   ■■■ gestión          registros, copias, automatización
■■■ Cargas de trabajo
■   ■■■ Producción       regiones, cifrado obligatorio, diagnóstico forzado
■   ■■■ No producción    regiones, límite de gasto
■■■ Sandbox              directivas laxas, presupuesto duro, caducidad
■■■ Desmantelado         suscripciones en retirada, solo lectura
```

Dos elementos que AWS no tiene explícitos y aquí conviene aprovechar:

El grupo «Desmantelado» recoge suscripciones que se van a cerrar, con directivas que impiden crear nada nuevo. Evita que una suscripción en retirada siga acumulando recursos durante meses.

La separación de plataforma en tres suscripciones —conectividad, identidad y gestión— responde a que cada una tiene un ciclo de vida y un equipo distintos. Es el equivalente a las cuentas de red, seguridad y auditoría de la clase 025.

Y una restricción práctica: la profundidad máxima es de seis niveles bajo la raíz. Suficiente, pero conviene no gastarlos en reflejar jerarquías organizativas que cambiarán.

Un recurso que no debe alojar cargas: el grupo de administración raíz. Igual que la cuenta de gestión de AWS, es el único punto sin nada por encima, y las asignaciones que se hacen ahí se heredan a todo el tenant.

Ejemplo trabajado

CloudShop despliega en Azure la misma aplicación de la parte 02. El equipo replica la estructura de AWS literalmente: un grupo de recursos por lo que allí era una cuenta. A las tres semanas, un despliegue de pruebas deja producción sin capacidad.

El incidente, idéntico al de la clase 017 con otro vocabulario:

```
$ az vm list-usage --location brazilsouth \
    --query "[?name.value=='standardDSv3Family'].{u:currentValue,l:limit}" -o tsv
148   150
$ az group list --query "[].name" -o tsv
rg-cloudshop-prod
rg-cloudshop-dev
rg-cloudshop-red
```

Tres grupos de recursos en una sola suscripción. El equipo tradujo «cuenta» por «grupo de recursos» y las cuotas no lo respetan:

```
cuota de la familia DSv3      150 núcleos
producción en régimen          88
experimento en rg-dev          60
total                         148 → producción no puede escalar
```

Se verifica que el grupo de recursos tampoco aísla la red:

```
$ az network vnet subnet list -g rg-cloudshop-red --vnet-name vnet-cloudshop \
    --query "[].name" -o tsv
snet-prod
snet-dev
# una VM de rg-dev en snet-dev alcanza una de rg-prod en snet-prod:
$ az network watcher test-ip-flow --vm vm-dev-01 --direction Outbound \
    --local 10.30.2.4:0 --remote 10.30.1.7:5432 --protocol TCP -o tsv --query access
Allow
```

Ni cuotas ni red separadas. La traducción era incorrecta.

Rediseño con la equivalencia correcta:

```
tenant cloudshop
■■■ mg-raiz
    ■■■ mg-plataforma
    ■   ■■■ sub-conectividad      red de concentrador, DNS privado
    ■■■ mg-cargas
    ■   ■■■ mg-produccion
    ■   ■   ■■■ sub-cloudshop-prod    cuota propia
    ■   ■■■ mg-no-produccion
    ■       ■■■ sub-cloudshop-dev     cuota propia
    ■■■ mg-sandbox
        ■■■ sub-lab-*                 presupuesto duro
```

Directiva de regiones, primero en modo auditoría:

```
$ az policy assignment create --name regiones --display-name "Regiones permitidas" \
    --scope /providers/Microsoft.Management/managementGroups/mg-cargas \
    --policy e56962a6-4747-49cd-b67b-bf8b01975c4c \
    --params '{"listOfAllowedLocations":{"value":["brazilsouth","eastus2"]}}' \
    --enforcement-mode DoNotEnforce
```

Tras 7 días, el informe de conformidad revela algo que nadie había notado:

```
$ az policy state summarize --management-group mg-cargas \
    --query "policyAssignments[0].results.{conformes:compliantResources,no:nonCompliantResources}"
{"conformes": 214, "no": 9}
$ az policy state list --management-group mg-cargas --filter "complianceState eq 'NonCompliant'" \
    --query "[].{r:resourceId,loc:resourceLocation}" -o tsv | awk '{print $2}' | sort | uniq -c
      7 westeurope
      2 global
```

Siete recursos en una región no aprobada —creados durante una prueba y olvidados— y dos globales, que son zonas DNS y no deben bloquearse. La directiva integrada ya los excluye; una escrita a mano los habría bloqueado.

Se migran los siete, se comprueba que la conformidad llega al 100 % y solo entonces se pasa a bloqueo:

```
$ az policy assignment update --name regiones --enforcement-mode Default
$ az group create --name rg-prueba --location westeurope
(RequestDisallowedByPolicy) Resource 'rg-prueba' was disallowed by policy   ✓
$ az network dns zone create -g rg-cloudshop-red -n cloudshop.cl
{... "location": "global" ...}                                              ✓ no bloqueado
```

Resultado, con la comparación que interesa para el resto del programa:

```
frontera de aislamiento                cuenta             suscripción
agrupador de gobierno                  unidad organizativa grupo de administración
cuota compartida entre entornos        no                 no (tras el rediseño)
política heredada                      SCP (solo denegar)  Directiva (auditar,
                                                           denegar, modificar,
                                                           desplegar)
identidad                              por cuenta          por tenant  ← DIFIERE
nivel intermedio                       —                   grupo de recursos ← DIFIERE
```

Las dos últimas filas son las decisiones de proveedor; el resto reaparece con otro nombre. Esa distinción es la que pedía la pregunta de cierre de la clase 036, y es la que hará que la parte 13 pueda hablar de portabilidad con criterio.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-03-azure-core-platform/037-tenant-management-groups-suscripciones-y-resource-groups/lab.py
```

El laboratorio selecciona el motor de práctica governance y produce labresult.json. El escenario, sus comprobaciones y el artefacto esperado corresponden a esta clase; no requiere credenciales y deja explícito qué debe revalidarse en un sandbox real.

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa jerarquia-azure en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es una jerarquía de política, ownership y evidencia. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye jerarquia-azure para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
Un entorno de pruebas deja sin capacidad a producción pese a estar en otro grupo de recursos	Las cuotas son de la suscripción; el grupo de recursos no aísla	Separa por suscripción donde en AWS separarías por cuenta; el grupo de recursos es ciclo de vida.
Una directiva de regiones bloquea la creación de zonas DNS	Los recursos globales declaran ubicación global y una directiva casera no los excluye	Usa la directiva integrada de ubicaciones permitidas, que ya los contempla.
Se aplica una directiva Deny y se paran despliegues legítimos	No se midió la conformidad del estado existente antes de bloquear	Asigna en modo auditoría, revisa el informe una o dos semanas y solo entonces refuerza.
Hay núcleos libres en la suscripción y la creación falla igual	La cuota es por familia de máquina además de por total	Consulta el uso por familia; el total disponible no garantiza capacidad de la familia que necesitas.
Un despliegue falla al superar el número de reglas de un grupo de seguridad	Es un límite de servicio, no una cuota: no se amplía	Rediseña con grupos de seguridad de aplicación o listas de prefijos en vez de reglas individuales.

■ Seguridad, ética y costo

Trabaja con cuentas propias o sandboxes autorizados. No publiques secretos, identificadores reales ni datos personales. Antes de crear recursos pagos define presupuesto, etiquetas y comando de destrucción; después verifica que no queden recursos huérfanos. Los ejemplos locales enseñan contratos, pero no certifican cumplimiento ni disponibilidad de producción.

■ Preguntas de comprobación

1. ¿Cuál es la frontera de aislamiento en Azure y por qué el grupo de recursos no lo es?
2. Nombra las dos diferencias estructurales entre la jerarquía de AWS y la de Azure que sí cambian el diseño.
3. ¿Qué puede hacer una directiva de Azure que una SCP no puede, y qué cuidado adicional exige?
4. Tienes 60 núcleos libres en la suscripción y la creación de una máquina falla. ¿Qué compruebas?
5. ¿Por qué una directiva se asigna primero en modo auditoría, y qué información produce esa fase?

Referencias

- Microsoft (2024). Organize your Azure resources with management groups — jerarquía, herencia y profundidad máxima. <https://learn.microsoft.com/en-us/azure/governance/management-groups/overview>
- Microsoft (2024). Azure Policy: understand policy effects — audit, deny, modify y deployIfNotExists. <https://learn.microsoft.com/en-us/azure/governance/policy/concepts/effects>
- Microsoft (2024). Azure subscription and service limits, quotas, and constraints — cuotas ampliables y límites duros. <https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/azure-subscription-service-limits>
- Microsoft (2024). Cloud Adoption Framework: enterprise-scale landing zones — estructura de grupos de administración de referencia. <https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/landing-zone/>
- Microsoft (2024). Azure Resource Manager: resource groups — ámbito, ciclo de vida y qué no aísla. <https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-portal>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 03 en PDF · Recorrido de Azure en PDF · Manual integral

Anterior	Índice	Siguiente
← 036 · Proyecto: aplicación de tres capas en AWS	Parte 03 · Programa	038 · Microsoft Entra ID, RBAC, managed identities y PIM →

Evaluación — 037 Tenant, management groups, suscripciones y resource groups

Preguntas

1. Explica el problema que resuelve esta capacidad sin mencionar un proveedor.
2. Identifica una frontera de responsabilidad y su propietario.
3. Compara dos alternativas mediante confiabilidad, seguridad, costo y operación.
4. ¿Qué demuestra el laboratorio y qué no puede demostrar?
5. Propón un caso límite o una prueba de fallo.

Reto verificable

Entrega jerarquia-azure con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

- [ ] Resultado reproducible por una segunda persona.
- [ ] Evidencia vinculada a cada afirmación técnica.
- [ ] Prueba negativa o de fallo incluida.
- [ ] Costos y permisos expresados con unidades y alcance.
- [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.

038 — Microsoft Entra ID, RBAC, managed identities y PIM

Parte: 03 — Azure: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: iam · Estado: EXECUTABLECORE

Propósito

Dominar el modelo de identidad de Azure, que difiere del de AWS en algo estructural: el directorio es del tenant y los permisos sobre recursos son otra cosa. Confundir un rol de Entra ID con uno de Azure RBAC es la causa de la mitad de los problemas de acceso al llegar de AWS, y de una parte de las escaladas de privilegio.

Resultados de aprendizaje

Al finalizar podrás:

1. Distinguir roles de Entra ID de roles de Azure RBAC y saber cuál gobierna cada acción.
2. Explicar por qué una denegación de Azure RBAC no es equivalente a un Deny de IAM y qué la sustituye.
3. Sustituir secretos de aplicación por identidades administradas y por federación desde un pipeline.
4. Aplicar acceso con privilegios mínimos en el tiempo mediante activación con caducidad y aprobación.
5. Diagnosticar un acceso denegado sabiendo en qué ámbito y en qué sistema de roles buscar.

Conceptos centrales

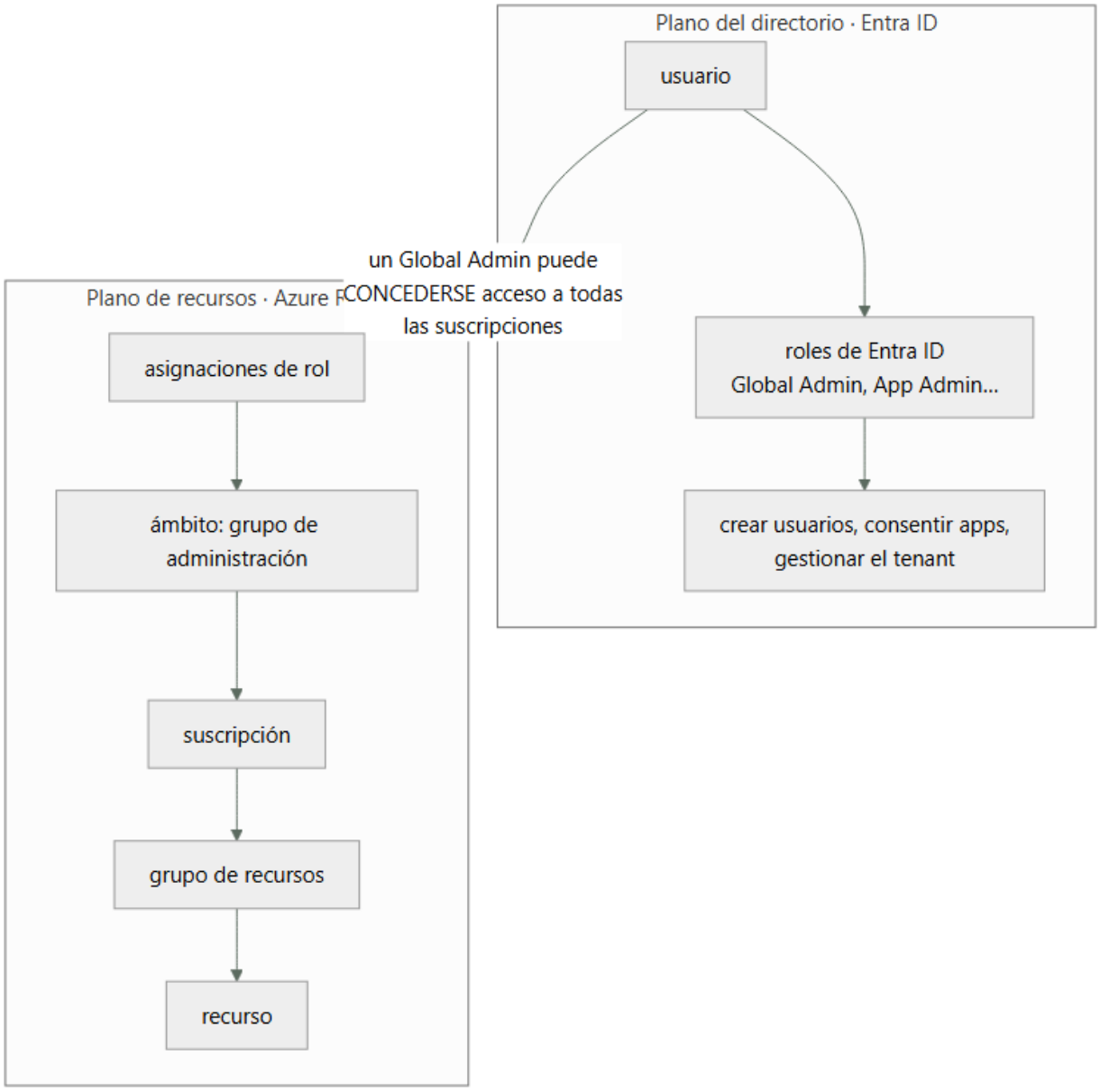
Concepto	Comprensión verificable
rol de Entra ID	Permiso sobre el directorio: crear usuarios, conceder consentimiento a aplicaciones, gestionar el propio tenant. No otorga ningún permiso sobre recursos de Azure.
rol de Azure RBAC	Permiso sobre recursos, asignado en un ámbito —grupo de administración, suscripción, grupo de recursos o recurso—. Se hereda hacia abajo y es acumulativo.
identidad administrada	Identidad que Azure crea y rota por ti, asociada a un recurso. Elimina el secreto: no hay nada que guardar ni que filtrar.
asignación de denegación	Mecanismo que bloquea acciones con independencia de los roles concedidos. A diferencia de IAM, no se puede crear directamente: solo la generan servicios como los blueprints o los entornos gestionados.
acceso con privilegios mínimos en el tiempo	Modelo en el que un rol privilegiado no está activo permanentemente: se activa por un periodo acotado, con justificación y posible aprobación.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Aplicado a esta clase, separa siempre cuatro planos: intención (qué necesita el usuario), configuración (qué declaramos), estado observado (qué existe de verdad) y evidencia (cómo sabemos que cumple). Confundirlos produce diseños que se ven correctos en un diagrama pero fallan al operar.

■ Flujo de razonamiento



Desarrollo

1. Dos sistemas de roles que no se solapan

Azure tiene dos planos de autorización independientes, y esa es la diferencia estructural con AWS:

	Roles de Entra ID	Roles de Azure RBAC
Gobiernan	El directorio	Los recursos
Ejemplos	Global Administrator, User Administrator	Owner, Contributor, Reader
Ámbito	Todo el tenant o unidades administrativas	Grupo de administración → recurso
Pregunta que responden	¿Puedes crear un usuario?	¿Puedes crear una máquina virtual?

Un Global Administrator no tiene por defecto ningún permiso sobre recursos. Puede crear usuarios y consentir aplicaciones, y no puede leer una base de datos.

Pero hay una conexión peligrosa que conviene conocer:

```
# Un Global Administrator puede concederse a sí mismo acceso a TODAS las suscripciones
$ az rest --method post --url "https://management.azure.com/providers/Microsoft.Authorization/
elevateAccess?api-version=2016-07-01"
```

Esa llamada le asigna User Access Administrator en el ámbito raíz. Es una función legítima —recuperar el control de una suscripción huérfana— y una escalada completa. Debe alertarse siempre, y el número de Global Administrators debe ser mínimo: entre dos y cuatro, todos con acceso privilegiado en el tiempo.

Al depurar «no tengo permiso», la primera pregunta es en qué plano está la acción. Crear un grupo de recursos es RBAC; invitar a un usuario externo es Entra ID. Añadir permisos en el plano equivocado no cambia nada, igual que añadir permisos de identidad contra un Deny de SCP en la clase 026.

2. RBAC es aditivo y las denegaciones no se crean a mano

El modelo de evaluación de Azure RBAC es más simple que el de AWS y su simplicidad tiene una consecuencia importante:

```
permiso efectivo = unión de todas las asignaciones en todos los ámbitos heredados
                   menos las asignaciones de denegación (que no puedes crear)
```

Es aditivo. No existe un Deny que puedas escribir para excluir una acción concreta de un rol amplio. Si alguien tiene Contributor en la suscripción, tiene todo lo que ese rol incluye sobre todo lo que hay debajo, y no hay forma de restarle una acción con otra asignación.

Las dos formas reales de acotar:

```
1. Rol personalizado con NotActions
   define exactamente lo que incluye y lo que excluye

2. Azure Policy con efecto Deny
   no es autorización, es gobierno: impide la operación aunque el rol la permita
```

La segunda es la que sustituye funcionalmente al Deny de IAM, y opera en otro plano: el rol dice que puedes y la directiva impide que ocurra. Al diagnosticar, el mensaje de error las distingue:

```
AuthorizationFailed             → falta el rol
RequestDisallowedByPolicy       → el rol lo permite y la directiva lo impide
```

Un rol personalizado con exclusiones:

```
{
  "Name": "Colaborador sin borrado",
  "Actions": ["Microsoft.Compute/*", "Microsoft.Network/*", "Microsoft.Storage/*"],
  "NotActions": [
    "Microsoft.Compute/virtualMachines/delete",
    "Microsoft.Storage/storageAccounts/delete"
  ],
  "AssignableScopes": ["/subscriptions/aaaa-bbbb"]
}
```

Y una precisión que evita sorpresas: NotActions no es una denegación. Si otra asignación concede esa misma acción, se permite. Solo excluye la acción de ese rol.

3. Identidades administradas: el secreto que no existe

Es el equivalente al rol de instancia de AWS, con dos variantes:

	Asignada por el sistema	Asignada por el usuario
Ciclo de vida	Ligado al recurso: se borra con él	Independiente
Compartible	No	Sí, entre varios recursos
Cuándo	Un recurso único	Flotas y despliegues azul-verde

La segunda es la que conviene en una flota: si la identidad estuviera ligada al recurso, cada instancia nueva tendría una identidad distinta y habría que asignarle roles al crearla. Con identidad asignada por el usuario, los roles se conceden una vez y todas las instancias los heredan.

El código no maneja ningún secreto:

```
from azure.identity import DefaultAzureCredential
from azure.keyvault.secrets import SecretClient

cred = DefaultAzureCredential()          # detecta la identidad administrada
cliente = SecretClient("https://kv-cloudshop.vault.azure.net", cred)
```

```
secreto = cliente.get_secret("db-password").value
```

DefaultAzureCredential prueba varias fuentes en orden: variables de entorno, identidad administrada, credenciales del desarrollador. Eso hace que el mismo código funcione en local y en producción, y también esconde una trampa: si la identidad administrada falla, puede caer silenciosamente a otra credencial y funcionar por el motivo equivocado. En producción conviene fijar la fuente explícitamente.

Para el pipeline, la equivalencia con la federación OIDC de la clase 026:

```
$ az ad app federated-credential create --id $APP_ID --parameters '{
    "name": "github-produccion",
    "issuer": "https://token.actions.githubusercontent.com",
    "subject": "repo:miorg/cloudshop:environment:produccion",
    "audiences": ["api://AzureADTokenExchange"]
  }'
```

El campo subject cumple exactamente la misma función crítica que en AWS: sin él acotado, cualquier repositorio puede obtener el token. Y aquí hay un detalle propio: el valor es de coincidencia exacta, no admite comodines, lo que elimina el error de usar StringLike con un patrón demasiado amplio.

4. Privilegio mínimo en el tiempo, no solo en el alcance

El privilegio mínimo clásico acota qué puedes hacer. Añadir la dimensión temporal acota cuándo:

```
modelo permanente   el rol está activo 24×7
                    una credencial comprometida a las 3 de la madrugada
                    tiene los mismos permisos que a mediodía

modelo elegible     el rol existe pero está inactivo
                    se activa por 1-8 h, con justificación y a veces aprobación
                    fuera de esa ventana, el permiso no existe
```

La reducción de superficie es aritmética:

```
rol permanente:  720 h/mes de exposición
rol elegible activado 4 h/semana: 16 h/mes
reducción: 97,8 %
```

La configuración añade tres controles a la activación:

```
duración máxima          8 h, no indefinida
justificación            obligatoria y registrada
aprobación               para los roles más sensibles
notificación             a un segundo par al activarse
```

La última es la más rentable y la menos usada: si alguien activa un rol privilegiado y otra persona lo ve al instante, un uso indebido se detecta en minutos en vez de en la revisión trimestral.

Los roles que deberían ser siempre elegibles y nunca permanentes:

```
Global Administrator          (Entra ID)
Privileged Role Administrator (Entra ID)
Owner y User Access Administrator en suscripciones de producción (RBAC)
```

Y una excepción deliberada: conviene mantener dos cuentas de emergencia con acceso permanente, excluidas de acceso condicional y con credenciales en custodia física. Son la salida si el sistema de activación o el proveedor de identidad falla. Su uso debe alertar de inmediato, y su existencia hay que documentarla — porque una cuenta de emergencia olvidada es una puerta trasera.

5. Diagnosticar un acceso denegado

El orden que acota el problema en minutos:

```
# 1. ¿Qué roles tengo y en qué ámbito?
$ az role assignment list --assignee $UPN --all -o table --include-inherited

# 2. ¿La acción concreta está en alguno de esos roles?
$ az role definition list --name "Storage Blob Data Reader" \
    --query "[0].permissions[0].{a:actions,da:dataActions}"

# 3. ¿Lo impide una directiva en vez de faltar el rol?
$ az policy state list --resource $ID --filter "complianceState eq 'NonCompliant'" \
    --query "[].policyDefinitionName"
```

El paso 2 esconde la sutileza que más confunde en Azure: existen actions y dataActions, y son planos distintos.

```
actions      operaciones sobre el recurso: crear la cuenta de almacenamiento,
             leer sus claves, cambiar su configuración
dataActions  operaciones sobre los DATOS: leer un blob concreto
```

Un rol Contributor sobre una cuenta de almacenamiento no permite leer los blobs — permite gestionarla y obtener sus claves, que es otra cosa. Leerlos exige un rol con dataActions, como Storage Blob Data Reader.

Eso produce el error más desconcertante para quien llega de AWS: alguien con Owner sobre la suscripción recibe AuthorizationPermissionMismatch al listar blobs. No es un fallo: es que Owner concede actions y no dataActions sobre datos.

Y el mensaje de error distingue los tres casos:

```
AuthorizationFailed                 falta el rol de gestión
AuthorizationPermissionMismatch     falta el rol de DATOS
RequestDisallowedByPolicy           hay rol y una directiva lo impide
```

Leerlo ahorra la búsqueda en el plano equivocado, igual que en la clase 026.

Ejemplo trabajado

Al desplegar CloudShop en Azure, el equipo replica el rol del pipeline de AWS y se encuentra con tres problemas en dos días.

Problema 1 — el pipeline no puede leer el almacenamiento pese a ser Owner.

```
$ az role assignment list --assignee $APP_ID --all --query "[].{rol:roleDefinitionName,ambito:scope}"
-o tsv
Owner    /subscriptions/aaaa-bbbb
$ az storage blob list --account-name stcloudshop --container-name facturas --auth-mode login
AuthorizationPermissionMismatch
```

Owner sobre la suscripción y no puede listar blobs. La causa es la separación entre actions y dataActions:

```
$ az role definition list --name Owner --query "[0].permissions[0].dataActions" -o tsv
(vacío)
```

Owner no tiene ninguna dataAction. Se asigna el rol de datos, acotado al contenedor:

```
$ az role assignment create --assignee $APP_ID \
    --role "Storage Blob Data Contributor" \
    --scope "/subscriptions/aaaa-bbbb/resourceGroups/rg-prod/providers/\
Microsoft.Storage/storageAccounts/stcloudshop/blobServices/default/containers/facturas"
$ az storage blob list --account-name stcloudshop --container-name facturas --auth-mode login -o tsv
| wc -l
412                                                                          ✓
```

Y se aprovecha para quitar Owner, que estaba de más:

```
antes:   Owner sobre la suscripción entera
después: Storage Blob Data Contributor sobre UN contenedor
         + Website Contributor sobre el grupo de recursos de la app
```

Problema 2 — el pipeline usaba un secreto de aplicación.

```
$ az ad app credential list --id $APP_ID --query "[].{fin:endDateTime}" -o tsv
2027-03-14T10:22:00Z
```

Un secreto con dos años de vigencia, guardado en el repositorio. Se sustituye por federación:

```
$ az ad app federated-credential create --id $APP_ID --parameters '{
    "name":"github-produccion",
    "issuer":"https://token.actions.githubusercontent.com",
    "subject":"repo:miorg/cloudshop:environment:produccion",
    "audiences":["api://AzureADTokenExchange"]}'
$ az ad app credential delete --id $APP_ID --key-id $KEY_ID
```

Pruebas del sujeto, igual que en la clase 026:

```
desde otro repositorio                  AADSTS70021: no matching federated identity  ✓
desde miorg/cloudshop, rama de trabajo  AADSTS70021                                  ✓
desde miorg/cloudshop, entorno produccion token emitido                              ✓
```

Problema 3 — un despliegue legítimo bloqueado.

```
RequestDisallowedByPolicy: Resource 'st-cloudshop-tmp' was disallowed by policy
  'Storage accounts should restrict network access'
```

El mensaje no dice AuthorizationFailed, así que no falta rol: hay rol y una directiva lo impide. Añadir permisos no habría servido de nada.

```
$ az policy assignment show --name red-almacenamiento --query "parameters" -o json
{"effect":{"value":"Deny"}}
```

La cuenta temporal se creaba sin restricción de red. Se corrige en la plantilla, no en la directiva.

Revisión final de la superficie de identidad:

```
$ az role assignment list --all --query "[?roleDefinitionName=='Owner']\
.{p:principalName,a:scope}" -o tsv | wc -l
11
```

Once asignaciones de Owner. Se reducen a dos y el resto pasa a elegible con activación:

```
Owner permanentes                   11            2 (emergencia, documentadas)
Owner elegibles                      0            9 (activación de 8 h con justificación)
Global Administrators                6            3, todos elegibles
secretos de aplicación               1            0
exposición de roles privilegiados  720 h/mes    ~22 h/mes   (−97 %)
```

La lección que traslada esta clase al resto del programa: el modelo de identidad de Azure se parece al de AWS en el objetivo y difiere en la mecánica. Traducir literalmente —«Owner es como AdministratorAccess»— produce a la vez permisos de más en el plano de gestión y permisos de menos en el plano de datos.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-03-azure-core-platform/038-microsoft-entra-id-rbac-managed-identities-y-pim/lab.py
```

El laboratorio selecciona el motor de práctica iam y produce labresult.json. El escenario, sus comprobaciones y el artefacto esperado corresponden a esta clase; no requiere credenciales y deja explícito qué debe revalidarse en un sandbox real.

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa identidad-azure en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es una matriz de acceso mínimo con prueba de denegación. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye identidad-azure para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
Un principal con Owner recibe AuthorizationPermissionMismatch al leer datos	Owner concede actions y no dataActions: son planos distintos	Asigna un rol de datos como Storage Blob Data Reader, acotado al contenedor.
Se añaden roles y el despliegue sigue fallando	El error es RequestDisallowedByPolicy: hay rol y una directiva lo impide	Lee el código de error; con directivas, la corrección va en la plantilla o en la directiva, no en los roles.
No se puede excluir una acción concreta de un rol amplio	Azure RBAC es aditivo y las asignaciones de denegación no se crean a mano	Usa un rol personalizado con NotActions o una directiva con efecto Deny.
Un Global Administrator obtiene acceso a todas las suscripciones	La elevación de acceso es una función legítima del rol	Minimiza los Global Administrators, hazlos elegibles y alerta siempre sobre elevateAccess.
Cada instancia nueva de una flota necesita asignaciones de rol propias	Se usó identidad asignada por el sistema, ligada al ciclo de vida del recurso	Usa identidad asignada por el usuario: los roles se conceden una vez y las instancias la comparten.

■ Seguridad, ética y costo

Trabaja con cuentas propias o sandboxes autorizados. No publiques secretos, identificadores reales ni datos personales. Antes de crear recursos pagos define presupuesto, etiquetas y comando de destrucción; después verifica que no queden recursos huérfanos. Los ejemplos locales enseñan contratos, pero no certifican cumplimiento ni disponibilidad de producción.

■ Preguntas de comprobación

1. ¿Qué puede hacer un Global Administrator sobre los recursos de una suscripción, y cómo puede cambiar eso?
2. ¿Por qué NotActions en un rol personalizado no equivale a un Deny de IAM?
3. Un principal con Owner no puede listar blobs. ¿Cuál es la causa y cuál el arreglo?
4. ¿Qué distingue AuthorizationFailed de RequestDisallowedByPolicy y qué corrige cada uno?
5. ¿Cuánto se reduce la exposición al pasar un rol de permanente a elegible activado 4 h por semana?

Referencias

- Microsoft (2024). Azure RBAC vs Microsoft Entra roles — los dos planos y qué gobierna cada uno. <https://learn.microsoft.com/en-us/entra/identity/role-based-access-control/custom-overview>
- Microsoft (2024). Understand Azure role definitions — actions, notActions, dataActions y su evaluación. <https://learn.microsoft.com/en-us/azure/role-based-access-control/role-definitions>
- Microsoft (2024). Managed identities for Azure resources — asignadas por el sistema y por el usuario. <https://learn.microsoft.com/en-us/entra/identity/managed-identities-azure-resources/overview>
- Microsoft (2024). Workload identity federation — credenciales federadas y coincidencia exacta del sujeto. <https://learn.microsoft.com/en-us/entra/workload-id/workload-identity-federation>
- Microsoft (2024). Privileged Identity Management — activación con caducidad, justificación y aprobación. <https://learn.microsoft.com/en-us/entra/id-governance/privileged-identity-management/pim-configure>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 03 en PDF · Recorrido de Azure en PDF · Manual integral

Anterior	Índice	Siguiente
← 037 · Tenant, management groups, suscripciones y resource groups	Parte 03 · Programa	039 · Virtual Network, subredes, NSG, UDR, peering y Private Link →

Evaluación — 038 Microsoft Entra ID, RBAC, managed identities y PIM

Preguntas

1. Explica el problema que resuelve esta capacidad sin mencionar un proveedor.
2. Identifica una frontera de responsabilidad y su propietario.
3. Compara dos alternativas mediante confiabilidad, seguridad, costo y operación.
4. ¿Qué demuestra el laboratorio y qué no puede demostrar?
5. Propón un caso límite o una prueba de fallo.

Reto verificable

Entrega identidad-azure con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

- [ ] Resultado reproducible por una segunda persona.
- [ ] Evidencia vinculada a cada afirmación técnica.
- [ ] Prueba negativa o de fallo incluida.
- [ ] Costos y permisos expresados con unidades y alcance.
- [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.

039 — Virtual Network, subredes, NSG, UDR, peering y Private Link

Parte: 03 — Azure: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: network · Estado: EXECUTABLECORE

Propósito

Diseñar la red de Azure sabiendo dónde falla la traducción literal desde una VPC. Tres puntos concretos rompen el modelo mental de la clase 027: no existe la subred privada por defecto, hay dos grupos de seguridad de red evaluándose en cadena y ambos deben permitir, y un punto de conexión privado sin zona DNS no falla — funciona por el camino público mientras crees que es privado. Es la base de las clases 040 a 048 y el origen de la mayoría de incidentes de conectividad de la parte.

Resultados de aprendizaje

Al finalizar podrás:

1. Planificar el espacio de direcciones contando las cinco IP reservadas por subred y las subredes de nombre obligatorio.
2. Explicar por qué una subred de Azure alcanza internet sin ninguna ruta declarada y qué cambió el 30 de septiembre de 2025.
3. Aplicar reglas de NSG sabiendo que subred y NIC se evalúan en cadena y que las reglas por defecto permiten todo dentro de la red virtual.
4. Decidir entre endpoint de servicio y punto de conexión privado con criterio de alcance, origen y costo.
5. Diagnosticar conectividad de abajo hacia arriba distinguiendo ruta, NSG y resolución de nombres.

Conceptos centrales

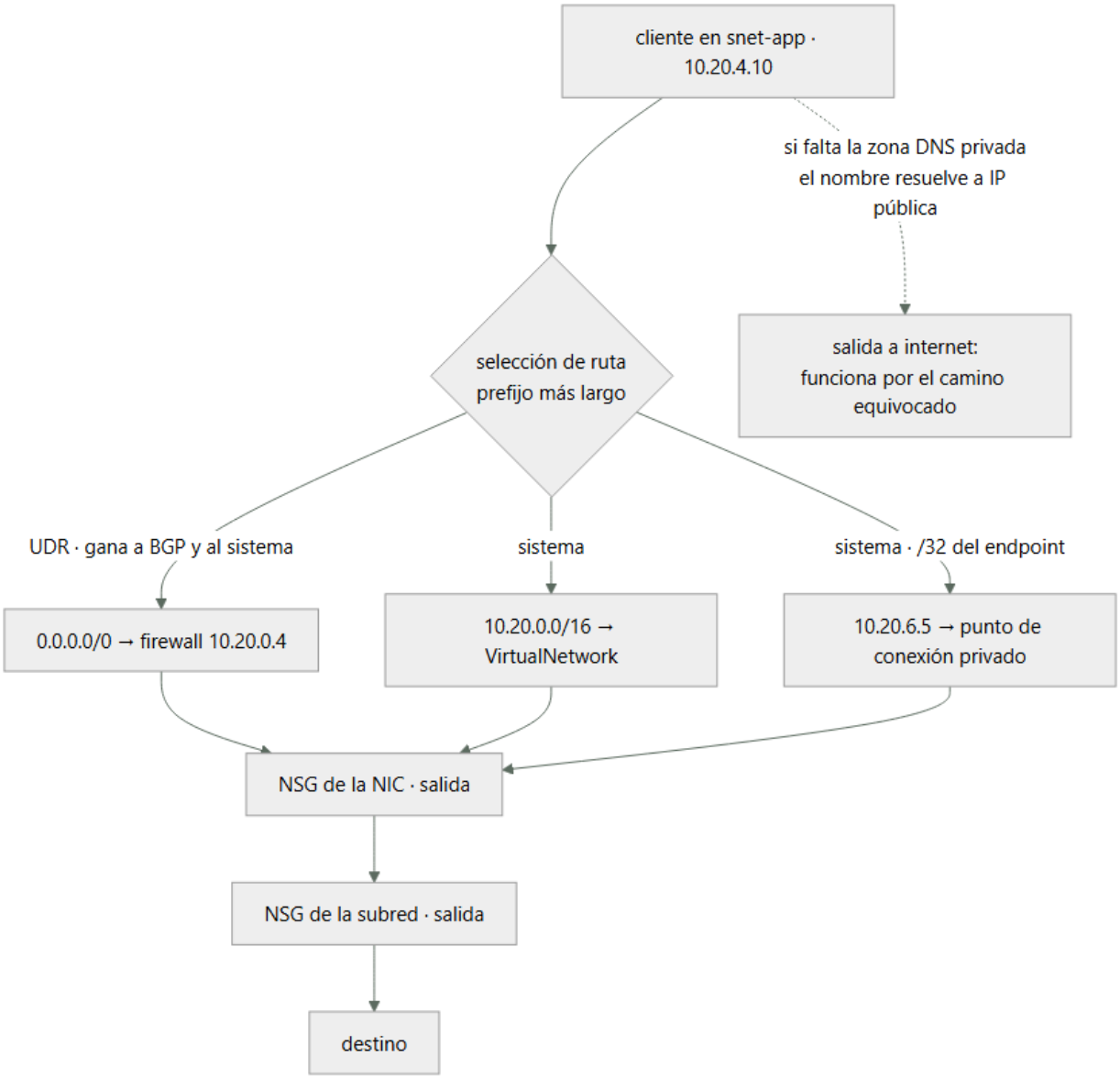
Concepto	Comprensión verificable
red virtual y espacio de direcciones	Ámbito de direccionamiento privado dentro de una región. Azure reserva cinco direcciones por subred —las cuatro primeras y la última—, así que una /24 ofrece 251 utilizables, no 254.
subred de nombre reservado	Subredes cuyo nombre es parte de la API: GatewaySubnet, AzureFirewallSubnet, AzureBastionSubnet. Escrito de otra forma, el servicio simplemente no se puede crear ahí.
ruta del sistema	Ruta que Azure crea sin que la declares, incluida 0.0.0.0/0 → Internet en toda subred. Es la razón de que no exista una subred privada por omisión.
UDR	Tabla de rutas definida por el usuario. Se elige por prefijo más largo y, a igualdad de prefijo, UDR gana a BGP y BGP gana al sistema.
NSG	Filtro con estado que puede aplicarse a la subred y a la NIC a la vez. Ambos se evalúan y ambos deben permitir; la primera regla que coincide decide.
punto de conexión privado	Interfaz de red con IP privada de tu subred que representa a un recurso concreto. Sin la zona DNS privada correspondiente no da error: el nombre sigue resolviendo a la IP pública.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Aplicado a esta clase, separa siempre cuatro planos: intención (qué necesita el usuario), configuración (qué declaramos), estado observado (qué existe de verdad) y evidencia (cómo sabemos que cumple). Confundirlos produce diseños que se ven correctos en un diagrama pero fallan al operar.

■ Flujo de razonamiento



Desarrollo

1. Cinco direcciones y tres nombres que no puedes elegir

El direccionamiento se planifica antes de crear nada, por la misma razón que en la clase 027: el emparejamiento entre redes virtuales rechaza espacios solapados, y dos equipos que eligieron 10.0.0.0/16 cada uno por su cuenta ya no pueden conectarse sin rehacer una de las dos.

Azure reserva cinco direcciones por subred, igual que AWS, pero con otros papeles:

```
10.20.1.0    identificador de red
10.20.1.1    puerta de enlace predeterminada
10.20.1.2    servicio DNS de Azure
10.20.1.3    reservada
10.20.1.255  difusión
→ una /24 ofrece 251 direcciones utilizables
```

Hasta aquí, aritmética conocida. Lo que no existe en AWS son las subredes cuyo nombre forma parte del contrato:

Nombre exacto	Tamaño mínimo	Para qué
GatewaySubnet	/27 recomendado	Puertas de enlace de VPN y ExpressRoute

Nombre exacto	Tamaño mínimo	Para qué
AzureFirewallSubnet	/26	Azure Firewall
AzureFirewallManagementSubnet	/26	Plano de gestión del firewall forzado
AzureBastionSubnet	/26	Bastion

El nombre no es una convención, es la API. Una subred llamada snet-gateway con el tamaño correcto no sirve: la puerta de enlace no se puede crear ahí. Y el orden importa, porque estas subredes deben caber en el espacio que ya repartiste: descubrir que hace falta una /26 para el firewall cuando solo quedan /28 libres obliga a ampliar el espacio de direcciones de la red virtual y a recrear emparejamientos.

Un reparto que deja sitio a lo que vendrá en las clases 040 a 048:

```
vnet-hub    10.20.0.0/22
  AzureFirewallSubnet   10.20.0.0/26
  AzureBastionSubnet    10.20.0.64/26
  GatewaySubnet         10.20.0.128/27
  snet-dns              10.20.0.160/28

vnet-spoke-app  10.20.4.0/22
  snet-app              10.20.4.0/24    (251 utilizables)
  snet-datos            10.20.5.0/24
  snet-endpoints        10.20.6.0/26    (puntos de conexión privados)
  snet-integracion      10.20.6.64/26   (delegada a App Service)
```

La última merece una nota: una subred delegada a un servicio —App Service, Container Apps, bases de datos flexibles— queda bajo control de ese servicio y no admite otros recursos. Se decide al crearla y cambiarla implica vaciarla, así que conviene reservarla desde el principio aunque todavía no se use.

2. No existe la subred privada: toda subred sale a internet

Esta es la diferencia estructural con AWS, y la que más incidentes produce al llegar.

En AWS, «pública» y «privada» son propiedades de la tabla de rutas: una subred es pública porque su tabla apunta a una puerta de enlace de internet, y es privada porque no lo hace. En Azure no hay puerta de enlace que adjuntar. Toda subred nace con esta ruta del sistema:

```
0.0.0.0/0  →  Internet
```

Una máquina virtual sin IP pública, sin UDR y sin NAT alcanza internet igualmente, mediante una traducción de direcciones implícita con una IP que elige la plataforma. Se llama acceso saliente predeterminado y tiene tres consecuencias:

```
1. La salida no está controlada: cualquier proceso sale sin que nadie lo declare.
2. La IP de origen no es estable ni tuya: no puede figurar en la lista de
   permitidos de un tercero, porque cambia sin aviso.
3. Los puertos de traducción son escasos y no se pueden dimensionar.
```

Qué cambió. Desde el 30 de septiembre de 2025 el acceso saliente predeterminado está retirado para las redes virtuales nuevas: un despliegue nuevo debe declarar cómo sale. Las redes virtuales creadas antes lo conservan, así que la pregunta útil sobre una plataforma existente no es «¿tenemos salida implícita?» sino «¿de qué fecha es esta red virtual?».

Las tres formas explícitas de salir, y por qué no son intercambiables:

Mecanismo	Puertos de traducción	Cuándo
IP pública en la NIC	64.000 por instancia	Una máquina suelta; no escala ni se audita bien
Regla de salida del balanceador	64.000 repartidos entre las instancias	Flotas pequeñas y estables
NAT Gateway	64.512 por IP, asignados bajo demanda, hasta 16 IP	Recomendado por defecto

El reparto de la segunda opción es la trampa aritmética:

```
64.000 puertos / 100 instancias = 640 puertos por instancia
un servicio que abre 200 conexiones cortas por segundo al mismo destino,
con 240 s de espera antes de reutilizar el puerto → agotamiento en segundos
```

El síntoma es característico y desorienta: conexiones que fallan por tiempo de espera solo hacia un destino concreto, mientras el resto de la red funciona. El NAT Gateway lo elimina porque asigna puertos bajo demanda en vez de repartirlos por adelantado.

Su costo tiene la misma forma que el de AWS, así que la lección de la clase 027 se traslada entera:

```
precio por hora         ~0,045 USD  →  32,85 USD/mes
precio por GB procesado ~0,045 USD
```

Y para conseguir de verdad una subred sin salida hacen falta tres cosas, no una:

```
# 1. desactivar el acceso saliente predeterminado en la subred
$ az network vnet subnet update -g rg-red --vnet-name vnet-spoke-app -n snet-datos \
    --default-outbound-access false

# 2. ninguna UDR que envíe 0.0.0.0/0 a Internet
# 3. NSG que deniegue la salida a la etiqueta Internet
$ az network nsg rule create -g rg-red --nsg-name nsg-datos -n deny-internet-out \
    --priority 4000 --direction Outbound --access Deny \
    --source-address-prefixes '*' --destination-address-prefixes Internet \
    --destination-port-ranges '*' --protocol '*'
```

Saltarse la primera es el error habitual: la regla de NSG bloquea el tráfico, pero la ruta sigue existiendo y basta con que alguien afloje el NSG para que la salida vuelva sin que nadie lo note.

3. Dos NSG en cadena y una regla por defecto que lo permite todo dentro

El NSG tiene estado, en la subred y en la NIC. Esto elimina de golpe la trampa de las ACL de red de AWS —aquellas reglas de vuelta para los puertos efímeros de la clase 027—, y a cambio introduce otra.

```
entrada:  NSG de la subred  →  NSG de la NIC   →  máquina
salida:   NSG de la NIC     →  NSG de la subred →  red
```

Ambos se evalúan y ambos deben permitir. Un NSG de subred impecable no sirve de nada si la NIC tiene otro que deniega, y el mensaje de error no dice cuál de los dos fue. Por eso conviene una regla de equipo simple: NSG en la subred, salvo excepción documentada. Dos capas de filtrado en manos de dos equipos distintos producen incidentes que nadie sabe reproducir.

Las reglas por defecto, que no se pueden borrar y sí sobrescribir con prioridad menor:

```
entrada  65000  AllowVnetInBound              VirtualNetwork → VirtualNetwork
         65001  AllowAzureLoadBalancerInBound AzureLoadBalancer → *
         65500  DenyAllInBound

salida   65000  AllowVnetOutBound             VirtualNetwork → VirtualNetwork
         65001  AllowInternetOutBound         * → Internet
         65500  DenyAllOutBound
```

La primera es la que cambia el modelo mental. VirtualNetwork no significa «esta subred»: incluye toda la red virtual, las redes emparejadas y los rangos anunciados desde la red corporativa. Es decir:

```
AWS    dos instancias con grupos de seguridad distintos NO se hablan
       hasta que una regla lo permita
Azure  dos subredes cualesquiera de la red virtual SÍ se hablan,
       en todos los puertos, desde el primer minuto
```

La segmentación entre subredes en Azure es trabajo explícito, no un estado inicial. Denegar el tráfico lateral y abrir solo lo necesario:

```
$ az network nsg rule create -g rg-red --nsg-name nsg-datos -n allow-app-5432 \
    --priority 100 --direction Inbound --access Allow --protocol Tcp \
    --source-address-prefixes 10.20.4.0/24 --destination-port-ranges 5432

$ az network nsg rule create -g rg-red --nsg-name nsg-datos -n deny-lateral \
    --priority 200 --direction Inbound --access Deny --protocol '*' \
    --source-address-prefixes VirtualNetwork --destination-port-ranges '*'
```

Se evalúa por prioridad ascendente y la primera coincidencia decide: la 100 permite la aplicación y la 200 corta todo lo demás que venga de dentro, incluida la sonda del balanceador si no se tuvo cuidado. Ese es el incidente clásico, porque AllowAzureLoadBalancerInBound vive en la 65001 y cualquier denegación de usuario la sobrescribe:

```
sonda bloqueada → todas las instancias marcadas como no sanas
                → el balanceador deja de enviar tráfico
                → 502 en el frontal, y las máquinas están perfectamente vivas
```

La corrección es una regla de permiso con la etiqueta de servicio AzureLoadBalancer por encima de la denegación. Las etiquetas de servicio son la mejora real frente a mantener listas de IP a mano:

```
VirtualNetwork      la red virtual, las emparejadas y lo anunciado desde on-premises
AzureLoadBalancer   las sondas de estado de la plataforma
Internet            todo lo que no es privado ni de Azure
Storage.WestEurope  los rangos de almacenamiento de una región concreta
AzureMonitor, Sql, AzureKeyVault…
```

Microsoft actualiza los rangos; tus reglas no cambian. Y para el equivalente al grupo de seguridad que referencia a otro grupo de seguridad —que en Azure no existe tal cual— está el grupo de seguridad de aplicación: una etiqueta que se asigna a NIC y se usa como origen o destino en las reglas, en lugar de un CIDR. Permite escribir «los servidores web pueden hablar con los de base de datos» sin depender de cómo se repartieron las subredes.

4. UDR, emparejamiento y las dos formas de dejar la red sin salida

La selección de ruta sigue dos criterios, en este orden:

```
1. prefijo más largo (la ruta más específica gana)
2. a igualdad de prefijo:  UDR  >  BGP  >  sistema
```

Forzar todo el tráfico saliente por un firewall es una UDR de una línea sobre la subred de carga de trabajo:

```
$ az network route-table route create -g rg-red --route-table-name rt-app \
    -n via-firewall --address-prefix 0.0.0.0/0 \
    --next-hop-type VirtualAppliance --next-hop-ip-address 10.20.0.4
```

Y tiene dos formas conocidas de provocar una caída total:

La primera: aplicar esa misma UDR a la subred del firewall. El firewall enruta hacia sí mismo, el tráfico entra en bucle y la salida desaparece para toda la red. La regla es que AzureFirewallSubnet y GatewaySubnet no llevan una ruta por defecto hacia el propio dispositivo; asociar la tabla «a todas las subredes por comodidad» es exactamente cómo ocurre.

La segunda: el enrutamiento asimétrico. El tráfico entra por la IP pública del balanceador y, con la UDR puesta, la respuesta intenta salir por el firewall. El firewall ve un paquete de vuelta de una conexión que nunca vio empezar y lo descarta. El síntoma engaña: la conexión se establece y se queda colgada hasta agotar el tiempo, así que parece un problema de la aplicación. La corrección es una ruta más específica que devuelva ese tráfico por donde entró, o mover la entrada detrás del mismo dispositivo que inspecciona la salida.

Sobre el emparejamiento, tres propiedades que hay que tener presentes a la vez:

```
no es transitivo        radio A ↔ concentrador ↔ radio B  NO da  A ↔ B
es bidireccional        hay que crearlo en los dos sentidos o queda a medias
se factura en los dos   se paga el GB que sale y el GB que entra
```

La no transitividad se resuelve como en AWS, con distinta mecánica: una UDR en cada radio que envíe el rango del otro al firewall del concentrador, más dos interruptores del emparejamiento que casi siempre se olvidan:

```
allowForwardedTraffic   sin él, la red emparejada descarta el tráfico cuyo
                        origen no le pertenece — es decir, todo lo reenviado
allowGatewayTransit /   permiten que los radios usen la puerta de enlace
useRemoteGateways       del concentrador en vez de tener una cada uno
```

El coste conviene calcularlo antes de dibujar el diagrama, porque un concentrador cobra el tránsito dos veces. Para 900 GB mensuales entre dos radios de la misma región:

```
emparejamiento radio A ↔ concentrador
  900 GB salida × 0,01 + 900 GB entrada × 0,01   18,00
emparejamiento concentrador ↔ radio B
  900 GB salida × 0,01 + 900 GB entrada × 0,01   18,00
proceso de datos del firewall  900 × 0,016       14,40
                                                 ■■■■■■■
                                                  50,40

emparejamiento directo A ↔ B                      18,00
```

Inspeccionar ese tráfico cuesta 32,40 USD al mes. No es una cifra que descalifique el diseño; es una cifra que hay que poder decir en voz alta. La decisión defendible es la que elige el concentrador porque el tráfico entre radios debe inspeccionarse, no la que lo elige porque el diagrama de referencia lo dibujaba así.

5. Endpoint de servicio y punto de conexión privado: el que falla es el DNS

Dos mecanismos para llegar a un servicio de plataforma sin pasar por internet, con nombres parecidos y comportamientos distintos:

	Endpoint de servicio	Punto de conexión privado
Qué hace	Añade una ruta optimizada y presenta la identidad de la subred al servicio	Crea una NIC con IP privada de tu subred que representa al recurso
Dirección de destino	La IP pública del servicio	Una IP privada tuya
DNS	Sin cambios	Hay que cambiarlo
Alcance	El servicio en la región, acotado por el cortafuegos del recurso	Un recurso concreto, e incluso un subrecurso (blob, file, table)
Desde on-premises o red emparejada	No funciona	Sí
Costo	Gratuito	0,01 USD/h + 0,01 USD/GB procesado

La equivalencia con la clase 027 es casi exacta: el endpoint de servicio se comporta como el endpoint de tipo gateway de AWS —gratuito, basado en ruta, regional, inútil desde la red corporativa— y el punto de conexión privado como el endpoint de interfaz: una interfaz de red con IP privada, con costo por hora y por GB, alcanzable desde fuera de la red virtual.

Y aquí está el fallo más peligroso de esta clase, porque no se manifiesta como un error.

Crear el punto de conexión privado no cambia nada para el cliente. El nombre público sigue existiendo y sigue resolviendo hacia fuera:

```
stcloudshop.blob.core.windows.net
  → CNAME stcloudshop.privatelink.blob.core.windows.net
    → 20.60.x.x        (IP pública, si no hay zona privada)
    → 10.20.6.5        (IP del endpoint, si la zona privada está enlazada)
```

Sin la zona DNS privada, la aplicación sigue funcionando: sale a internet por el NAT, llega al almacenamiento por su IP pública y devuelve datos correctos. El punto de conexión privado está creado, facturado y sin usar. No hay alerta, no hay excepción, no hay traza roja en ningún panel. La única señal es una consulta:

```
$ nslookup stcloudshop.blob.core.windows.net      # desde dentro de snet-app
Address: 10.20.6.5                                 ✓
```

Los tres pasos completos, en orden:

```
# 1. la zona privada, con el nombre EXACTO que corresponde al servicio
$ az network private-dns zone create -g rg-red -n privatelink.blob.core.windows.net

# 2. enlazarla a cada red virtual que deba resolverlo
$ az network private-dns link vnet create -g rg-red -n link-spoke-app \
    -z privatelink.blob.core.windows.net -v vnet-spoke-app -e false

# 3. cerrar la puerta pública del recurso
$ az storage account update -n stcloudshop --public-network-access Disabled
```

El tercero es el que convierte esto en un control. Con la puerta pública abierta, el camino privado es solo un camino más: bonito en el diagrama e irrelevante para el auditor, porque cualquiera con la clave sigue entrando desde internet.

Dos detalles que ahorran una tarde de depuración:

Las zonas se centralizan. Una zona por nombre de servicio, alojada en la suscripción de conectividad y enlazada a todas las redes virtuales. Si cada equipo crea la suya, dos endpoints del mismo servicio en redes distintas producen resoluciones incoherentes según desde dónde preguntes. Lo sostenible es una directiva de Azure que registre automáticamente cada punto de conexión nuevo en la zona correcta — el mismo patrón de gobierno de la clase 037: la regla se aplica sola en vez de recordarse.

El NSG sobre el endpoint puede estar sin efecto. La subred tiene un interruptor, privateEndpointNetworkPolicies, que decide si los NSG y las UDR se aplican al punto de conexión. Si está deshabilitado, tus reglas se ignoran en silencio y el panel las muestra igualmente. Comprobarlo forma parte de la evidencia:

```
$ az network vnet subnet show -g rg-red --vnet-name vnet-spoke-app -n snet-endpoints \
    --query privateEndpointNetworkPolicies -o tsv
Enabled                                              ✓
```

Ejemplo trabajado

CloudShop traslada a Azure la VPC diseñada en la clase 027. El plano se copia en una tarde y produce cuatro incidentes en tres semanas — cada uno en uno de los puntos donde la traducción no era válida.

El punto de partida, traducido literalmente:

```
vnet-spoke-app 10.20.4.0/22
  snet-app     10.20.4.0/24   sin IP pública  → «privada»
  snet-datos   10.20.5.0/24   sin IP pública  → «privada»
```

Incidente 1 — las subredes privadas no eran privadas.

Una revisión de dependencias detecta que el servicio de catálogo descarga paquetes durante el arranque. No debería tener salida.

```
$ ssh app-01 'curl -s ifconfig.io'
20.103.44.187
```

Hay salida, y con una IP que nadie declaró. Es el acceso saliente predeterminado: la red virtual se creó en 2024 y lo conserva. Se corrige con NAT Gateway explícito y se cierra la subred de datos:

```
$ az network nat gateway create -g rg-red -n natgw-spoke-app \
    --public-ip-addresses pip-nat --idle-timeout 10
$ az network vnet subnet update -g rg-red --vnet-name vnet-spoke-app -n snet-app \
    --nat-gateway natgw-spoke-app
$ az network vnet subnet update -g rg-red --vnet-name vnet-spoke-app -n snet-datos \
    --default-outbound-access false
$ ssh datos-01 'curl -s --max-time 5 ifconfig.io; echo rc=$?'
rc=28                                                                    ✓
```

El tráfico de salida medido es de 3,2 TB/mes, de los cuales 2,4 TB son lecturas y escrituras contra el almacenamiento:

```
NAT Gateway  32,85 + 3.200 × 0,045              176,85
```

Incidente 2 — 502 en el frontal con todas las máquinas vivas.

Al aplicar segmentación se añadió una denegación del tráfico lateral:

```
200  deny-lateral   Inbound  VirtualNetwork → *  Deny
```

A los diez minutos, el balanceador devuelve 502 y las cuatro instancias figuran como no sanas. Ninguna se ha caído.

```
$ az network watcher test-ip-flow --vm app-01 --direction Inbound --protocol TCP \
    --local 10.20.4.10:8080 --remote 168.63.129.16:60000
Access: Deny   RuleName: deny-lateral
```

La sonda de estado llega desde AzureLoadBalancer, permitida por defecto en la prioridad 65001 — y la regla 200 la sobrescribe. Se antepone el permiso explícito:

```
$ az network nsg rule create -g rg-red --nsg-name nsg-app -n allow-lb-probe \
    --priority 100 --direction Inbound --access Allow --protocol '*' \
    --source-address-prefixes AzureLoadBalancer --destination-port-ranges '*'
$ az network lb show -g rg-red -n lb-app --query "probes[0].name" -o tsv && \
  curl -s -o /dev/null -w '%{http_code}\n' https://cloudshop.example
200                                                                      ✓
```

Incidente 3 — dos radios que no se ven, y la factura de arreglarlo.

El servicio de pedidos, en vnet-spoke-app, debe llamar al de facturación, en vnet-spoke-fin. Ambos están emparejados con el concentrador y no se alcanzan entre sí: el emparejamiento no es transitivo. Se añaden las UDR hacia el firewall y sigue sin funcionar.

```
$ az network vnet peering show -g rg-red --vnet-name vnet-spoke-app -n to-hub \
    --query "{fwd:allowForwardedTraffic}" -o tsv
False
```

El concentrador descartaba el tráfico reenviado. Con el interruptor activado en los cuatro emparejamientos —dos por sentido— la ruta se completa. El volumen entre radios es de 900 GB/mes:

```
vía concentrador con inspección                        50,40
emparejamiento directo entre radios, sin inspección    18,00
```

```
diferencia: el precio de inspeccionar                    32,40
```

Se deja el paso por el concentrador y se anota el motivo: son llamadas que cruzan una frontera de datos de pago y deben registrarse. La cifra queda escrita en la decisión, no descubierta en la factura.

Incidente 4 — el punto de conexión privado que llevaba cuatro meses sin usarse.

Una auditoría pide evidencia de que el acceso al almacenamiento no cruza internet. El endpoint existe desde el primer despliegue y la aplicación funciona sin fallos.

```
$ ssh app-01 'nslookup stcloudshop.blob.core.windows.net | tail -2'
Address: 20.60.191.132
```

Una IP pública. Nunca se creó la zona DNS privada, así que el tráfico salía por el NAT y volvía por la puerta pública del almacenamiento. Todo funcionaba, y ninguna de las dos afirmaciones del diagrama era cierta.

```
$ az network private-dns zone create -g rg-hub -n privatelink.blob.core.windows.net
$ az network private-dns link vnet create -g rg-hub -n link-spoke-app \
    -z privatelink.blob.core.windows.net -v vnet-spoke-app -e false
$ az storage account update -n stcloudshop --public-network-access Disabled

$ ssh app-01 'nslookup stcloudshop.blob.core.windows.net | tail -2'
Address: 10.20.6.5                                                    ✓
$ curl -s -o /dev/null -w '%{http_code}\n' \
    https://stcloudshop.blob.core.windows.net/facturas   # desde fuera de la red
403                                                                   ✓
```

La prueba negativa es la mitad que faltaba: que resuelva a una IP privada demuestra que el camino existe; que desde fuera devuelva 403 demuestra que es el único.

Y se recuperan 2,4 TB del NAT:

```
tráfico por NAT Gateway              3.200 GB      800 GB
costo del NAT   32,85 + GB × 0,045    176,85       68,85
punto de conexión privado (7,30 + 2.400 × 0,01)     0,00 → ya facturado    31,30
                                    ■■■■■■■■■■    ■■■■■■■■■■
                                     176,85       100,15   (−43 %)
```

El punto de conexión ya se estaba pagando desde el primer día. Enlazar la zona DNS no añadió costo: hizo que el gasto sirviera para algo.

Resumen del rediseño:

```
subredes con salida no declarada             2               0
camino de datos al almacenamiento          internet          privado
acceso público al almacenamiento           habilitado        deshabilitado
tráfico lateral permitido por defecto      todos los puertos  5432 desde snet-app
salida mensual por NAT                     3,2 TB            0,8 TB
costo mensual de red                       227,25 USD        150,55 USD
```

La lección que esta clase traslada al resto de la parte: en Azure, la conectividad es el estado por defecto y el aislamiento es trabajo explícito. Al revisar un diseño heredado, la pregunta útil no es «¿qué hemos abierto?» sino «¿qué hemos cerrado, y cómo lo demostramos?».

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-03-azure-core-platform/039-virtual-network-subredes-nsg-udr-peering-y-private-link/lab.py
```

El laboratorio selecciona el motor de práctica network y produce labresult.json. El escenario, sus comprobaciones y el artefacto esperado corresponden a esta clase; no requiere credenciales y deja explícito qué debe revalidarse en un sandbox real.

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa vnet-azure en evidence/ usando la plantilla indicada.

5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es una topología con rutas, puertos y controles. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye vnet-azure para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
Una máquina sin IP pública en una subred «privada» descarga desde internet	La ruta del sistema 0.0.0.0/0 → Internet existe en toda subred y el acceso saliente predeterminado la resuelve	Declara la salida con NAT Gateway y desactiva defaultOutboundAccess en las subredes que no deben salir; el NSG por sí solo no basta.
El balanceador devuelve 502 y todas las instancias figuran como no sanas, pero están vivas	Una regla de denegación de usuario sobrescribe AllowAzureLoadBalancerInBound, que vive en la prioridad 65001	Añade un permiso explícito para la etiqueta AzureLoadBalancer con prioridad menor que la denegación.
Dos subredes se hablan en todos los puertos sin que nadie lo haya permitido	AllowVnetInBound permite por defecto todo el tráfico de la red virtual, las emparejadas y lo anunciado desde on-premises	La segmentación es explícita: permite lo necesario y deniega el resto del origen VirtualNetwork.
El punto de conexión privado existe, la aplicación funciona y el tráfico sigue saliendo a internet	Falta la zona DNS privada enlazada, así que el nombre resuelve a la IP pública del servicio	Crea y enlaza privatelink.<servicio>, verifica con nslookup desde dentro y deshabilita el acceso público del recurso.
La conexión se establece y se queda colgada hasta agotar el tiempo de espera	Enrutamiento asimétrico: la entrada llega por el balanceador y la UDR devuelve la respuesta por el firewall, que nunca vio empezar la conexión	Añade una ruta más específica que devuelva ese tráfico por donde entró, o unifica entrada y salida en el mismo dispositivo.
Se pierde la salida de toda la red al aplicar una tabla de rutas	La UDR con 0.0.0.0/0 → dispositivo virtual se asoció también a AzureFirewallSubnet o a GatewaySubnet	Asocia las tablas de rutas subred por subred; esas dos nunca llevan la ruta por defecto hacia el propio dispositivo.

■ Seguridad, ética y costo

Trabaja con cuentas propias o sandboxes autorizados. No publiques secretos, identificadores reales ni datos personales. Antes de crear recursos pagos define presupuesto, etiquetas y comando de destrucción; después verifica que no queden recursos huérfanos. Los ejemplos locales enseñan contratos, pero no certifican cumplimiento ni disponibilidad de producción.

■ Preguntas de comprobación

1. ¿Por qué una máquina virtual sin IP pública y sin UDR alcanza internet, y qué tres cambios hacen falta para impedirlo?
2. Un NSG de subred permite el tráfico y la conexión falla. ¿Dónde más hay que mirar y por qué el error no lo indica?
3. ¿Qué permite AllowVnetInBound exactamente, y en qué se diferencia del comportamiento por defecto entre grupos de seguridad de AWS?
4. ¿Cuándo elegirías un endpoint de servicio en vez de un punto de conexión privado, y qué requisito descarta al primero?
5. El nslookup de una cuenta de almacenamiento devuelve una IP pública desde dentro de la red virtual y la aplicación funciona. ¿Qué está ocurriendo y qué evidencia demuestra la corrección?

Referencias

• Microsoft (2025). Default outbound access in Azure — la salida implícita y su retirada para redes virtuales nuevas. <https://learn.microsoft.com/en-us/azure/virtual-network/ip-services/default-outbound-access>

• Microsoft (2025). Network security groups — reglas por defecto, evaluación en subred y NIC, y etiquetas de servicio. <https://learn.microsoft.com/en-us/azure/virtual-network/network-security-groups-overview>

• Microsoft (2025). Virtual network traffic routing — rutas del sistema y precedencia entre UDR, BGP y sistema. <https://learn.microsoft.com/en-us/azure/virtual-network/virtual-networks-udr-overview>

• Microsoft (2025). Private endpoint DNS configuration — zonas privatelink, enlaces a redes virtuales y resolución desde on-premises. <https://learn.microsoft.com/en-us/azure/private-link/private-endpoint-dns>

• Microsoft (2025). Hub-spoke network topology — transitividad, tráfico reenviado y tránsito de puerta de enlace. <https://learn.microsoft.com/en-us/azure/architecture/networking/architecture/hub-spoke>

• Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 03 en PDF · Recorrido de Azure en PDF · Manual integral

Anterior	Índice	Siguiente
← 038 · Microsoft Entra ID, RBAC, managed identities y PIM	Parte 03 · Programa	040 · Virtual Machines, Scale Sets y Load Balancer →

Evaluación — 039 Virtual Network, subredes, NSG, UDR, peering y Private Link

Preguntas

1. Explica el problema que resuelve esta capacidad sin mencionar un proveedor.
2. Identifica una frontera de responsabilidad y su propietario.
3. Compara dos alternativas mediante confiabilidad, seguridad, costo y operación.
4. ¿Qué demuestra el laboratorio y qué no puede demostrar?
5. Propón un caso límite o una prueba de fallo.

Reto verificable

Entrega vnet-azure con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

• [ ] Resultado reproducible por una segunda persona.

- [ ] Evidencia vinculada a cada afirmación técnica.
- [ ] Prueba negativa o de fallo incluida.
- [ ] Costos y permisos expresados con unidades y alcance.
- [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.

040 — Virtual Machines, Scale Sets y Load Balancer

Parte: 03 — Azure: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: compute · Estado: EXECUTABLECORE

Propósito

Dimensionar cómputo en Azure sabiendo que hay dos límites de disco y casi siempre manda el que no compraste: el de la máquina. Las clases 028 y 029 dejaron el criterio —familia, IOPS, elasticidad, comprobación de estado—; aquí cambian las piezas y aparecen decisiones que en AWS no existen: los dominios de actualización, las dos orquestaciones de un conjunto de escalado, cuatro balanceadores con un reparto distinto y un aviso de desalojo de 30 segundos en vez de dos minutos.

Resultados de aprendizaje

Al finalizar podrás:

1. Leer el nombre de un tamaño de máquina virtual como una especificación y detectar qué falta cuando no lleva s ni d.
2. Distinguir el límite de IOPS del disco del límite de la máquina y localizar cuál es el cuello de botella real.
3. Elegir entre conjunto de disponibilidad, zonas y conjunto de escalado según el fallo que se quiere sobrevivir.
4. Configurar un conjunto de escalado con orquestación flexible, mezcla de prioridades y política de reducción explícita.
5. Seleccionar entre los cuatro balanceadores de Azure a partir del ámbito, la capa y el tiempo de conmutación aceptable.

Conceptos centrales

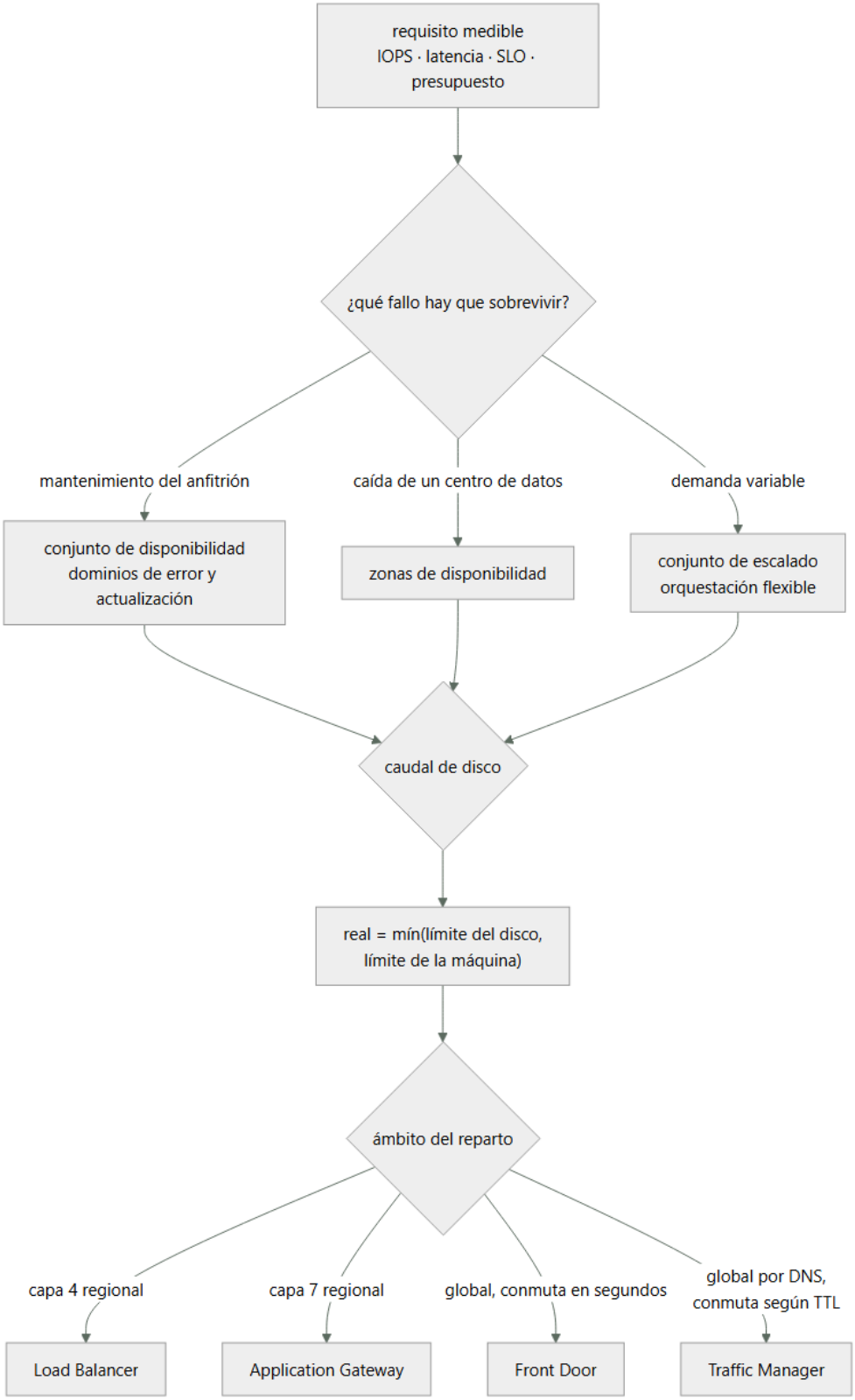
Concepto	Comprensión verificable
tamaño de máquina virtual	Cadena con estructura: familia, número de vCPU y letras aditivas. s habilita discos premium, d añade disco temporal local, a indica AMD, p indica ARM. Sin la letra, la capacidad no existe.
límite de la máquina frente al del disco	Cada tamaño tiene un tope propio de IOPS y de rendimiento de disco, distinto del que ofrece el disco. El caudal real es el menor de los dos.
almacenamiento en caché del host	Modo de caché del disco en el anfitrión: None, ReadOnly o ReadWrite. Cambia el límite aplicable y, en discos de registro de transacciones, ReadWrite puede costar datos.
dominio de actualización	Grupo de máquinas que el mantenimiento planificado de la plataforma reinicia a la vez. Un conjunto de disponibilidad con cinco garantiza que como mucho una quinta parte se reinicie de golpe.
orquestación flexible	Modo de conjunto de escalado en el que las instancias son máquinas virtuales reales, gestionables una a una y con tamaños o prioridades mezcladas. Es el modo por defecto.
aviso de desalojo	Notificación de eventos programados que anuncia el desalojo de una máquina de excedente. En Azure son 30 segundos, no dos minutos: solo sirve para abortar, no para drenar.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Aplicado a esta clase, separa siempre cuatro planos: intención (qué necesita el usuario), configuración (qué declaramos), estado observado (qué existe de verdad) y evidencia (cómo sabemos que cumple). Confundirlos produce diseños que se ven correctos en un diagrama pero fallan al operar.

Flujo de razonamiento



Desarrollo

1. Las letras del tamaño son la especificación

La clase 028 dejó establecido que el nombre de una instancia es una especificación, no una etiqueta. En Azure el alfabeto es otro y el orden importa:

```
Standard_D8ads_v6
          ■ ■■■ ■■■ generación
          ■ ■■■■■■■■ s : admite discos premium
          ■ ■■■■■■■■ d : incluye disco temporal local
          ■ ■■■■■■■■ a : procesador AMD
          ■■■■■■■■■■ D8 : familia de propósito general, 8 vCPU
```

Las familias, en una línea cada una:

```
B   ráfaga con créditos      cargas ociosas la mayor parte del tiempo
D   propósito general        4 GB de memoria por vCPU
E   memoria                  8 GB por vCPU — bases de datos, cachés
F   cómputo                  2 GB por vCPU — cálculo, compilación
L   almacenamiento           NVMe local grande y muy rápido
M   memoria extrema          cargas SAP y bases de datos monolíticas
N   GPU
```

Las dos letras que más problemas dan son las que no están:

Sin s, el tamaño no admite discos premium. Se descubre al desplegar, con un error de compatibilidad que no menciona la letra, y obliga a cambiar el tamaño —y por tanto a reiniciar— con la plantilla ya en producción.

Sin d, no hay disco temporal local. Y aquí está el error silencioso: mucha automatización heredada asume que existe /mnt en Linux o D: en Windows porque «siempre estuvo ahí». En un D8sv5 no está. El servicio arranca, escribe en la raíz del disco del sistema operativo, y semanas después el disco se llena o el rendimiento cae, porque el disco del sistema no está dimensionado para eso.

Y una advertencia sobre el disco temporal cuando sí existe: es efímero de verdad. Se borra al desasignar la máquina y al moverla de anfitrión, sin aviso y sin copia. Sirve para archivos de intercambio, cachés reconstruibles y ficheros temporales de compilación. Poner ahí datos de una base de datos es la versión de Azure del problema que cerró la clase 028: el estado local es el enemigo de la elasticidad, y en este caso ni siquiera hay que escalar para perderlo.

Sobre la familia B basta con una nota, porque el mecanismo de créditos ya se trabajó en 028: es el mismo desplome, con distinta contabilidad. La diferencia práctica es que la señal se llama CPU Credits Remaining y hay que graficarla junto al porcentaje de CPU. Sin ella, un servicio que agota créditos aparece en el panel como una CPU al 20 % —el límite base— mientras las latencias se disparan, exactamente el falso negativo que describía aquella clase.

2. Dos límites de disco, y casi siempre manda el que no compraste

Este es el punto donde Azure se aparta más de AWS, y produce la conversación de «duplicamos el disco y no cambió nada».

En el SSD premium clásico, el rendimiento se compra comprando tamaño. No hay un parámetro de IOPS que ajustar: cada escalón de tamaño trae su cifra:

```
P10    128 GiB      500 IOPS     100 MB/s
P20    512 GiB    2.300 IOPS     150 MB/s
P30  1.024 GiB    5.000 IOPS     200 MB/s
P40  2.048 GiB    7.500 IOPS     250 MB/s
```

Quien viene de gp3 —donde IOPS y tamaño se compran por separado— compra tamaño que no necesita para conseguir IOPS que sí. Las alternativas modernas lo desacoplan: SSD premium v2 y Ultra Disk permiten fijar tamaño, IOPS y rendimiento de forma independiente, que es el comportamiento esperado; a cambio, no admiten caché del host y v2 tiene restricciones de zona.

Pero la cifra del disco es solo la mitad. Cada tamaño de máquina tiene su propio tope de IOPS y de MB/s, y el caudal real es el menor de los dos:

```
Standard_D2s_v5         3.750            85
Standard_D4s_v5         6.400           145
Standard_D8s_v5        12.800           290
Standard_D16s_v5       25.600           600
```

```
P40 (7.500 IOPS) en un D2s_v5 (3.750)  →  real: 3.750
cambiar a P50 no mueve la cifra
cambiar a D8s_v5 la sube a 7.500
```

La regla operativa es corta: antes de comprar disco, mira la fila de la máquina. Y para diagnosticar, la señal que lo distingue es la profundidad de cola junto al porcentaje de uso del disco: si el disco no está saturado y la cola crece, el techo es de la máquina.

La caché del host añade un tercer factor y una decisión con consecuencias:

```
None       lecturas y escrituras van al disco
           → discos de registro de transacciones y cargas de escritura intensa
ReadOnly   lecturas desde la memoria del anfitrión, escrituras al disco
           → discos de datos con lectura predominante; suele ser la mejor opción
ReadWrite  escrituras confirmadas en la caché del anfitrión
           → solo el disco del sistema operativo
```

ReadWrite en un disco de datos o de registro es un error con consecuencias reales: si el anfitrión se reinicia de forma no ordenada, las escrituras confirmadas a la aplicación y aún no bajadas al disco se pierden, y un motor transaccional se encuentra con un registro incoherente. El motivo por el que ocurre es prosaico: ReadWrite es el valor por defecto del disco del sistema, y quien copia la plantilla lo arrastra al resto.

Y un detalle contable: la caché cambia el límite aplicable. Un D8sv5 tiene un tope con caché mayor que sin ella, pero ese tope solo aplica a lo que la caché sirve. Contar con el número grande para dimensionar una carga de escritura es contar con capacidad que no se va a usar.

3. Tres formas de sobrevivir y no todas se combinan

Azure separa en tres construcciones lo que AWS resuelve casi todo con zonas:

	Qué fallo sobrevive	Ámbito	SLA orientativo
Máquina única con disco premium	Ninguno estructural	Un anfitrión	99,9 %
Conjunto de disponibilidad	Fallo de bastidor y mantenimiento planificado	Un centro de datos	99,95 %
Zonas de disponibilidad	Caída de un centro de datos completo	Una región	99,99 %

El conjunto de disponibilidad no tiene equivalente directo en AWS y aporta algo que las zonas no dan: los dominios de actualización. Son la unidad del mantenimiento planificado de la plataforma:

```
dominios de error        (hasta 3)  bastidores distintos: alimentación y red
dominios de actualización (hasta 20) grupos que se reinician por separado
                                     al actualizar el anfitrión
```

Con cinco dominios de actualización y diez máquinas, un mantenimiento reinicia como mucho dos a la vez. Sin conjunto de disponibilidad, nada impide que la plataforma reinicie las diez.

La restricción que hay que conocer antes de dibujar: conjunto de disponibilidad y zonas son excluyentes. Una máquina está en uno o en otras, y cambiar de decisión implica recrearla. En una región con zonas, la elección casi siempre es zonas; el conjunto de disponibilidad queda para regiones sin ellas.

El conjunto de escalado es la pieza que integra ambas cosas con elasticidad, y tiene dos modos:

	Uniforme	Flexible
Las instancias son	Objetos del conjunto	Máquinas virtuales reales
Tamaños mezclados	No	Sí
Excedente y bajo demanda mezclados	No	Sí
Gestión individual	Limitada	az vm funciona sobre ellas
Reparto por dominios de error	Implícito	Explícito y configurable

Flexible es el modo por defecto y el que conviene salvo que se necesite escalar a miles de instancias idénticas muy rápido. La razón práctica: permite mezclar prioridades —una base bajo demanda y el pico en excedente— y deja diagnosticar una instancia concreta con las mismas herramientas que cualquier otra máquina.

El autoescalado vive en Azure Monitor, no en el conjunto. Los cuatro parámetros y el efecto de la histéresis ya se trabajaron en la clase 029; lo que cambia aquí son dos cosas concretas:

```
política de reducción    Default | NewestVM | OldestVM
                         decide a quién apagar; con Default, el reparto
                         entre zonas se mantiene equilibrado
perfiles                 por defecto + perfiles con horario
                         para carga predecible, un perfil horario evita
                         escalar por reacción
```

Y la trampa compartida con AWS, que conviene repetir porque cuesta una noche: la memoria no es una métrica del anfitrión. El porcentaje de CPU se publica sin instalar nada; la memoria y el espacio en disco exigen el agente de Azure Monitor. Una regla de escalado por memoria configurada sin agente no falla: no dispara nunca.

4. Cuatro balanceadores y la conmutación que depende del TTL

El reparto de Azure no coincide con el de AWS, y traducir «esto es el ALB» produce elecciones equivocadas:

Servicio	Capa	Ámbito	Conmuta en	Equivalente aproximado
Load Balancer	4	Regional	Segundos	Network Load Balancer
Application Gateway	7	Regional	Segundos	Application Load Balancer + WAF
Front Door	7	Global, anycast	Segundos	CloudFront + enrutamiento global
Traffic Manager	DNS	Global	Lo que dure el TTL y la caché del cliente	Políticas de enrutamiento de Route 53

La fila que hay que leer dos veces es la última. Traffic Manager no ve el tráfico: responde consultas de DNS. Cuando una región cae, la conmutación no ocurre hasta que cada cliente vuelve a resolver el nombre, y los clientes no respetan el TTL de forma uniforme:

```
TTL de 60 s + caché del resolutor + caché de la JVM o del navegador
→ conmutación observada de varios minutos, con una cola larga
   de clientes que siguen yendo a la región caída
```

Para un requisito de conmutación en segundos, la respuesta es Front Door: el cliente conecta siempre a la misma dirección anycast y el borde decide a qué origen enviar. Traffic Manager sigue siendo útil para lo que sí resuelve bien: dirigir por proximidad geográfica, repartir entre destinos que no son HTTP y encaminar hacia recursos fuera de Azure.

Sobre el Load Balancer estándar hay cuatro hechos que producen incidentes:

Es denegar por defecto. El SKU básico se retiró el 30 de septiembre de 2025, y el estándar no acepta tráfico entrante si un NSG no lo permite explícitamente — el básico sí lo aceptaba. Una migración literal deja el servicio inalcanzable con toda la configuración aparentemente correcta.

No da salida por tener entrada. Una regla de balanceo entrante no proporciona conectividad saliente. Unido a la retirada del acceso saliente predeterminado de la clase 039, un despliegue nuevo se queda literalmente sin salida hasta que se declara un NAT Gateway o una regla de salida.

Las sondas llegan desde 168.63.129.16. Es una dirección virtual de la plataforma, la misma que sirve DHCP, DNS y el canal del agente de la máquina. Bloquearla no solo tumba las sondas: deja al agente sin comunicación, así que las extensiones dejan de responder y las operaciones desde el portal se quedan colgadas. Es un fallo con dos síntomas que parecen no tener relación.

La sonda debe reflejar disponibilidad real. Vale aquí íntegra la lección de la clase 029: un / que devuelve 200 mientras la base de datos no responde mantiene en rotación a una instancia que no puede servir. La forma correcta es un punto de comprobación que verifique las dependencias críticas, con un umbral de fallos consecutivos que tolere un parpadeo.

Un detalle propio de Azure que aparece en migraciones de bases de datos con clúster: la IP flotante —retorno directo desde el servidor— hace que el destino vea la dirección del frontal en lugar de la suya. Es imprescindible para grupos de disponibilidad de SQL Server y rompe cualquier servicio que no espere ese comportamiento. Y la regla de puertos de alta disponibilidad, que balancea todos los puertos y protocolos a la vez, existe para poner dispositivos virtuales de red en alta disponibilidad detrás de un balanceador interno: es la pieza que faltaba en el diseño de concentrador y radio de la clase 039.

5. Excedente: 30 segundos cambian el diseño

Las máquinas de excedente cuestan entre un 60 % y un 90 % menos y se retiran cuando Azure necesita la capacidad. La clase 028 fijó el criterio de los modelos de compra; lo que cambia aquí es un número:

```
AWS   aviso de interrupción de excedente   ~120 s
Azure aviso de desalojo                     ~30 s
```

Treinta segundos no dan para drenar conexiones, terminar una petición larga ni volcar un estado grande. Dan para abortar de forma ordenada: dejar de aceptar trabajo, confirmar o descartar la unidad en curso y salir. Cualquier diseño que dependa de un drenado educado sobre excedente en Azure está apostando a que el aviso llegue antes de lo prometido.

El aviso se consulta contra el servicio de metadatos, no llega solo:

```
$ curl -s -H Metadata:true \
    "http://169.254.169.254/metadata/scheduledevents?api-version=2020-07-01" | jq -r \
    '.Events[] | "\(.EventType) \(.NotBefore)"'
Preempt  Mon, 03 Aug 2026 11:42:07 GMT
```

Y la política de desalojo decide qué queda después:

```
Deallocate   la máquina se detiene y se conserva; sigue pagando el disco
             y puede volver a arrancar cuando haya capacidad
Delete       se elimina; no queda nada que pagar ni que recuperar
```

Para un conjunto de escalado, Delete es casi siempre lo correcto: máquinas desasignadas que nadie recuerda son una partida silenciosa en la factura y una lista de recursos huérfanos. Deallocate tiene sentido en una estación de trabajo con estado en su disco.

La forma sensata de usar excedente es mezclarlo, y para eso hace falta orquestación flexible:

```
$ az vmss create -g rg-app -n vmss-catalogo --orchestration-mode Flexible \
    --instance-count 4 --zones 1 2 3 \
    --priority Spot --eviction-policy Delete --max-price -1
```

```
base bajo demanda   capacidad mínima que sostiene el SLO aunque se vaya todo el excedente
pico en excedente   el resto
```

El dimensionado de la base no es una preferencia: es la carga que el servicio debe seguir atendiendo si el excedente desaparece entero en el mismo minuto, que es un escenario real cuando la región se queda sin capacidad. Y `--max-price -1` significa «acepto pagar hasta el precio bajo demanda», que evita un desalojo por precio además del desalojo por capacidad: fijar un máximo bajo añade una segunda causa de interrupción que casi nunca compensa.

Ejemplo trabajado

CloudShop traslada la capa de cómputo a Azure con la arquitectura de las clases 028 y 029. La traducción es correcta en el papel y produce cuatro problemas medibles.

Punto de partida:

```
base de datos   Standard_D2s_v5  + disco P40 (2 TiB, 7.500 IOPS), caché ReadWrite
catálogo        4 × Standard_D4s_v5 en conjunto de escalado uniforme
reparto         Load Balancer básico + Traffic Manager entre dos regiones
```

Problema 1 — se duplicó el disco y el rendimiento no se movió.

La base de datos se queda en 3.750 IOPS bajo carga, con la cola de disco creciendo.

```
$ az monitor metrics list --resource $VM_ID --metric "Data Disk IOPS Consumed Percentage" \
    --aggregation Maximum --interval PT1M --query "value[0].timeseries[0].data[-3:].maximum"
[49.8, 50.1, 49.9]
```

El disco está al 50 %. No es el disco. La fila de la máquina lo explica:

```
P40             7.500 IOPS
D2s_v5          3.750 IOPS sin caché   ← el techo real
```

Se corrige por el lado correcto y se aprovecha para dejar de pagar tamaño que no se usa:

```
máquina           D2s_v5 (3.750 IOPS)   D8s_v5 (12.800 IOPS)
disco             P40 · 2 TiB · ~259     P30 · 1 TiB · ~135
IOPS reales             3.750                 5.000
costo de disco     259 USD/mes            135 USD/mes
```

El volumen ocupado eran 640 GiB: la P40 se había comprado por sus IOPS, que la máquina nunca dejó alcanzar.

Problema 2 — un reinicio del anfitrión deja el registro de transacciones incoherente.

Durante un mantenimiento no planificado, el motor no arranca y reporta escrituras confirmadas que no están en disco.

```
$ az vm show -g rg-datos -n vm-db-01 \
    --query "storageProfile.dataDisks[].{lun:lun,cache:caching}" -o tsv
0   ReadWrite
1   ReadWrite
```

Ambos discos de datos con ReadWrite, heredado de la plantilla del disco del sistema. Se corrige por función:

```
$ az vm update -g rg-datos -n vm-db-01 --disk-caching 0=ReadOnly 1=None
```

```
LUN 0  datos      ReadOnly   lectura predominante, la caché ayuda
LUN 1  registro   None       ninguna escritura confirmada fuera del disco
```

Y se añade la consecuencia operativa que faltaba: la restauración se prueba, no se supone. La prueba de recuperación pasa a ejecutarse mensualmente con evidencia, siguiendo el criterio de la clase 030 — replicación no es copia de seguridad.

Problema 3 — la migración del balanceador deja el servicio inalcanzable.

Al sustituir el balanceador básico por el estándar, el frontal deja de responder. La configuración es idéntica.

```
$ az network watcher test-ip-flow --vm cat-01 --direction Inbound --protocol TCP \
    --local 10.20.4.11:8080 --remote 168.63.129.16:62000
Access: Deny   RuleName: defaultSecurityRules/DenyAllInBound
```

Dos cosas a la vez: el estándar es denegar por defecto y no había NSG que permitiera nada, porque el básico nunca lo había exigido. Además, no había salida.

```
$ az network nsg rule create -g rg-red --nsg-name nsg-app -n allow-lb-probe \
    --priority 100 --direction Inbound --access Allow --protocol '*' \
    --source-address-prefixes AzureLoadBalancer --destination-port-ranges '*'
$ az network nsg rule create -g rg-red --nsg-name nsg-app -n allow-appgw-8080 \
    --priority 110 --direction Inbound --access Allow --protocol Tcp \
    --source-address-prefixes 10.20.4.128/26 --destination-port-ranges 8080
$ az network vnet subnet update -g rg-red --vnet-name vnet-spoke-app -n snet-app \
    --nat-gateway natgw-spoke-app
```

Y se corrige la sonda, que apuntaba a /:

```
antes    GET /          200 mientras la base de datos no respondía
después  GET /readyz    comprueba base de datos y caché;
                        3 fallos consecutivos para retirar de rotación
```

Problema 4 — una conmutación de región que tardó nueve minutos.

Un simulacro corta la región primaria. Traffic Manager marca el destino como degradado en 40 s; el tráfico tarda mucho más en moverse.

```
t+0:00   se corta la región primaria
t+0:40   Traffic Manager marca el destino como degradado
t+1:10   el DNS ya devuelve la región secundaria
t+9:20   el último cliente deja de intentar contra la región caída
```

Los nueve minutos no son del servicio: son de la caché de DNS de los clientes, que no respetan el TTL de 60 s. Se antepone Front Door para el tráfico HTTP y se conserva Traffic Manager solo para el enrutamiento geográfico de un servicio que no es HTTP:

```
simulacro repetido con Front Door
t+0:00   se corta la región primaria
t+0:35   el borde deja de enviar al origen caído
t+0:41   error observado por el cliente: ninguno tras el reintento
```

Y una revisión de costo que no estaba en el plan. El conjunto de escalado uniforme no permitía mezclar prioridades. Al pasarlo a flexible:

```
orquestación                uniforme             flexible
capacidad base          4 × D4s_v5 bajo demanda   2 × D4s_v5 bajo demanda
pico                    escala en bajo demanda    hasta 6 × D4s_v5 excedente
costo mensual de cómputo      ~702 USD              ~412 USD
disco de base de datos         259 USD               135 USD
                          ■■■■■■■■■■            ■■■■■■■■■■
```

```
                961 USD                547 USD   (−43 %)
```

La base de dos instancias bajo demanda no es un número redondeado a ojo: es la capacidad que sostiene el percentil 95 de tráfico observado si todo el excedente desaparece en el mismo minuto. Se comprobó apagando las seis instancias de excedente a la vez y midiendo la latencia resultante:

```
con 2 bajo demanda y 0 excedente, a carga de p95
p99 de latencia   340 ms  (SLO: 500 ms)                                  ✓
```

La lección que esta clase traslada al resto de la parte: en Azure el cuello de botella rara vez está donde se compró la capacidad. El disco, la máquina y la caché tienen topes distintos, y el balanceador y el DNS tienen tiempos de reacción distintos. La cifra que importa siempre es el mínimo de la cadena, y es la que hay que medir antes de firmar un SLO.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-03-azure-core-platform/040-virtual-machines-scale-sets-y-load-balancer/lab.py
```

El laboratorio selecciona el motor de práctica compute y produce labresult.json. El escenario, sus comprobaciones y el artefacto esperado corresponden a esta clase; no requiere credenciales y deja explícito qué debe revalidarse en un sandbox real.

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa servicio-elastico-azure en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es una selección de capacidad justificada y observable. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye servicio-elastico-azure para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

Errores frecuentes

Síntoma	Causa probable	Corrección
Se aumenta el tamaño del disco y el rendimiento no cambia	El tope efectivo es el de la máquina virtual, no el del disco	Consulta las IOPS y MB/s sin caché del tamaño elegido; el caudal real es el mínimo de los dos y se corrige cambiando de tamaño.
La automatización falla porque /mnt o D: no existe	El tamaño elegido no lleva la letra d, así que no tiene disco temporal local	Elige un tamaño con d o retira la dependencia del disco temporal; nunca guardes ahí datos que deban sobrevivir.
Tras un reinicio no ordenado del anfitrión, el motor transaccional reporta escrituras perdidas	El disco de registro tenía caché ReadWrite, heredada de la plantilla del disco del sistema	None en el disco de registro, ReadOnly en los de datos con lectura predominante; ReadWrite solo en el disco del sistema.
Al migrar del balanceador básico al estándar, el servicio queda inalcanzable	El estándar deniega el tráfico entrante salvo que un NSG lo permita, y no proporciona salida por tener reglas de entrada	Añade reglas de NSG para la etiqueta AzureLoadBalancer y para el origen real, y declara la salida con NAT Gateway.
La regla de autoescalado por memoria nunca dispara	La memoria no es una métrica del anfitrión: requiere el agente de Azure Monitor	Instala el agente y verifica que la métrica llega antes de confiar la elasticidad a esa regla.
La conmutación entre regiones tarda minutos pese a que la sonda detecta el fallo en segundos	Traffic Manager conmuta por DNS y los clientes cachean más allá del TTL	Usa Front Door para el tráfico HTTP; reserva Traffic Manager para enrutamiento geográfico y destinos que no son HTTP.

Seguridad, ética y costo

Trabaja con cuentas propias o sandboxes autorizados. No publiques secretos, identificadores reales ni datos personales. Antes de crear recursos pagos define presupuesto, etiquetas y comando de destrucción; después verifica que no queden recursos huérfanos. Los ejemplos locales enseñan contratos, pero no certifican cumplimiento ni disponibilidad de producción.

Preguntas de comprobación

1. ¿Qué capacidades pierdes con un tamaño sin s y sin d, y cuándo lo descubrirías en cada caso?
2. Un disco P40 rinde 3.750 IOPS. ¿Qué medirías para decidir si el problema es el disco o la máquina?
3. ¿Qué aporta un conjunto de disponibilidad que las zonas no dan, y por qué no se pueden combinar?
4. ¿Por qué la orquestación flexible permite una estrategia de costo que la uniforme no?
5. El aviso de desalojo de excedente es de 30 segundos. ¿Qué diseños quedan descartados y cómo se dimensiona la capacidad base?

Referencias

- Microsoft (2025). Sizes for virtual machines in Azure — nomenclatura, letras aditivas y límites por tamaño. <https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/overview>
- Microsoft (2025). Azure managed disk types — escalones de rendimiento, SSD premium v2 y Ultra Disk. <https://learn.microsoft.com/en-us/azure/virtual-machines/disks-types>
- Microsoft (2025). Virtual Machine Scale Sets orchestration modes — flexible frente a uniforme. <https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-orchestration-modes>
- Microsoft (2025). Load-balancing options — Load Balancer, Application Gateway, Front Door y Traffic Manager. <https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/load-balancing-overview>
- Microsoft (2025). Azure Spot Virtual Machines — aviso de desalojo, políticas y precio máximo. <https://learn.microsoft.com/en-us/azure/virtual-machines/spot-vms>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 03 en PDF · Recorrido de Azure en PDF · Manual integral

Anterior	Índice	Siguiente
← 039 · Virtual Network, subredes, NSG, UDR, peering y Private Link	Parte 03 · Programa	041 · Blob Storage, Files, redundancia y lifecycle →

Evaluación — 040 Virtual Machines, Scale Sets y Load Balancer

Preguntas

1. Explica el problema que resuelve esta capacidad sin mencionar un proveedor.
2. Identifica una frontera de responsabilidad y su propietario.
3. Compara dos alternativas mediante confiabilidad, seguridad, costo y operación.
4. ¿Qué demuestra el laboratorio y qué no puede demostrar?
5. Propón un caso límite o una prueba de fallo.

Reto verificable

Entrega servicio-elastico-azure con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

- [ ] Resultado reproducible por una segunda persona.
- [ ] Evidencia vinculada a cada afirmación técnica.
- [ ] Prueba negativa o de fallo incluida.
- [ ] Costos y permisos expresados con unidades y alcance.
- [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.

041 — Blob Storage, Files, redundancia y lifecycle

Parte: 03 — Azure: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: storage · Estado: EXECUTABLECORE

Propósito

Operar el almacenamiento de Azure sabiendo que la unidad de configuración no es el contenedor sino la cuenta, y que casi todo lo que protege los datos son interruptores separados que hay que activar uno a uno. La clase 030 dejó el criterio —versionado, clases, público bloqueado, replicación no es copia—; aquí cambia dónde vive cada control, y aparecen tres decisiones que en AWS no existen: la región emparejada no se elige, la conmutación la inicias tú, y borrar un contenedor no es lo mismo que borrar sus blobs.

Resultados de aprendizaje

Al finalizar podrás:

1. Dimensionar cuentas de almacenamiento sabiendo qué límites son de la cuenta y qué configuraciones arrastra todo lo que hay dentro.
2. Elegir redundancia distinguiendo lo que protege de lo que cuesta, y sabiendo qué ocurre con la cuenta después de una conmutación.
3. Calcular el punto de equilibrio de un nivel de acceso frío antes de mover datos a él.
4. Configurar los interruptores de protección —versionado, eliminación temporal de blob y de contenedor, inmutabilidad— y demostrar cada uno con una prueba de recuperación.
5. Sustituir claves de cuenta y firmas irrevocables por identidad y firmas de delegación de usuario.

Conceptos centrales

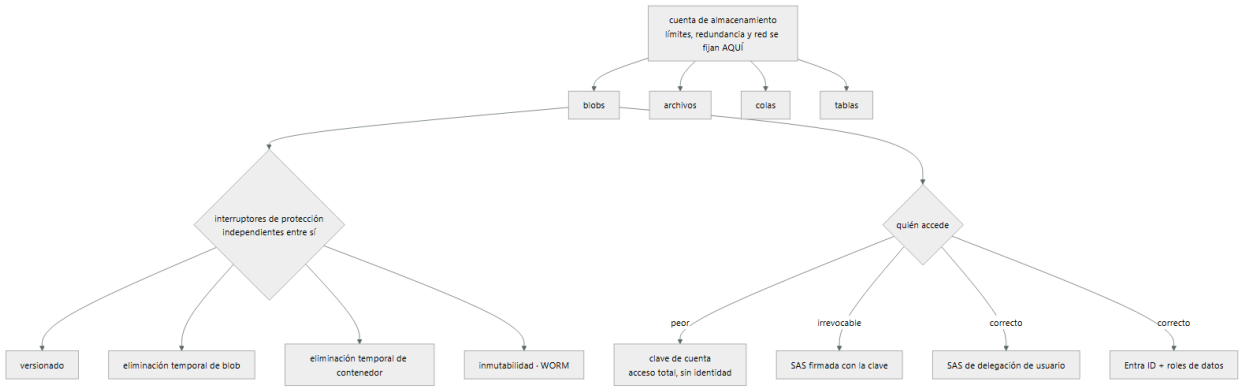
Concepto	Comprensión verificable
cuenta de almacenamiento	Unidad de configuración, facturación y límites. Contiene blobs, archivos, colas y tablas a la vez, y su nombre es una etiqueta DNS global: 3 a 24 caracteres, minúsculas y dígitos, sin guiones.
región emparejada	Destino de la replicación geográfica, asignado por Microsoft y no elegible. Si el dato no puede salir de una jurisdicción, esa asignación puede descartar la redundancia geográfica.
conmutación de cuenta	Cambio de la región principal a la secundaria. La inicia el cliente, pierde lo no replicado y deja la cuenta como LRS, lo que obliga a reconfigurar la redundancia después.
eliminación temporal	Dos interruptores independientes: uno para blobs y otro para contenedores. Activar el primero no protege de borrar el contenedor entero.
firma de delegación de usuario	SAS firmada con una clave derivada de Entra ID en vez de con la clave de la cuenta. Caduca en 7 días como máximo y se puede revocar sin romper el resto.
regla de ciclo de vida	Directiva JSON de la cuenta que mueve o borra por antigüedad. Si solo apunta a baseBlob, las versiones e instantáneas se quedan y la factura no baja.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Aplicado a esta clase, separa siempre cuatro planos: intención (qué necesita el usuario), configuración (qué declaramos), estado observado (qué existe de verdad) y evidencia (cómo sabemos que cumple). Confundirlos produce diseños que se ven correctos en un diagrama pero fallan al operar.

■ Flujo de razonamiento



Desarrollo

1. La cuenta es la unidad, y arrastra todo lo que hay dentro

En AWS el bucket es la unidad: sus políticas, su cifrado y su bloqueo de acceso público valen para él y para nadie más. En Azure la unidad es la cuenta de almacenamiento, y dentro de ella conviven cuatro servicios:

```
cuenta stcloudshopprod
  ■■ blobs      objetos
  ■■ archivos   recursos compartidos SMB y NFS
  ■■ colas      mensajería simple
  ■■ tablas     clave-valor
```

Lo que se decide en la cuenta y no se puede matizar por contenedor:

```
redundancia (LRS, ZRS, GRS, GZRS)
reglas de red y punto de conexión privado
versión mínima de TLS
acceso con clave compartida habilitado o no
acceso público anónimo permitido o no
nivel de acceso por defecto
cifrado y clave gestionada por el cliente
```

Esa lista es la razón por la que una cuenta para todo es un error de diseño, aunque parezca ordenado. Datos con requisitos distintos de residencia, de retención o de exposición terminan compartiendo la misma configuración, y el único modo de separarlos después es copiar terabytes.

Además la cuenta tiene límites propios que un bucket no tiene:

```
capacidad                    5 PiB (ampliable por solicitud)
frecuencia de solicitudes    ~20.000 por segundo
salida en la región primaria  límite por cuenta, no por contenedor
```

Un servicio ruidoso puede limitar a los demás inquilinos de la misma cuenta. El síntoma es un 503 ServerBusy intermitente en un servicio que no cambió nada, causado por otro que sí. La respuesta a x-ms-request-id y el contador Throttling Error lo identifican en un minuto; el arreglo es repartir en varias cuentas, no reintentar más fuerte.

Y el nombre importa más de lo que parece: es una etiqueta DNS global, de 3 a 24 caracteres, minúsculas y dígitos, sin guiones ni puntos. Una convención de nombres que incluya guiones —muy común al llegar de AWS— no se puede aplicar aquí, así que conviene fijarla antes de crear la primera:

```
st  cloudshop  prod  weu  01
■■  ■■■■■■■■  ■■■■  ■■■  ■■ secuencia
■■  ■■■■■■■■  ■■■■  ■■■■■■■■■ región
■■  ■■■■■■■■  ■■■■■■■■■■■■■■■ entorno
■■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■ sistema
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ tipo de recurso
```

2. Redundancia: la región no se elige y la conmutación la inicias tú

Cinco opciones, dos ejes: cuántas copias y dónde.

	Copias	Sobrevive a	Lectura en la secundaria
LRS	3 en un centro de datos	Fallo de disco o bastidor	—
ZRS	3 en tres zonas	Caída de un centro de datos	—
GRS	3 locales + 3 remotas	Caída de la región	No
GZRS	3 zonas + 3 remotas	Caída de la región, con zonas en la principal	No
RA-GRS / RA-GZRS	Igual	Igual	Sí, con un nombre -secondary

Tres hechos cambian cómo se diseña con esto, y ninguno tiene equivalente directo en la replicación entre regiones de AWS:

La región emparejada la asigna Microsoft. No se elige el destino. Para una carga sujeta a residencia de datos, eso puede descalificar la redundancia geográfica entera: si el par asignado está fuera de la jurisdicción aprobada, la opción correcta es ZRS más una replicación de objetos explícita hacia una cuenta en la región que sí lo está. Es más trabajo y es la única que se puede documentar ante un auditor.

La replicación es asíncrona y su retraso no es un compromiso de servicio. El indicador es Last Sync Time, y es la única fuente honesta del RPO real:

```
$ az storage account show -n stcloudshopprod -g rg-datos \
    --expand geoReplicationStats --query "geoReplicationStats.lastSyncTime" -o tsv
2026-08-01T09:47:12Z
```

Todo lo escrito después de esa marca no está en la secundaria. Un RPO declarado de 15 minutos que no se contrasta con esta métrica es una aspiración, no un dato.

La conmutación no es automática y deja la cuenta en LRS. Nadie la ejecuta por ti: es una operación que inicia el cliente, y después de completarse la cuenta queda como LRS en la que era región secundaria. Es decir:

```
antes del incidente   GRS   principal weu, secundaria nue
durante la conmutación      se pierde lo posterior a lastSyncTime
después                LRS   una sola región, sin redundancia geográfica
acción pendiente             reconfigurar a GRS y esperar la replicación inicial
```

Ese último renglón es el que sorprende: justo después de un incidente regional, la plataforma está menos protegida que antes, y volver a estarlo tarda lo que tarde copiar todo el volumen. Un plan de recuperación que no incluya ese paso está incompleto.

Y el cambio entre modelos tampoco es un interruptor: pasar de LRS a ZRS exige una conversión —solicitada o mediante copia manual—, con ventana y sin garantía de instantaneidad. Conviene elegir bien al crear la cuenta, porque corregirlo después cuesta tiempo y tráfico.

3. Niveles de acceso: calcula el punto de equilibrio antes de mover nada

Cuatro niveles, con mínimos de permanencia que son la mitad de la decisión:

	Almacenamiento aprox.	Permanencia mínima	Recuperación
Caliente	0,0184 USD/GiB/mes	—	Inmediata
Frío (cool)	0,0100	30 días	Inmediata, con cargo por GiB
Muy frío (cold)	0,0036	90 días	Inmediata, con cargo mayor
Archivo	0,0010	180 días	Hay que rehidratar: hasta 15 h

La clase 030 ya estableció que el ahorro por nivel a veces no lo es. Aquí está la aritmética con las cifras de Azure, para 1 TiB:

```
ahorro mensual al pasar de caliente a frío
  (0,0184 − 0,0100) × 1.024 GiB = 8,60 USD/mes

coste de leer ese TiB una vez al mes
  1.024 GiB × 0,01 USD/GiB de recuperación = 10,24 USD
```

Leer el conjunto entero una sola vez al mes ya destruye el ahorro. El punto de equilibrio está en releer menos del 84 % del volumen cada mes; por encima, el nivel frío sale más caro que el caliente además de más lento.

Y encima está la penalización por eliminación temprana, que se cobra prorrateada: borrar a los 10 días un blob en nivel frío factura los 20 días restantes de los 30 mínimos. En un flujo con mucha rotación —archivos temporales, informes diarios que se sustituyen— mover al nivel frío puede multiplicar la factura en lugar de reducirla.

El nivel archivo es una decisión distinta, no un escalón más: el blob no se puede leer. Hay que rehidratarlo, y eso tiene tiempo y precio:

```
prioridad estándar   hasta 15 h
prioridad alta       menos de 1 h para blobs de menos de 10 GiB, con sobrecoste
```

Quince horas es incompatible con casi cualquier RTO de servicio y perfectamente compatible con una obligación legal de conservar siete años. Esa es la frontera: archivo es para lo que hay que guardar, no para lo que hay que poder leer.

Las reglas de ciclo de vida automatizan el movimiento, y tienen dos comportamientos que hay que anticipar:

```
{
  "rules": [{
    "name": "facturas",
    "type": "Lifecycle",
    "definition": {
      "filters": {"blobTypes": ["blockBlob"], "prefixMatch": ["facturas/"]},
      "actions": {
        "baseBlob": {
          "tierToCool":    {"daysAfterModificationGreaterThan": 30},
          "tierToArchive": {"daysAfterModificationGreaterThan": 365}
        },
        "version":  {"delete": {"daysAfterCreationGreaterThan": 90}},
        "snapshot": {"delete": {"daysAfterCreationGreaterThan": 90}}
      }
    }
  }]
}
```

Primero: las secciones version y snapshot son las que casi nadie escribe, y son la causa de que la factura no baje después de una limpieza. Con versionado activo, borrar un blob no libera nada: crea una versión. Es el mismo mecanismo que el marcador de borrado de la clase 030, con otro nombre y la misma consecuencia contable.

Segundo: la directiva se evalúa una vez al día y la primera ejecución puede tardar hasta 48 horas. No es un fallo de configuración, es su cadencia; comprobarla a los diez minutos y volver a escribirla es la forma habitual de perder una tarde.

Y daysAfterLastAccessTimeGreaterThan —mover por último acceso en vez de por última modificación— exige activar el seguimiento de acceso, que tiene su propio costo por transacción. En un contenedor con lecturas muy frecuentes puede costar más que lo que ahorra el cambio de nivel.

4. Cuatro interruptores para proteger lo mismo, y ninguno cubre al otro

Aquí está la trampa más cara de esta clase. Los mecanismos de protección son independientes, y activar el evidente deja el hueco:

```
versionado                        conserva una versión por cada sobrescritura o borrado
eliminación temporal de blob      recupera blobs borrados durante N días
eliminación temporal de contenedor recupera CONTENEDORES borrados durante N días
restauración a un punto anterior  devuelve un rango de blobs a un instante
inmutabilidad                     impide modificar o borrar durante un plazo
```

La eliminación temporal de blob no protege de borrar el contenedor. Es literal: con eliminación temporal de blob a 7 días y eliminación temporal de contenedor desactivada, un az storage container delete se lleva todo el contenido de forma irreversible. El interruptor que parecía cubrirlo cubría otra cosa.

La configuración completa, y su dependencia entre piezas:

```
$ az storage account blob-service-properties update -n stcloudshopprod -g rg-datos \
    --enable-versioning true \
    --enable-delete-retention true --delete-retention-days 30 \
    --enable-container-delete-retention true --container-delete-retention-days 30 \
    --enable-change-feed true \
    --enable-restore-policy true --restore-days 29
```

La restauración a un punto anterior exige versionado, fuente de cambios y eliminación temporal de blob, y su ventana debe ser menor que la de retención. Activarla suelta da un error que no explica cuál de las tres falta.

La inmutabilidad es otra cosa y responde a otro requisito. Se aplica a contenedor o a versión, en dos formas:

```
retención por tiempo   ningún borrado ni modificación durante N días
retención legal        indefinida hasta que se retire la etiqueta
```

Y el matiz que la hace útil ante un auditor: mientras la directiva está bloqueada, el plazo se puede alargar y no se puede acortar ni eliminar — tampoco por el propietario de la suscripción, tampoco por un administrador global. Es la diferencia entre una política y un control: la política la cambia quien tiene permiso, el control no lo cambia nadie. La contrapartida hay que aceptarla de antemano: la cuenta no se puede borrar mientras haya datos bajo retención, ni siquiera si se creó por error con un plazo de siete años.

Una última pieza que la clase 030 dejó dicha y aquí tiene dos interruptores en vez de uno: el acceso público anónimo se controla en la cuenta y en el contenedor, y basta con que el de la cuenta esté abierto y alguien marque un contenedor como público para exponerlo:

```
$ az storage account update -n stcloudshopprod -g rg-datos \
    --allow-blob-public-access false
```

Con eso, ningún contenedor puede ser público aunque su configuración diga que lo es. Es el equivalente al bloqueo en el nivel de la cuenta, y es el que hay que auditar: comprobar contenedor por contenedor es trabajo que se deshace solo.

5. Claves, firmas y la que no se puede revocar

Cuatro formas de acceder, ordenadas de peor a mejor:

Clave de cuenta. Dos claves, rotables. Conceden acceso total a los cuatro servicios de la cuenta y no llevan identidad: el registro dice qué se hizo, no quién. Una clave filtrada es la cuenta entera. La instrucción es desactivarlas:

```
$ az storage account update -n stcloudshopprod -g rg-datos --allow-shared-key-access false
```

SAS firmada con la clave de cuenta. Es cómoda y tiene un defecto que define la decisión: no se puede revocar. Una firma emitida con vencimiento en 2029 sigue siendo válida hasta 2029, y la única forma de invalidarla es rotar la clave con la que se firmó — lo que rompe a la vez a todos los demás que la usan. El repositorio se limpia, la firma sigue funcionando.

Un paliativo parcial es la directiva de acceso almacenada: la firma referencia una directiva del contenedor y borrar la directiva invalida las firmas asociadas. Requiere haberlo previsto al emitirlas.

SAS de delegación de usuario. Se firma con una clave derivada de Entra ID, no con la clave de cuenta:

```
$ az storage blob generate-sas --account-name stcloudshopprod \
    --container-name facturas --name 2026-07.pdf \
    --permissions r --expiry 2026-08-01T18:00Z --as-user --auth-mode login -o tsv
```

Caduca en 7 días como máximo, hereda los permisos del principal que la emite —no puede conceder más de lo que ese principal tiene— y se revoca retirando el rol o revocando las claves de delegación, sin tocar nada más. Es la opción correcta para dar acceso temporal a un objeto concreto.

Entra ID con roles de datos. Es la vía por defecto para servicios. Y aquí vuelve, con consecuencias, la distinción de la clase 038:

```
Storage Account Contributor       gestiona la cuenta y LEE SUS CLAVES
                                  → sin dataActions, pero con la clave se accede a todo
Storage Blob Data Reader          lee los datos, no gestiona la cuenta
Storage Blob Data Contributor     lee y escribe los datos
```

La primera fila es una escalada disfrazada de permiso administrativo: quien puede leer las claves puede hacer todo lo que hacen las claves, sin necesitar ningún rol de datos y sin dejar rastro de identidad. Con allowSharedKeyAccess desactivado, ese camino se cierra: la clave existe y no sirve para autenticarse.

Y para Azure Files conviene anticipar una decisión de identidad, porque no es simétrica con los blobs:

	Estándar	Premium
Soporte	HDD, cobro por transacción	SSD, capacidad aprovisionada

	Estándar	Premium
Rendimiento	Variable	IOPS en función de los GiB aprovisionados
NFS	No	Sí, con punto de conexión privado
Se factura	Lo usado	Lo aprovisionado

La última fila es la que produce facturas inesperadas: un recurso compartido premium de 5 TiB aprovisionado y 300 GiB usados factura 5 TiB. Y para SMB con identidad de dominio hace falta integrar con AD DS o con Entra Domain Services; sin eso, el acceso es con clave de cuenta, que es justo lo que se acaba de desactivar. Conviene resolverlo antes de migrar un recurso compartido de archivos, no después.

Ejemplo trabajado

CloudShop lleva a Azure el almacenamiento diseñado en la clase 030. La configuración inicial parece completa: versionado activo, replicación geográfica y niveles de acceso. Cuatro sucesos demuestran que ninguna de las tres cosas hacía lo que el equipo creía.

Estado inicial:

```
cuenta stcloudshopprod  GRS  ·  una sola cuenta para facturas, medios y registros
versionado                     activo
eliminación temporal de blob   7 días
eliminación temporal de contenedor  desactivada
acceso con clave compartida    habilitado
```

Suceso 1 — un contenedor borrado por error se lleva 412 GiB.

Un script de limpieza mal parametrizado borra facturas en lugar de facturas-tmp.

```
$ az storage container restore -n facturas --account-name stcloudshopprod
ERROR: Container soft delete is not enabled for this account.
```

La eliminación temporal de blob estaba activa y no cubría esto. No hay recuperación. Se reconstruye desde el sistema de facturación —tres días de trabajo— y se cierran los cuatro interruptores, no uno:

```
$ az storage account blob-service-properties update -n stcloudshopprod -g rg-datos \
    --enable-versioning true \
    --enable-delete-retention true --delete-retention-days 30 \
    --enable-container-delete-retention true --container-delete-retention-days 30 \
    --enable-change-feed true --enable-restore-policy true --restore-days 29
```

Y se prueba la recuperación en lugar de suponerla:

```
$ az storage container delete -n prueba-recuperacion --account-name stcloudshopprod
$ az storage container restore -n prueba-recuperacion --account-name stcloudshopprod
$ az storage blob list -c prueba-recuperacion --account-name stcloudshopprod -o tsv | wc -l
50                                                                              ✓
```

Además, las facturas tienen obligación legal de siete años, así que se añade inmutabilidad bloqueada — con la consecuencia aceptada por escrito de que la cuenta ya no se podrá eliminar durante ese plazo.

Suceso 2 — se borran 3 TiB y la factura no baja.

```
datos visibles      1,8 TiB
facturado           4,9 TiB
```

```
$ az storage account show -n stcloudshopprod -g rg-datos \
    --query "{v:blobServiceProperties.isVersioningEnabled}"
$ az storage account management-policy show --account-name stcloudshopprod -g rg-datos \
    --query "policy.rules[0].definition.actions" -o json
{"baseBlob": {"tierToCool": {"daysAfterModificationGreaterThan": 30}}}
```

La regla solo tocaba baseBlob. Con versionado activo, cada borrado había creado una versión que nadie retiraba nunca. Es el marcador de borrado de la clase 030 con otro nombre. Se completa la directiva con version y snapshot, y se espera a su cadencia diaria en lugar de reescribirla:

```
facturado                4,9 TiB     1,9 TiB
costo mensual de blobs   92,20 USD   35,80 USD
```

Suceso 3 — una firma en una aplicación móvil, válida hasta 2029.

Una revisión de seguridad encuentra una SAS incrustada en el paquete de la aplicación.

```
se=2029-01-01T00%3A00%3A00Z&sp=racwdl&sr=c&sig=…
```

Permisos de lectura, escritura, borrado y listado sobre el contenedor entero, durante tres años. Firmada con la clave de la cuenta, así que revocarla exige rotar la clave, y la clave la usan seis servicios más.

La secuencia, en este orden y no en otro:

```
1. migrar los seis servicios a identidad administrada + roles de datos
2. verificar que ninguno usa ya la clave  (métrica de autenticación por clave = 0)
3. rotar las dos claves  → la firma de 2029 deja de valer
4. desactivar el acceso con clave compartida
5. la aplicación móvil pasa a pedir una SAS de delegación de usuario
   al backend, de 15 minutos y para un solo blob
```

```
$ az storage account update -n stcloudshopprod -g rg-datos --allow-shared-key-access false
$ az storage blob list -c facturas --account-name stcloudshopprod \
    --account-key $CLAVE_ANTIGUA -o tsv
KeyBasedAuthenticationNotPermitted                                    ✓
```

Suceso 4 — el simulacro de región revela dos supuestos falsos.

El plan decía «GRS conmuta a la región emparejada». El simulacro mide otra cosa:

```
$ az storage account show -n stcloudshopprod -g rg-datos --expand geoReplicationStats \
    --query "geoReplicationStats.{estado:status,ultima:lastSyncTime}" -o tsv
Live   2026-08-01T09:36:00Z
```

Con la hora del corte a las 09:47, el RPO real de ese momento era de 11 minutos, no de cero. Y la conmutación no ocurre sola: hay que ejecutarla, y al terminar la cuenta queda como LRS.

```
conmutación                          automática      manual, ~1 h
RPO                                    0 min          11 min
redundancia después de conmutar         GRS            LRS
```

Además, la región emparejada asignada estaba fuera de la jurisdicción aprobada para los datos de facturación. La corrección separa lo que nunca debió compartir cuenta:

```
facturas   ■                              GZRS + replicación de objetos
medios     ■■ una cuenta GRS                a una región elegida y aprobada
registros  ■                  medios:      GRS, ciclo de vida a frío a 30 días
                              registros:   LRS, borrado a 90 días
```

Resumen del rediseño:

```
cuentas de almacenamiento                    1            3
interruptores de protección activos          2 de 5       5 de 5
recuperación de contenedor probada           no           sí, mensual
acceso con clave compartida                  sí           no
firmas irrevocables en circulación           1            0
RPO documentado y medido                     no           sí, 11 min
datos facturados                           4,9 TiB      1,9 TiB
costo mensual de almacenamiento           92,20 USD    41,60 USD
```

La lección que esta clase traslada al resto de la parte: en Azure, la protección de datos no es una propiedad que se activa sino una lista de interruptores independientes, y ninguno de ellos cubre el hueco del de al lado. La única forma honesta de saber cuáles están activos es borrar algo a propósito y recuperarlo.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-03-azure-core-platform/041-blob-storage-files-redundancia-y-lifecycle/lab.py
```

El laboratorio selecciona el motor de práctica storage y produce labresult.json. El escenario, sus comprobaciones y el artefacto esperado corresponden a esta clase; no requiere credenciales y deja explícito qué debe revalidarse en un sandbox real.

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.

4. Materializa storage-gobernado-azure en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es una política de durabilidad, acceso, retención y costo. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye storage-gobernado-azure para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
Se borra un contenedor y no hay forma de recuperarlo pese a tener eliminación temporal	La eliminación temporal de blob y la de contenedor son interruptores distintos	Activa ambos y demuéstralo borrando y restaurando un contenedor de prueba cada mes.
Se liberan terabytes y la factura no cambia	Con versionado activo, borrar crea una versión, y la regla de ciclo de vida solo apuntaba a baseBlob	Añade las secciones version y snapshot a la directiva y espera a su evaluación diaria.
Una SAS filtrada no se puede revocar sin romper otros servicios	Se firmó con la clave de la cuenta, así que solo se invalida rotando esa clave	Migra los servicios a identidad, rota las claves, desactiva el acceso con clave compartida y emite SAS de delegación de usuario.
Tras una conmutación regional la cuenta ya no tiene redundancia geográfica	La conmutación deja la cuenta como LRS en la región secundaria	Incluye en el plan de recuperación el paso de reconfigurar la redundancia y el tiempo de la replicación inicial.
Mover datos al nivel frío aumenta el costo en vez de reducirlo	Los cargos por recuperación y la penalización por eliminación temprana superan el ahorro de almacenamiento	Calcula el punto de equilibrio con el volumen releído al mes antes de aplicar la regla.
Un principal sin roles de datos accede a todos los blobs	Tenía un rol de gestión que permite leer las claves de la cuenta	Desactiva allowSharedKeyAccess: la clave sigue existiendo y deja de servir para autenticarse.

■ Seguridad, ética y costo

Trabaja con cuentas propias o sandboxes autorizados. No publiques secretos, identificadores reales ni datos personales. Antes de crear recursos pagos define presupuesto, etiquetas y comando de destrucción; después verifica que no queden recursos huérfanos. Los ejemplos locales enseñan contratos, pero no certifican cumplimiento ni disponibilidad de producción.

■ Preguntas de comprobación

1. ¿Qué configuraciones se deciden en la cuenta y no se pueden matizar por contenedor, y qué implica eso para el diseño?

2. ¿Por qué la redundancia geográfica puede quedar descartada por un requisito de residencia de datos?
3. Con 1 TiB, ¿a partir de qué volumen releído al mes deja de compensar el nivel frío?
4. Tienes versionado y eliminación temporal de blob activos. ¿De qué pérdida NO estás protegido?
5. ¿Qué diferencia operativa concreta hay entre una SAS firmada con la clave de cuenta y una de delegación de usuario?

Referencias

• Microsoft (2025). Azure Storage redundancy — LRS a GZRS, región emparejada y lectura en la secundaria. <https://learn.microsoft.com/en-us/azure/storage/common/storage-redundancy>

• Microsoft (2025). Disaster recovery and storage account failover — conmutación iniciada por el cliente, lastSyncTime y estado posterior. <https://learn.microsoft.com/en-us/azure/storage/common/storage-disaster-recovery-guidance>

• Microsoft (2025). Access tiers for blob data — niveles, permanencia mínima y rehidratación desde archivo. <https://learn.microsoft.com/en-us/azure/storage/blobs/access-tiers-overview>

• Microsoft (2025). Data protection overview — versionado, eliminación temporal, restauración a un punto anterior e inmutabilidad. <https://learn.microsoft.com/en-us/azure/storage/blobs/data-protection-overview>

• Microsoft (2025). Grant limited access with shared access signatures — tipos de SAS y delegación de usuario. <https://learn.microsoft.com/en-us/azure/storage/common/storage-sas-overview>

• Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 03 en PDF · Recorrido de Azure en PDF · Manual integral

Anterior	Índice	Siguiente
← 040 · Virtual Machines, Scale Sets y Load Balancer	Parte 03 · Programa	042 · Azure SQL, Cosmos DB y Azure Cache for Redis →

Evaluación — 041 Blob Storage, Files, redundancia y lifecycle

Preguntas

1. Explica el problema que resuelve esta capacidad sin mencionar un proveedor.
2. Identifica una frontera de responsabilidad y su propietario.
3. Compara dos alternativas mediante confiabilidad, seguridad, costo y operación.
4. ¿Qué demuestra el laboratorio y qué no puede demostrar?
5. Propón un caso límite o una prueba de fallo.

Reto verificable

Entrega storage-gobernado-azure con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

• [ ] Resultado reproducible por una segunda persona.

• [ ] Evidencia vinculada a cada afirmación técnica.

• [ ] Prueba negativa o de fallo incluida.

• [ ] Costos y permisos expresados con unidades y alcance.

• [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.

042 — Azure SQL, Cosmos DB y Azure Cache for Redis

Parte: 03 — Azure: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: data · Estado: EXECUTABLECORE

Propósito

Elegir y operar los motores de datos de Azure con la disciplina de la clase 031 —los patrones de acceso primero— y con tres precisiones que solo aparecen aquí: el modelo de compra decide si vas a poder diagnosticar, la unidad de solicitud de Cosmos DB pone precio a cada consulta en la cabecera de la respuesta, y dos decisiones de esta clase son puertas de un solo sentido que no se corrigen editando una plantilla.

Resultados de aprendizaje

Al finalizar podrás:

1. Diagnosticar una base de datos lenta identificando cuál de los cinco límites se agotó, y no solo la CPU.
2. Elegir modelo de compra y nivel de servicio a partir de la latencia de E/S exigida y de la capacidad de diagnóstico.
3. Diseñar una clave de partición sabiendo que es inmutable y que la partición lógica tiene un tope duro de 20 GB.
4. Calcular cuándo el escalado automático de Cosmos DB cuesta más que aprovisionar el pico.
5. Configurar una caché con política de expulsión y reserva de memoria que no se bloquee al llenarse.

Conceptos centrales

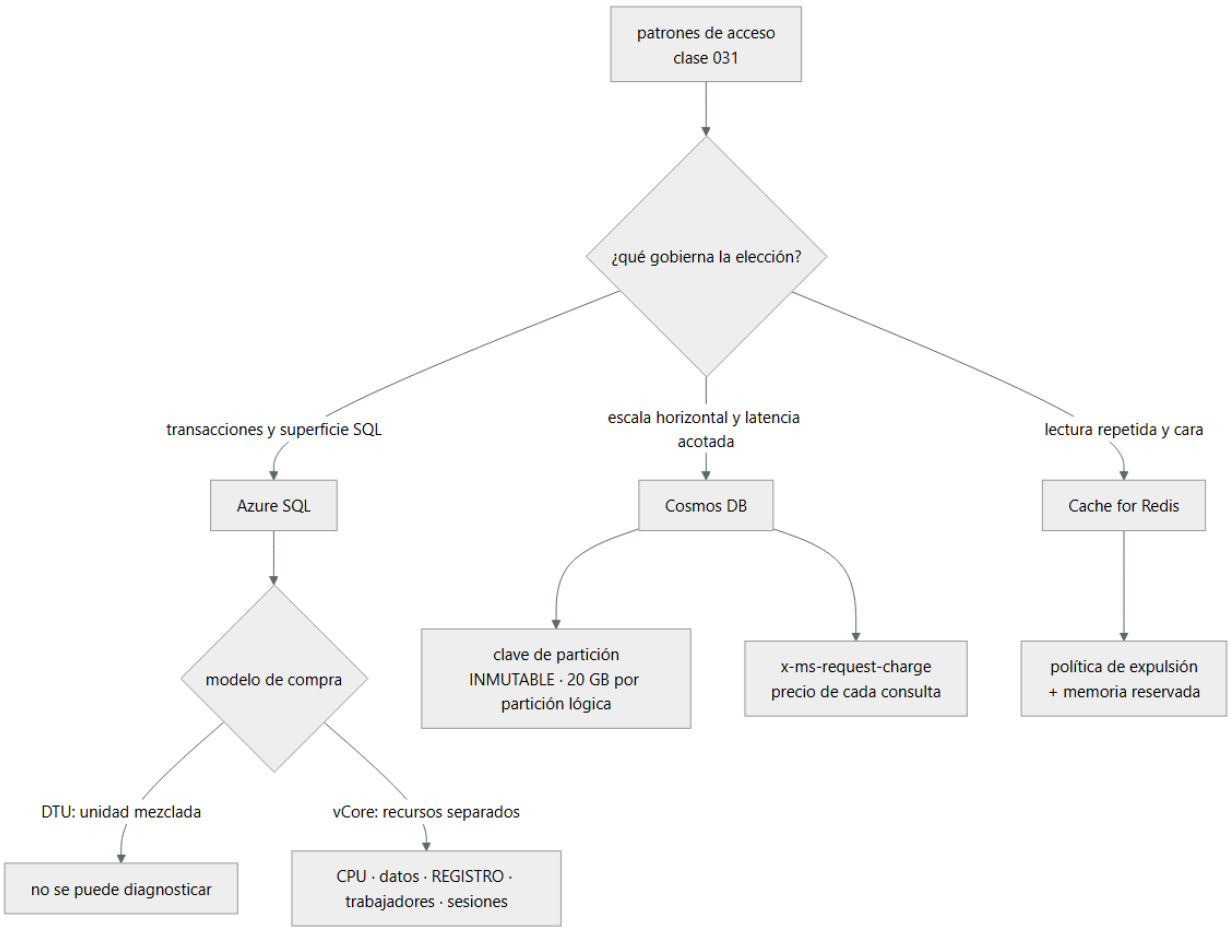
Concepto	Comprensión verificable
DTU frente a vCore	La DTU es una unidad mezclada de CPU, memoria y E/S: cuando se agota, no dice cuál. El modelo vCore expone los recursos por separado y es el único que permite diagnosticar.
gobernador de velocidad de registro	Límite de MB/s de escritura en el registro de transacciones, independiente de la CPU. Es el techo que más se alcanza y el que menos se mira.
unidad de solicitud (RU)	Moneda única de Cosmos DB: toda lectura, escritura y consulta cuesta RU. La cabecera x-ms-request-charge devuelve el precio exacto de cada operación.
partición lógica	Conjunto de documentos con la misma clave de partición. Tiene un tope duro de 20 GB: al alcanzarlo las escrituras fallan, no se ralentizan.
coherencia de sesión	Nivel por defecto de Cosmos DB: garantiza leer lo que tú escribiste mientras se propague el testigo de sesión. Si el cliente se comparte entre usuarios sin propagarlo, la garantía desaparece en silencio.
política de expulsión	Regla que decide qué claves se descartan cuando la caché se llena. volatile-lru —la de por defecto— solo expulsa claves con vencimiento: sin TTL, la caché se llena y deja de aceptar escrituras.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Aplicado a esta clase, separa siempre cuatro planos: intención (qué necesita el usuario), configuración (qué declaramos), estado observado (qué existe de verdad) y evidencia (cómo sabemos que cumple). Confundirlos produce diseños que se ven correctos en un diagrama pero fallan al operar.

■ Flujo de razonamiento



Desarrollo

1. El modelo de compra decide si vas a poder diagnosticar

Azure SQL se compra de dos formas, y la elección no es de precio sino de observabilidad.

La DTU es una unidad mezclada de CPU, memoria y E/S. Cuando una base de datos S3 «está al 100 %», el panel no dice al 100 % de qué. Se sube de nivel, mejora un poco, y a las tres semanas vuelve el mismo síntoma: se compró más de todo para resolver la escasez de uno.

El modelo vCore expone los recursos por separado, y con él la vista que resuelve la mayoría de los casos:

```
SELECT TOP 5 end_time,
       avg_cpu_percent, avg_data_io_percent, avg_log_write_percent,
       max_worker_percent, max_session_percent
FROM sys.dm_db_resource_stats ORDER BY end_time DESC;
```

```
end_time   cpu   data_io   log_write   worker   session
21:14:00   18,2    31,0      100,0       12,4      3,1
```

CPU al 18 % y registro al 100 %. Ese es el patrón que casi nadie busca. El gobernador de velocidad de registro limita los MB/s que pueden confirmarse en el registro de transacciones, y es independiente del resto. Un proceso de importación por lotes lo satura mientras todo lo demás parece ocioso, y el tipo de espera lo confirma:

```
SELECT wait_type, wait_time_ms FROM sys.dm_os_wait_stats
WHERE wait_type LIKE 'LOG_RATE_GOVERNOR' OR wait_type = 'WRITELOG';
```

Las correcciones no pasan por más CPU: reducir el registro generado —cargas mínimamente registradas, lotes más pequeños con menos transacciones abiertas, índices que no se reconstruyan durante la ventana— o elegir un nivel con más caudal de registro.

Las cinco columnas cubren cinco agotamientos distintos:

```
avg_cpu_percent        consultas caras o falta de índices
avg_data_io_percent    lecturas que no caben en memoria
```

```
avg_log_write_percent  escritura por lotes, reconstrucción de índices
max_worker_percent     demasiadas conexiones activas: falta agrupación
max_session_percent    conexiones abiertas sin cerrar
```

Las dos últimas son la firma de un problema de aplicación, no de base de datos, y subir de nivel las empeora porque permite abrir más.

La elección de nivel de servicio es una decisión de latencia, y su precio es conocido:

	Almacenamiento	Latencia de E/S	Réplica de lectura	Costo relativo
Propósito general	Remoto	5-10 ms	No	1×
Crítico para la empresa	SSD local + réplicas	1-2 ms	Sí, incluida	2,7×
Hiperescala	Servidores de páginas	Variable	Hasta 4	Intermedio

Y dos avisos sobre las opciones que parecen atractivas:

Sin servidor pausa la base tras un periodo de inactividad y cobra por vCore-segundo. Reanudarla tarda cerca de un minuto: la primera petición después de la pausa agota su tiempo de espera. Es excelente para desarrollo e intermitencia real, y una fuente de incidentes si se pone delante de un servicio con tráfico esporádico y usuarios reales.

Hiperescala escala a 100 TB y restaura en minutos en vez de horas. Es una puerta de un solo sentido en la práctica: entrar es sencillo y volver no. Se decide con el mismo criterio que cualquier decisión irreversible de la clase 026 — con evidencia, no por si acaso.

Y una elección que no es de tamaño sino de compatibilidad: la instancia administrada existe para lo que la base de datos única no tiene —agente SQL, consultas entre bases de datos, Service Broker, CLR—. Una migración heredada que falla lo hace por ausencia de esas funciones, no por rendimiento; comprobarlo antes ahorra rehacer el plan a medias.

Un último punto sobre las copias de seguridad, que aquí son automáticas y crean una falsa sensación de seguridad: la restauración a un punto anterior cubre de 1 a 35 días y la retención a largo plazo llega a 10 años. Pero las copias cuelgan del servidor lógico: eliminar una base de datos conserva sus copias hasta que expire la retención, y eliminar el servidor se lleva las de todas sus bases a la vez. Es el borrado que no tiene vuelta atrás.

2. Cosmos DB: la cabecera que pone precio a cada consulta

Cosmos DB tiene una virtud pedagógica que ningún otro motor del programa ofrece: devuelve el costo de cada operación en la respuesta.

```
x-ms-request-charge: 2.87        lectura puntual con clave de partición
x-ms-request-charge: 1240.15     la misma consulta sin clave de partición
```

La clase 031 argumentaba que los patrones de acceso van primero. Aquí ese argumento deja de ser una recomendación y se convierte en una factura por línea: una consulta que abanica sobre todas las particiones cuesta 432 veces más que la lectura equivalente dirigida. Instrumentar esa cabecera en el registro de la aplicación es la mejor inversión de esta clase, porque convierte una discusión de diseño en un número que cualquiera puede ver.

La clave de partición concentra las dos decisiones irreversibles:

```
tope duro por partición lógica     20 GB
tope de rendimiento por partición  10.000 RU/s
al alcanzar el tope de tamaño      las escrituras FALLAN con 403
                                   (no se ralentizan: fallan)
cambiar la clave de partición      IMPOSIBLE: hay que crear otro contenedor
                                   y migrar los documentos
```

El fallo con 403 es cualitativamente distinto de la limitación por throughput: no es un 429 que el SDK reintenta, es un rechazo definitivo de la escritura mientras esa partición siga llena. Y como el tope es por partición lógica y no por contenedor, un contenedor de 4 TB puede estar rechazando escrituras porque una sola clave concentra 20 GB.

Los criterios para elegirla, en orden:

```
1. cardinalidad alta          muchos valores distintos, ninguno dominante
2. reparto uniforme           de escrituras y de almacenamiento
3. presente en las consultas  si no, todas abanican
```

Cuando ninguna propiedad natural cumple las tres, la salida es una clave sintética: concatenar dos campos, o añadir un sufijo calculado que reparta. Se decide al crear el contenedor, con los patrones de acceso escritos delante, porque después el arreglo cuesta una migración.

El modelo de rendimiento tiene tres opciones y una aritmética que conviene hacer:

```
aprovisionado manual   se paga el pico declarado, 24 h al día
escalado automático    escala entre el 10 % y el 100 % del máximo,
                       y se factura a 1,5× la tarifa por RU
sin servidor           por operación; tope de 5.000 RU/s y 50 GB por contenedor
```

El factor 1,5 fija el punto de equilibrio: el escalado automático solo compensa por debajo del 66 % de utilización media. Con un contenedor que sostiene 8.200 RU/s de forma estable y un máximo de 10.000:

```
escalado automático   8.200/100 × 0,012 USD/h = 0,984 → ~718 USD/mes
aprovisionado manual 10.000/100 × 0,008 USD/h = 0,800 → ~584 USD/mes
diferencia por pagar elasticidad que no se usa:      ~134 USD/mes
```

Y una lectura contraria a la intuición sobre los errores: en Cosmos DB, un 429 no es un fallo. Es la señal de control de flujo, viene con x-ms-retry-after-ms y el SDK reintenta solo. Una tasa de 429 sostenida en cero significa que se está pagando capacidad ociosa. Lo que sí hay que alertar es la tasa de 429 que el SDK no consigue resolver dentro del presupuesto de reintentos, porque esa sí llega al usuario.

Los cinco niveles de coherencia son el diferenciador del servicio y admiten un resumen operativo corto:

```
fuerte              lectura siempre actualizada; NO disponible con escritura
                    multirregión; lecturas al doble de RU
obsolescencia       retraso acotado por operaciones o por tiempo
  acotada           lecturas al doble de RU
sesión (por defecto) lees lo que tú escribiste, dentro de tu sesión
prefijo coherente   nunca ves escrituras desordenadas
eventual            lo más barato y lo más rápido
```

La trampa está en la de sesión, que es la que casi todo el mundo usa: la garantía viaja en un testigo de sesión. Si el cliente es un objeto compartido entre peticiones de usuarios distintos y el testigo no se propaga, un usuario puede no leer lo que acaba de escribir. No hay error, no hay alerta: solo un comportamiento intermitente que en desarrollo nunca se reproduce porque hay una sola instancia.

Y con escritura multirregión, los conflictos se resuelven por defecto con el último que escribe gana, comparando la marca de tiempo. Es determinista y pierde datos en silencio. Si esa pérdida no es aceptable —dos almacenes actualizando el mismo inventario—, hay que escribir un procedimiento de resolución o renunciar a la escritura multirregión.

3. Redis: la caché que se llena y deja de aceptar escrituras

La clase 031 dejó el criterio de cuándo una caché aporta y cómo se invalida. Lo que se añade aquí son tres decisiones de plataforma que producen incidentes con un patrón reconocible.

El nivel decide si hay SLA. El nivel básico es un nodo único sin acuerdo de nivel de servicio y sin réplica: un reinicio de mantenimiento vacía la caché y corta las conexiones. Es perfectamente válido para desarrollo y es el error de producción más repetido, porque funciona durante meses hasta el primer mantenimiento.

```
básico       nodo único, sin SLA            desarrollo
estándar     réplica principal-secundaria    producción básica
premium      persistencia, clúster, red virtual, zonas
empresa      módulos de Redis y replicación geográfica activa
```

La política de expulsión por defecto no expulsa casi nada. volatile-lru solo considera candidatas las claves con vencimiento. Si parte de las claves se escribe sin TTL —muy habitual: sesiones, catálogos «que se refrescan solos», banderas—, esas claves nunca salen. La caché se llena de lo que no puede expulsar y responde a las escrituras con un error de memoria:

```
síntoma      OOM command not allowed when used memory > 'maxmemory'
señal        used_memory al máximo y evicted_keys = 0
causa        volatile-lru + claves sin TTL
```

La combinación evictedkeys = 0 con memoria al máximo es diagnóstica: una caché sana expulsa. Que no expulse nada y esté llena significa que no puede.

Dos correcciones, y conviene aplicar las dos:

```
1. allkeys-lru  → cualquier clave es candidata; la caché nunca se bloquea
2. TTL siempre  → una clave sin vencimiento en una caché es una fuga de memoria
```

La memoria reservada no es opcional. Redis necesita memoria libre para replicar, para persistir y para la fragmentación. Sin maxmemory-reserved, la operación de guardado intenta duplicar estructuras en una instancia ya llena y el nodo se queda sin memoria durante la copia. La reserva se configura como porcentaje y su ausencia produce caídas justo durante el mantenimiento, que es cuando peor sientan.

Y un detalle de cliente que aparece en casi toda migración desde .NET: el multiplexor de conexión debe ser único y de larga vida. Crear uno por petición agota los puertos de salida de la máquina —el mismo agotamiento de puertos de traducción de la clase 040— y produce tiempos de espera que parecen de la caché y son del cliente:

```
un multiplexor por proceso, compartido y reutilizado          correcto
un multiplexor por petición                                   agota puertos
```

Por último, escalar una caché no es una operación transparente: cambiar de tamaño o de número de particiones reconfigura los nodos, cierra conexiones y, según el camino, puede vaciar el contenido. Toda aplicación que use caché debe funcionar —más lenta— con la caché vacía o inaccesible. Si no puede, no es una caché: es una base de datos sin copia de seguridad.

4. Las dos puertas de un solo sentido de esta clase

La clase 026 introdujo la distinción entre decisiones reversibles e irreversibles. Esta clase contiene dos de las irreversibles más caras del programa, y conviene tratarlas de forma explícita:

```
1. la clave de partición de un contenedor de Cosmos DB
2. la eliminación de un servidor lógico de Azure SQL
```

La clave de partición no se edita. Corregirla implica crear un contenedor nuevo y mover los documentos, y ese movimiento consume RU en los dos contenedores a la vez, así que la migración compite con el tráfico de producción por la misma capacidad. El procedimiento sensato:

```
1. crear el contenedor destino con la nueva clave
2. aprovisionar capacidad extra TEMPORAL en ambos
3. copiar con la fuente de cambios, no con una consulta de barrido
4. doble escritura durante la ventana de corte
5. verificar recuentos por partición antes de conmutar lecturas
6. devolver la capacidad extra
```

El paso 3 importa: leer el origen con una consulta que abanica multiplica el costo y castiga a las particiones ya saturadas. La fuente de cambios lee en orden de partición y es la vía prevista para esto.

El servidor lógico concentra las copias de seguridad de todas sus bases de datos. Eliminarlo las elimina. La protección es doble y ambas mitades hacen falta:

```
# 1. bloqueo de recurso: impide el borrado incluso a un propietario
$ az lock create --name no-borrar-sql --lock-type CanNotDelete \
    --resource-group rg-datos --resource-name sql-cloudshop-prod \
    --resource-type Microsoft.Sql/servers

# 2. retención a largo plazo: copias que NO cuelgan del ciclo de vida del servidor
$ az sql db ltr-policy set -g rg-datos -s sql-cloudshop-prod -n pedidos \
    --weekly-retention P4W --monthly-retention P12M --yearly-retention P7Y --week-of-year 1
```

El bloqueo de recurso es el mecanismo de gobierno de la clase 037 aplicado al caso concreto: no depende de que nadie se equivoque. Y conviene saber su límite: un bloqueo impide borrar, no impide vaciar. Un DROP TABLE pasa igual; para eso está la restauración a un punto anterior, que hay que haber probado.

La prueba, que es la parte que se omite:

```
$ az sql db restore -g rg-datos -s sql-cloudshop-prod -n pedidos \
    --dest-name pedidos-prueba --time "2026-07-31T22:00:00"
$ sqlcmd -S sql-cloudshop-prod.database.windows.net -d pedidos-prueba \
    -Q "SELECT COUNT(*) FROM dbo.pedidos WHERE creado < '2026-07-31 22:00'"
1284391                                                          ✓
$ az sql db delete -g rg-datos -s sql-cloudshop-prod -n pedidos-prueba --yes
```

Una restauración deja una base de datos nueva: no sobrescribe la original. Eso la hace segura de ensayar en producción, y es la razón por la que no hay excusa para no haberla ensayado nunca. El dato que hay que registrar de esa prueba no es que funcionó, sino cuánto tardó: ese número es el RTO real, y suele ser bastante mayor que el que figura en el plan.

5. Elegir entre los tres sin repetir la discusión cada trimestre

Los tres motores resuelven cosas distintas y la conversación se repite porque nadie escribe el criterio. Escrito, cabe en una tabla:

Pregunta	Azure SQL	Cosmos DB	Redis
¿Transacciones entre varias entidades?	Sí	Dentro de una partición lógica	No
¿Consultas no previstas?	Sí	Caras: abanican	No
¿Latencia de un dígito de milisegundo?	Solo crítico para la empresa	Sí	Sí
¿Escala de escritura horizontal?	Limitada	Sí, si la clave reparte	—
¿Fuente de verdad?	Sí	Sí	Nunca

La última fila es la que hay que defender cuando alguien propone «guardar el carrito solo en la caché porque es más rápido»: un dato que solo existe en una caché es un dato que se pierde en el próximo mantenimiento.

Y sobre el costo, la clase 031 avisaba de que a veces decide al revés de lo esperado. Aquí ocurre por una razón concreta: el precio de Cosmos DB depende del diseño, no del volumen. La misma carga puede costar 584 o 4.900 USD al mes según cómo reparta la clave de partición y cuántas consultas abaniquen. Ningún ajuste de infraestructura compensa un modelo de datos que obliga a leer todas las particiones para responder una pregunta frecuente.

La forma de mantener esa disciplina sin auditorías periódicas es hacer visible el número:

```
respuesta = contenedor.query_items(consulta, partition_key=cliente_id)
log.info("consulta=%s ru=%.2f", nombre, contenedor.client_connection.last_response_headers[
    "x-ms-request-charge"])
```

Con ese registro, una consulta cara aparece en el panel el día que se despliega, no en la factura del mes siguiente. Es el mismo principio que la etiqueta de costo de la clase 025 aplicado a la operación individual: lo que no tiene número no se discute, se opina.

Ejemplo trabajado

CloudShop lleva a Azure su capa de datos. El catálogo va a Cosmos DB, los pedidos a Azure SQL y las sesiones a Redis. Los tres funcionan durante seis semanas y luego fallan por motivos distintos — y ninguno es el que el panel sugiere.

Caso 1 — «la base de datos está lenta» con la CPU al 18 %.

La importación nocturna de catálogo pasa de 40 minutos a más de tres horas. El panel de la base de datos S3 —modelo DTU— muestra 100 % de DTU.

```
DTU al 100 %  →  ¿de qué?
```

No se puede saber. Se migra a vCore, propósito general, 4 vCore, y la vista aparece:

```
end_time   cpu   data_io   log_write   worker   session
02:41:00   18,2    24,7      100,0        9,1      2,4
```

El techo es el registro de transacciones. La importación insertaba fila a fila dentro de una transacción por fila:

```
tamaño de lote                      1 fila     5.000 filas
reconstrucción de índices          durante     después
registro generado por 1 M de filas   4,1 GB      0,9 GB
durante la ventana
duración de la importación         3 h 12 min    26 min
```

No hizo falta subir de nivel. La conclusión que se documenta: el modelo DTU no era caro, era ciego.

Caso 2 — el percentil 99 de los pedidos no baja de 40 ms.

Con el registro resuelto, la latencia de lectura sigue alta. La E/S de datos está en el 24 % y la latencia por operación es de 7 ms: es el almacenamiento remoto del nivel de propósito general.

```
latencia de E/S                 ~7 ms               ~1,5 ms
p99 de la operación de pedido    41 ms               9 ms
réplica de lectura               no                  incluida
costo mensual de cómputo        ~365 USD            ~985 USD
```

La decisión no se aplica a todo: solo la base de datos de pedidos sube de nivel. Catálogo e informes se quedan en propósito general, y los informes pasan a leer de la réplica que el nivel crítico incluye, lo que además quita carga de la principal.

```
pedidos      propósito general 365    crítico 985
catálogo     propósito general 365    propósito general 365
informes     propósito general 365    réplica de lectura de pedidos: 0
                              ■■■■■■■■■          ■■■■■■■■■
                               1.095 USD          1.350 USD
```

Cuesta 255 USD más al mes y elimina una base de datos entera. El p99 pasa de 41 a 9 ms.

Caso 3 — las escrituras del catálogo empiezan a fallar con 403.

No es limitación por throughput: es un rechazo.

```
403 · Partition key reaching storage quota
```

La clave de partición era /pais, y el 82 % de los documentos son de un solo país:

```
$ az cosmosdb sql container show -a cosmos-cloudshop -d tienda -n catalogo \
    -g rg-datos --query "resource.partitionKey.paths" -o tsv
/pais
```

```
partición "CL"    19,8 GB  ← contra el tope de 20 GB
partición "AR"     1,4 GB
partición "PE"     0,9 GB
resto             < 0,3 GB
```

La clave es inmutable. Se migra a /categoriaId —cardinalidad alta, reparto uniforme y presente en el 94 % de las consultas— con fuente de cambios y doble escritura:

```
capacidad temporal extra durante la migración   +6.000 RU/s durante 9 h
documentos migrados                             41,2 millones
mayor partición resultante                       1,1 GB
corte de servicio                                ninguno
```

Y con la cabecera instrumentada aparece lo que llevaba seis semanas escondido:

```
lectura de ficha por id (con clave)             2,87 RU     2,87 RU
listado por categoría                       1.240,15 RU     8,44 RU
RU/s medias del contenedor                     8.200        2.100
```

Caso 4 — el modelo de rendimiento que costaba de más.

Con 8.200 RU/s estables antes de la migración, el contenedor estaba en escalado automático con máximo 10.000. La utilización media era del 82 %, muy por encima del punto de equilibrio del 66 %:

```
escalado automático a 8.200 RU/s medias    718
aprovisionado manual a 10.000 RU/s         584
```

Después de la migración, con 2.100 RU/s medias y picos de 6.000, la aritmética se invierte y el escalado automático vuelve a ser lo correcto:

```
escalado automático, máx. 6.000, media 2.100   ~184 USD/mes
aprovisionado manual a 6.000                   ~350 USD/mes
```

El criterio queda escrito para no repetir la discusión: por encima del 66 % de utilización media, manual; por debajo, automático, y se revisa cuando el patrón de carga cambie.

Caso 5 — la caché deja de aceptar escrituras un domingo.

```
OOM command not allowed when used memory > 'maxmemory'
used_memory       6,0 GB / 6,0 GB
evicted_keys      0
```

Memoria llena y ninguna expulsión: la política era volatile-lru y el 61 % de las claves —sesiones y banderas de configuración— se escribía sin vencimiento. Además la instancia era de nivel básico, así que el reinicio de mantenimiento de esa madrugada había vaciado la caché sin réplica que la sostuviera.

```
nivel                            básico            estándar con réplica
política de expulsión          volatile-lru        allkeys-lru
claves sin TTL                    61 %                 0 %
memoria reservada                 0 %                 25 %
multiplexor de conexión        uno por petición     uno por proceso
prueba con caché vacía            nunca             en cada despliegue
```

La última fila es la que convierte esto en un diseño: la aplicación se despliega con una prueba que arranca con la caché vacía y comprueba que responde —más lenta— sin errores.

Resumen de la capa de datos:

```
modelo de compra de Azure SQL                DTU           vCore
cuello de botella identificable              no            sí, cinco señales
p99 de la operación de pedido               41 ms          9 ms
importación nocturna                      3 h 12 min       26 min
escrituras rechazadas en Cosmos DB         sí (403)        0
RU/s medias del catálogo                    8.200          2.100
bases de datos SQL                             3             2
costo mensual de la capa de datos           2.190 USD      1.640 USD
```

La lección que esta clase traslada al resto de la parte: los tres motores tienen un límite que no es la CPU y que el panel por defecto no muestra —el registro en SQL, la partición lógica en Cosmos, la política de expulsión en Redis—. Un servicio de datos que solo se vigila por CPU está vigilando el recurso que casi nunca se agota primero.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-03-azure-core-platform/042-azure-sql-cosmos-db-y-azure-cache-for-redis/lab.py
```

El laboratorio selecciona el motor de práctica data y produce labresult.json. El escenario, sus comprobaciones y el artefacto esperado corresponden a esta clase; no requiere credenciales y deja explícito qué debe revalidarse en un sandbox real.

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa matriz-datos-azure en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es un modelo de datos ligado a patrones de acceso. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye matriz-datos-azure para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

Errores frecuentes

Síntoma	Causa probable	Corrección
La base de datos está al 100 % y no se sabe de qué recurso	El modelo DTU mezcla CPU, memoria y E/S en una sola unidad	Migra a vCore y consulta sys.dmdbresourcestats: cinco columnas, cinco agotamientos distintos.
Un proceso por lotes va lentísimo con la CPU al 18 %	El gobernador de velocidad de registro está al 100 %	Reduce el registro generado —lotes mayores, menos transacciones, índices fuera de la ventana— antes de subir de nivel.
Las escrituras en Cosmos DB fallan con 403 y no con 429	Una partición lógica alcanzó el tope duro de 20 GB	La clave es inmutable: crea otro contenedor con una clave que reparta y migra con la fuente de cambios y doble escritura.
Una consulta frecuente cuesta cientos de RU	Abanica sobre todas las particiones porque la clave no está en el filtro	Registra x-ms-request-charge en cada consulta y rediseña la clave o el modelo para que la consulta sea dirigida.
La caché responde con error de memoria y no ha expulsado ninguna clave	La política volatile-lru solo expulsa claves con vencimiento y muchas se escriben sin TTL	Cambia a allkeys-lru, exige TTL en todas las escrituras y reserva memoria para replicación y fragmentación.
Al eliminar un servidor lógico desaparecen las copias de seguridad de todas sus bases	La restauración a un punto anterior cuelga del servidor, no de la base de datos	Bloqueo de recurso contra el borrado y retención a largo plazo, con una prueba de restauración cuya duración se registra como RTO real.

Seguridad, ética y costo

Trabaja con cuentas propias o sandboxes autorizados. No publiques secretos, identificadores reales ni datos personales. Antes de crear recursos pagos define presupuesto, etiquetas y comando de destrucción; después verifica que no queden recursos huérfanos. Los ejemplos locales enseñan contratos, pero no certifican cumplimiento ni disponibilidad de producción.

Preguntas de comprobación

1. El panel muestra 100 % de DTU. ¿Qué información te falta y cómo la obtienes?
2. ¿Qué distingue un 403 de un 429 en Cosmos DB, y por qué solo uno de los dos lo resuelve el SDK?
3. ¿A partir de qué utilización media deja de compensar el escalado automático, y por qué ese número?
4. Una caché tiene la memoria llena y evictedkeys en cero. ¿Qué está ocurriendo?
5. ¿Qué dos decisiones de esta clase no se pueden deshacer editando una plantilla, y cómo se protege cada una?

Referencias

- Microsoft (2025). vCore purchasing model for Azure SQL Database — recursos separados y niveles de servicio. <https://learn.microsoft.com/en-us/azure/azure-sql/database/service-tiers-sql-database-vcore>
- Microsoft (2025). Resource limits and sys.dmdbresourcestats — CPU, E/S de datos, velocidad de registro, trabajadores y sesiones. <https://learn.microsoft.com/en-us/azure/azure-sql/database/resource-limits-logical-server>
- Microsoft (2025). Partitioning and horizontal scaling in Azure Cosmos DB — partición lógica, tope de 20 GB y claves sintéticas. <https://learn.microsoft.com/en-us/azure/cosmos-db/partitioning-overview>
- Microsoft (2025). Consistency levels in Azure Cosmos DB — los cinco niveles, testigo de sesión y costo en RU. <https://learn.microsoft.com/en-us/azure/cosmos-db/consistency-levels>
- Microsoft (2025). Azure Cache for Redis best practices — niveles, políticas de expulsión, memoria reservada y conexiones. <https://learn.microsoft.com/en-us/azure/azure-cache-for-redis/cache-best-practices-development>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Anterior	Índice	Siguiente
← 041 · Blob Storage, Files, redundancia y lifecycle	Parte 03 · Programa	043 · App Service, Functions y Container Apps →

Evaluación — 042 Azure SQL, Cosmos DB y Azure Cache for Redis

Preguntas

1. Explica el problema que resuelve esta capacidad sin mencionar un proveedor.
2. Identifica una frontera de responsabilidad y su propietario.
3. Compara dos alternativas mediante confiabilidad, seguridad, costo y operación.
4. ¿Qué demuestra el laboratorio y qué no puede demostrar?
5. Propón un caso límite o una prueba de fallo.

Reto verificable

Entrega matriz-datos-azure con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

- [ ] Resultado reproducible por una segunda persona.
- [ ] Evidencia vinculada a cada afirmación técnica.
- [ ] Prueba negativa o de fallo incluida.
- [ ] Costos y permisos expresados con unidades y alcance.
- [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.

043 — App Service, Functions y Container Apps

Parte: 03 — Azure: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: serverless · Estado: EXECUTABLECORE

Propósito

Elegir dónde corre la aplicación en Azure sabiendo cuál es la unidad que escala y cuál es la que factura, porque en las tres opciones son distintas y ninguna coincide con la de AWS. El plan de App Service escala para todas sus aplicaciones a la vez, la aplicación de funciones escala para todas sus funciones a la vez, y una aplicación de contenedor escala por la regla que le pongas —y si esa regla es la equivocada, se apaga y deja de trabajar sin dar ningún error.

Resultados de aprendizaje

Al finalizar podrás:

1. Identificar la unidad de escalado y la unidad de facturación de App Service, Functions y Container Apps, y el radio de impacto de cada una.
2. Elegir plan de hospedaje de funciones a partir de duración máxima, acceso a red privada y tolerancia al arranque en frío.
3. Diagnosticar el agotamiento de puertos de traducción de salida, que se manifiesta como fallo del destino y no del origen.
4. Configurar un intercambio de espacios con calentamiento y ajustes fijados al espacio, y explicar qué se lleva el intercambio.
5. Definir reglas de escalado que no apaguen a un consumidor que sigue teniendo trabajo pendiente.

Conceptos centrales

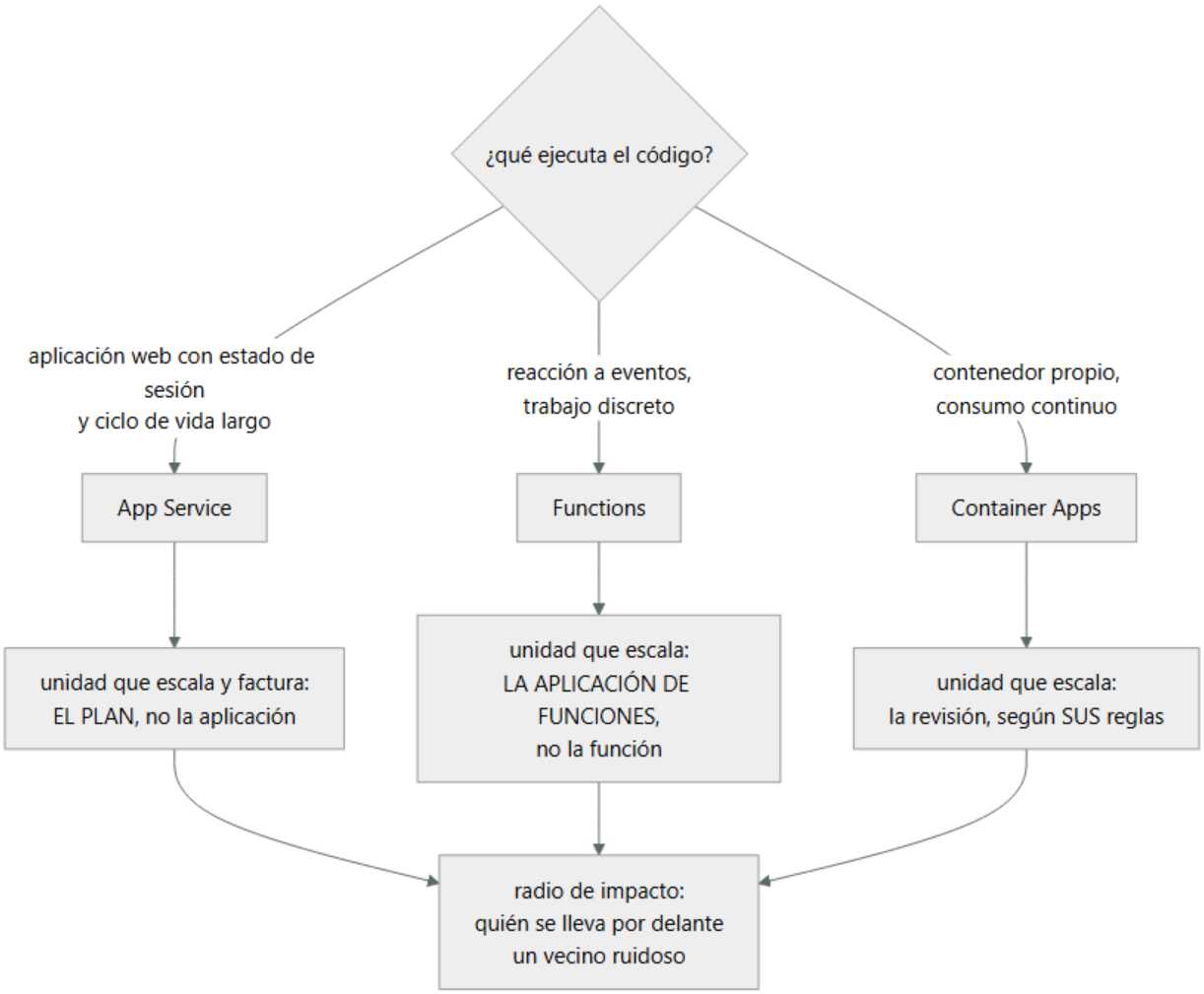
Concepto	Comprensión verificable
plan de App Service	Conjunto de máquinas que ejecuta todas las aplicaciones asignadas a él. Se factura el plan, no la aplicación, y el escalado es del plan: diez aplicaciones compiten por la misma CPU.
Always On	Ajuste que impide que la aplicación se descargue tras unos veinte minutos sin tráfico. Sin él, la primera petición tras el silencio paga un arranque completo.
ajuste fijado al espacio	Configuración marcada para quedarse en el espacio durante un intercambio. Lo que no está marcado viaja con la aplicación: una cadena de conexión de pruebas puede acabar en producción.
puertos de traducción de salida	Cupo de conexiones salientes por instancia —128 por defecto en App Service—. Al agotarse, las llamadas a servicios externos fallan de forma intermitente y el error parece del destino.
plan de hospedaje de funciones	Decide tres cosas a la vez: duración máxima de una ejecución, si hay acceso a red virtual y si existe arranque en frío. No es una decisión de precio.
regla de escalado	Señal que determina cuántas réplicas hay. Una aplicación de contenedor con una única regla HTTP se reduce a cero aunque tenga una cola llena esperando.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Aplicado a esta clase, separa siempre cuatro planos: intención (qué necesita el usuario), configuración (qué declaramos), estado observado (qué existe de verdad) y evidencia (cómo sabemos que cumple). Confundirlos produce diseños que se ven correctos en un diagrama pero fallan al operar.

■ Flujo de razonamiento



Desarrollo

1. El plan es la unidad, no la aplicación

Es la primera sorpresa al llegar de AWS, donde cada servicio escala por su cuenta. En App Service se compra un plan —un conjunto de máquinas— y las aplicaciones se asignan a él:

```
plan asp-cloudshop-prod  (2 × P1v3)
  ■■ tienda            tráfico de clientes
  ■■ admin             uso interno
  ■■ informes          trabajo nocturno pesado
  ■■ webhooks          picos impredecibles
```

Las cuatro comparten CPU, memoria y conexiones salientes. El trabajo nocturno de informes degrada la tienda, y el panel de la tienda no muestra nada raro: su código no cambió y su CPU no subió, subió la del vecino.

La regla de diseño se deduce sola: el plan es la frontera del radio de impacto. Se reparten aplicaciones por plan según qué debe seguir en pie cuando otra cosa se rompa, no según qué es más barato de agrupar. Y como el escalado también es del plan, escalar por culpa de informes multiplica el costo de las cuatro.

Dos ajustes del plan que producen incidentes por sí solos:

Always On. Sin él, la aplicación se descarga tras unos veinte minutos sin tráfico y la siguiente petición paga el arranque completo del proceso. En un servicio interno que se usa dos veces al día, todo el mundo lo percibe como «esto va lentísimo» y las métricas medias no lo muestran, porque afecta a una petición de cada cien. Tampoco los trabajos programados dentro del proceso sobreviven a la descarga: se planifican y no se ejecutan.

Escalado del plan. El escalado automático es de Azure Monitor, con las mismas reglas de la clase 029, y con una diferencia de tiempos: añadir una instancia a un plan tarda entre uno y tres minutos, no segundos. Una regla que reaccione a un pico de 60 segundos llega siempre tarde. Para carga predecible —los picos de una tienda lo son— un perfil por horario evita escalar por reacción.

Y el agotamiento de conexiones salientes merece su propio párrafo porque el error apunta al sitio equivocado. Cada instancia dispone de unos 128 puertos de traducción hacia un mismo destino. Un cliente HTTP que se crea y se descarta en cada petición no reutiliza conexiones, y en cuanto el tráfico sube:

```
síntoma    tiempos de espera intermitentes hacia la pasarela de pago
           el resto de la aplicación funciona
lectura    parece que la pasarela de pago falla
señal real métrica "SNAT Connection Count" al máximo
           y "SNAT Port Usage" en ascenso
```

Las tres correcciones, en orden de eficacia:

```
1. reutilizar el cliente HTTP (una instancia por proceso, no por petición)
2. integración con red virtual + NAT Gateway  → 64.512 puertos por IP (clase 039)
3. punto de conexión privado para los destinos que sean de Azure
   → no consumen puertos de traducción en absoluto
```

La tercera es la que más rinde en una plataforma que ya hizo el trabajo de la clase 039: el tráfico a almacenamiento, bases de datos y colas deja de salir, deja de contar y deja de fallar.

Un apunte sobre la red que se confunde en casi todos los equipos: la integración con red virtual es de salida. Permite que la aplicación alcance recursos privados; no hace que la aplicación sea privada. Para eso hace falta un punto de conexión privado hacia la propia aplicación, que es la otra dirección y otro recurso.

2. Espacios de implementación: qué viaja y qué se queda

Un espacio es una instancia paralela de la aplicación con su propio nombre de host. El intercambio no vuelve a desplegar nada: cambia de sitio la configuración y redirige el tráfico, lo que hace que la vuelta atrás sea igual de rápida que la ida. Esa es toda su virtud, y es grande: un despliegue que se puede revertir en quince segundos cambia el apetito de riesgo de un equipo.

La parte que hay que dominar es qué se lleva el intercambio:

```
viaja con la aplicación   código, ajustes NO marcados, cadenas de conexión NO marcadas,
                          configuración de tiempo de ejecución
se queda en el espacio    ajustes marcados como fijados al espacio,
                          nombre de host, certificados, escalado,
                          configuración de red privada
```

La primera línea contiene el incidente clásico: si la cadena de conexión a la base de datos de pruebas no está marcada como fijada al espacio, el intercambio la lleva a producción. La aplicación arranca sin errores y empieza a escribir pedidos reales en la base de datos de pruebas. No hay excepción, no hay alerta; solo datos en el sitio equivocado.

```
$ az webapp config appsettings set -g rg-app -n app-tienda --slot staging \
    --slot-settings DB_CONNECTION=... APPINSIGHTS_KEY=...
$ az webapp config appsettings list -g rg-app -n app-tienda --slot staging \
    --query "[?slotSetting].name" -o tsv
DB_CONNECTION
APPINSIGHTS_KEY                                                    ✓
```

La comprobación de esa lista debería formar parte de la revisión de cada ajuste nuevo, porque el valor por defecto es el peligroso: lo que no se marca, viaja.

La segunda pieza es el calentamiento. El intercambio reinicia la aplicación en el espacio de destino, así que sin calentamiento las primeras peticiones tras el intercambio pagan el arranque completo. Azure espera a que la aplicación responda antes de completar el intercambio si se le indica qué pedir:

```
<applicationInitialization>
  <add initializationPage="/readyz" />
  <add initializationPage="/api/catalogo?limit=1" />
</applicationInitialization>
```

Y vale aquí, entera, la lección de la clase 029: la ruta de calentamiento debe ejercitar las dependencias, no devolver 200 desde memoria. Un /readyz que comprueba base de datos y caché convierte el intercambio en una prueba de humo automática; un / estático lo convierte en un ritual.

El patrón completo que conviene dejar escrito:

```
1. desplegar en el espacio de preparación
2. ejecutar pruebas contra el nombre de host del espacio
3. intercambiar con calentamiento
4. observar cinco minutos las señales de error y latencia
5. si algo se degrada, intercambiar de vuelta — no desplegar la versión anterior
```

El paso 5 es la razón de todo lo demás: la reversión es un intercambio, no un despliegue. Un equipo que responde a un despliegue malo desplegando otra vez tarda entre diez y cuarenta minutos; uno que intercambia tarda menos de un minuto.

3. Functions: el plan decide tres cosas y ninguna es el precio

La clase 032 estableció que en serverless la concurrencia es el recurso que se agota y que el arranque en frío se compara con el percentil del SLO, no con la media. Ese método se traslada tal cual. Lo que cambia en Azure es que el plan de hospedaje decide simultáneamente tres cosas que en Lambda son ajustes independientes:

	Duración máxima	Red virtual	Arranque en frío
Consumo	5 min por defecto, 10 como tope	No	Sí
Consumo flexible	30 min por defecto	Sí	Se mitiga con instancias siempre listas
Premium	Sin tope práctico (60 min por defecto)	Sí	No: instancias precalentadas
Dedicado (plan de App Service)	Sin tope	Sí	No, con Always On

El tope de diez minutos del plan de consumo es duro y no se negocia con una solicitud de soporte. Un proceso de facturación mensual que tarda catorce minutos no se arregla subiendo la memoria: se arregla partiéndolo o llevándolo a Durable Functions, que es el mecanismo de orquestación de la clase 032 con estado gestionado por la plataforma.

Y aquí está la diferencia estructural con AWS, la que produce el incidente que nadie anticipa:

```
AWS Lambda    escala POR FUNCIÓN
Azure         escala POR APLICACIÓN DE FUNCIONES
```

Todas las funciones de una misma aplicación comparten el plan, el proceso anfitrión y la decisión de escalado. Consecuencia directa: una avalancha de mensajes en una cola hace que el controlador escale la aplicación entera, y la función HTTP que atiende un webhook de pago compite por el mismo anfitrión con el procesador de la cola. Su percentil 99 se dispara sin que su tráfico haya cambiado.

La regla que evita la clase entera de problemas: una aplicación de funciones por perfil de carga, no por dominio funcional. Lo que responde a un usuario y lo que procesa en segundo plano no viven juntos, aunque compartan repositorio y despliegue.

Sobre el arranque en frío, los tres factores que más pesan y se pueden controlar:

```
tamaño del paquete       menos dependencias, arranque más corto
lenguaje                 los tiempos de ejecución interpretados arrancan antes
                         que los que compilan al vuelo
instancias siempre listas convierten el arranque en frío en un costo fijo
```

La tercera es un intercambio explícito: se paga por tener instancias vivas para no pagar latencia. Cuánto vale eso lo decide el SLO, con el método de la clase 032: si el percentil 99 del objetivo es de 500 ms y un arranque en frío cuesta 2,3 s, basta con que ocurra en el 1,2 % de las peticiones para incumplirlo. Ese cálculo es el que justifica el gasto, no la incomodidad de verlo en una traza.

4. Container Apps: la regla que apaga al que tenía trabajo

Container Apps ejecuta contenedores sobre Kubernetes con KEDA y Dapr debajo, sin exponer ninguno de los tres. La parte 06 entra en Kubernetes; aquí importa lo que se decide sin verlo.

La regla de escalado es la pieza central, y su valor por defecto contiene una trampa cara. Una aplicación con regla HTTP se reduce a cero cuando no hay peticiones. Eso es correcto para un servicio web y es destructivo para un consumidor de cola:

```
consumidor con regla HTTP únicamente
  no recibe peticiones HTTP nunca  →  0 réplicas
```

```
la cola crece                    →  sigue en 0 réplicas
nadie recibe un error            →  nadie se entera
```

El síntoma no es un fallo: es trabajo que no se hace. Se descubre cuando alguien pregunta por qué no llegan los correos de confirmación, con cuarenta mil mensajes acumulados. La corrección es una regla que mire la señal correcta:

```
scale:
  minReplicas: 1          # un consumidor no baja de uno
  maxReplicas: 20
  rules:
    - name: cola-pedidos
      custom:
        type: azure-servicebus
        metadata:
          queueName: pedidos
          messageCount: "20"     # una réplica por cada 20 mensajes pendientes
```

minReplicas: 1 en un consumidor no es desperdicio: es la diferencia entre un retraso de segundos y un retraso que dura hasta que alguien lo note. Reducir a cero está bien donde el disparador es la propia llegada de la petición.

Las revisiones son la otra pieza: cada despliegue crea una revisión inmutable y el tráfico se reparte entre ellas por porcentaje.

```
$ az containerapp ingress traffic set -g rg-app -n ca-pedidos \
    --revision-weight pedidos--v7=90 pedidos--v8=10
```

Es un despliegue canario sin infraestructura adicional, y su valor depende por completo de tener una señal por revisión. Repartir 90/10 sin poder comparar la tasa de error de cada una es repartir a ciegas; la clase 045 monta esa parte.

Los perfiles de carga de trabajo deciden dónde corre cada contenedor:

```
consumo    se paga por vCPU-segundo y GiB-segundo, escala a cero
dedicado   máquinas reservadas, precio por hora, aislamiento y tamaños grandes
```

El de consumo es el correcto por defecto; el dedicado aparece cuando hace falta más memoria de la que el de consumo ofrece, aislamiento de cómputo o costo predecible con carga estable — el mismo razonamiento de los modelos de compra de la clase 028.

Y la tabla que cierra la elección de la clase, que es el entregable real:

Pregunta	App Service	Functions	Container Apps
¿Contenedor propio con dependencias del sistema?	Limitado	No	Sí
¿Reducir a cero?	No	Sí	Sí
¿Ejecución de más de 30 min?	Sí	Solo en Premium o dedicado	Sí
¿Despliegue canario integrado?	Espacios	Espacios	Revisiones con peso
¿Escala por longitud de cola?	Con métrica personalizada	Sí, nativo	Sí, nativo
Unidad de escalado	El plan	La aplicación de funciones	La revisión

La última fila es la que hay que llevarse. Las tres opciones escalan; lo que las distingue es qué se arrastra cuando escalan.

5. Salir a la red privada desde una plataforma administrada

Las tres opciones corren fuera de tu red virtual por defecto. Como la clase 039 terminó cerrando el acceso público de la base de datos y del almacenamiento, este es el punto donde ambas decisiones se encuentran — y donde una plataforma bien diseñada deja de funcionar por hacerlo bien.

El mapa de lo que hace falta en cada caso:

```
App Service        integración regional con red virtual (salida)
                   + punto de conexión privado hacia la app (entrada privada)
Functions          plan de consumo: NO hay acceso a red virtual
                   consumo flexible, premium o dedicado: sí
Container Apps     el entorno se despliega EN una subred delegada
                   → la decisión se toma al crear el entorno, no después
```

La segunda línea es la que rompe migraciones: un conjunto de funciones en plan de consumo no puede alcanzar una base de datos con acceso público deshabilitado. No es un problema de reglas de red que se arregle abriendo un puerto —no hay ruta. La corrección es cambiar de plan, y cambiar de plan cambia el precio, así que el requisito de red debe entrar en la elección desde el principio y no aparecer al final.

La tercera es una decisión de un solo sentido más: la subred del entorno de Container Apps se fija al crearlo y no se cambia. Junto con el tamaño —el entorno exige un bloque de direcciones considerable— es exactamente el tipo de reserva que la planificación de la clase 039 debía haber previsto.

Y hay un detalle de DNS que reproduce el fallo silencioso de aquella clase. Con integración de red virtual, la aplicación resuelve nombres usando el DNS de la red virtual solo si se le indica:

```
$ az webapp config appsettings set -g rg-app -n app-tienda --settings \
    WEBSITE_VNET_ROUTE_ALL=1 WEBSITE_DNS_SERVER=168.63.129.16
```

Sin el primero, únicamente el tráfico hacia rangos privados entra por la red virtual y el resto sale por internet. Sin el segundo, la zona DNS privada no se consulta y el nombre del almacenamiento vuelve a resolver a su IP pública. Es el mismo síntoma de la clase 039 con otro origen: todo funciona y nada es privado, hasta que se cierra el acceso público y entonces deja de funcionar de golpe.

La comprobación honesta es siempre la misma y cuesta un minuto:

```
$ az webapp ssh -g rg-app -n app-tienda --command \
    "nslookup stcloudshopprod.blob.core.windows.net"
Address: 10.20.6.5                                                  ✓
```

Ejemplo trabajado

CloudShop lleva a Azure su capa de aplicación: tienda y administración en App Service, procesamiento de pedidos en Functions y el consumidor de notificaciones en Container Apps. Cinco semanas después hay cinco incidentes, y cuatro de ellos apuntan al componente equivocado.

Punto de partida:

```
plan asp-cloudshop-prod (2 × P1v3, ~248 USD/mes)
  tienda · admin · informes · webhooks
func-cloudshop (plan de consumo)
  procesar-pedido (cola) · webhook-pago (HTTP) · facturar-mes (temporizador)
ca-notificaciones (Container Apps, regla HTTP, mín. 0)
```

Incidente 1 — la tienda se degrada cada noche a las 02:00.

El percentil 99 de la tienda pasa de 180 ms a 1,9 s durante veinte minutos. Su CPU no sube.

```
$ az monitor metrics list --resource $PLAN_ID --metric CpuPercentage \
    --aggregation Maximum --interval PT5M --query "value[0].timeseries[0].data[-6:]"
[31.0, 94.2, 97.8, 96.1, 92.4, 28.7]
```

La CPU del plan al 97 %: es informes, que genera el cierre diario. Las cuatro aplicaciones comparten las dos instancias. Se separa por radio de impacto:

```
asp-prod   tienda, admin,     tienda, webhooks   2 × P1v3   248 USD
           informes, webhooks
asp-interno    —              admin, informes    1 × P1v3   124 USD
                              ■■■■■■■■■■■          ■■■■■■■■■■■■■■■■■■■■
                              248 USD                 372 USD
```

Cuesta 124 USD más al mes. El p99 nocturno de la tienda vuelve a 180 ms y el cierre diario deja de tener que competir por CPU: baja de 20 a 11 minutos.

Incidente 2 — la pasarela de pago «falla» en los picos.

Tiempos de espera intermitentes hacia el proveedor de pago, siempre por encima de 400 peticiones por minuto. El proveedor confirma que no ve las solicitudes.

```
$ az monitor metrics list --resource $APP_ID --metric SnatConnectionCount \
    --aggregation Total --interval PT1M --query "value[0].timeseries[0].data[-3:]"
[128, 128, 128]
```

El cupo de puertos, clavado en el techo. El código creaba un cliente HTTP por petición.

```
cliente HTTP                      uno por petición  uno por proceso
```

```
puertos en uso en el pico              128 (tope)         19
integración con red virtual              no           sí + NAT Gateway
fallos hacia la pasarela               3,1 %            0,0 %
```

Y se aprovecha para que el tráfico hacia almacenamiento y base de datos deje de contar: con puntos de conexión privados, esas llamadas no consumen ningún puerto de traducción.

Incidente 3 — un intercambio de espacios escribe pedidos en la base de datos de pruebas.

Durante cincuenta minutos, 214 pedidos acaban en el entorno equivocado. La aplicación no dio un solo error.

```
$ az webapp config appsettings list -g rg-app -n app-tienda --slot staging \
    --query "[?slotSetting].name" -o tsv
APPINSIGHTS_KEY
```

DBCONNECTION no estaba marcada, así que viajó con la aplicación. Se corrige y se añade el calentamiento que faltaba:

```
ajustes fijados al espacio                1                6
ruta de calentamiento                  ninguna          /readyz con dependencias
primeras peticiones tras intercambiar  2,4 s p99          190 ms p99
reversión ensayada                       no           sí, 14 s medidos
```

Los 214 pedidos se recuperan de la base de pruebas y se reinsertan. La medida que evita la repetición no es la corrección: es la comprobación de la lista de ajustes fijados en cada revisión de configuración.

Incidente 4 — el webhook de pago se degrada cuando llega una avalancha de pedidos.

Y el cierre mensual, además, se corta a los diez minutos:

```
FunctionTimeoutException: Timeout value of 00:10:00 exceeded
```

Dos problemas con la misma raíz: tres funciones muy distintas en una sola aplicación con plan de consumo. Se separan por perfil de carga y por requisito:

```
webhook-pago     consumo, compartido        consumo flexible, app propia,
                                            2 instancias siempre listas
procesar-pedido  consumo, compartido        consumo flexible, app propia
facturar-mes     consumo, corta a 10 min    Durable Functions, en la app de proceso
```

```
p99 del webhook durante avalancha    4,8 s      210 ms
cierre mensual                     no termina   38 min, en 6 etapas
acceso a la base de datos privada   imposible   sí (consumo flexible)
costo mensual de funciones         ~28 USD     ~96 USD
```

Los 68 USD adicionales compran tres cosas que el plan de consumo no podía dar: aislamiento entre cargas, acceso a la red privada y un proceso largo que termina.

Incidente 5 — 41.000 notificaciones sin enviar.

Nadie recibe los correos de confirmación desde hace dos días. La aplicación de contenedor está sana y en cero réplicas.

```
$ az containerapp show -g rg-app -n ca-notificaciones \
    --query "properties.template.scale" -o json
{"minReplicas": 0, "maxReplicas": 10, "rules": [{"name": "http", "http": {"metadata": {
"concurrentRequests": "50"}}}]}
```

La única regla miraba peticiones HTTP, y el consumidor no recibe ninguna: lee de una cola. Con cero peticiones, cero réplicas, y nadie vacía la cola. No hubo ningún error que alertar.

```
regla de escalado              HTTP              longitud de cola
réplicas mínimas                 0                    1
mensajes pendientes            41.000               < 30
alerta sobre profundidad de cola  no        sí, > 500 durante 5 min
```

La alerta sobre la profundidad de la cola es la corrección de fondo: el consumidor puede volver a fallar por otro motivo, y lo que hay que detectar es trabajo que se acumula, no procesos que se caen.

Resumen de la capa de aplicación:

```
planes de App Service                       1            2
aplicaciones de funciones                   1            2
fallos hacia la pasarela de pago          3,1 %         0,0 %
p99 nocturno de la tienda                 1,9 s         180 ms
p99 del webhook en avalancha              4,8 s         210 ms
```

```
cierre mensual                              no termina      38 min
trabajo en segundo plano detenido         2 días sin señal  alerta a 5 min
costo mensual de cómputo de aplicación      276 USD         468 USD
```

La lección que esta clase traslada al resto de la parte: en las tres plataformas administradas, lo que hay que preguntar antes de desplegar no es cuánto escala, sino qué más se lleva consigo cuando escala — y qué señal existe cuando decide no escalar en absoluto. Un componente en cero réplicas con trabajo pendiente es el fallo más silencioso de esta parte, porque desde fuera es indistinguible de un sistema en reposo.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-03-azure-core-platform/043-app-service-functions-y-container-apps/lab.py
```

El laboratorio selecciona el motor de práctica serverless y produce labresult.json. El escenario, sus comprobaciones y el artefacto esperado corresponden a esta clase; no requiere credenciales y deja explícito qué debe revalidarse en un sandbox real.

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa api-plataforma-azure en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es una función con límites, reintentos e idempotencia. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye api-plataforma-azure para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

Errores frecuentes

Síntoma	Causa probable	Corrección
Una aplicación se degrada sin que su tráfico ni su CPU hayan cambiado	Comparte plan de App Service con otra que consume la CPU del plan	Reparte las aplicaciones por radio de impacto: el plan es la frontera, no una agrupación de conveniencia.
Fallos intermitentes hacia un servicio externo que confirma no recibir las solicitudes	Agotamiento de puertos de traducción de salida por crear un cliente HTTP en cada petición	Reutiliza el cliente, integra con red virtual y NAT Gateway, y usa puntos de conexión privados para los destinos de Azure.
Tras un intercambio de espacios, la aplicación escribe en la base de datos equivocada	La cadena de conexión no estaba marcada como fijada al espacio, así que viajó con la aplicación	Marca como fijado al espacio todo lo que identifique al entorno y revisa la lista en cada cambio de configuración.
Una función HTTP se degrada cuando llega una avalancha a una cola distinta	En Azure el escalado es por aplicación de funciones, no por función	Una aplicación de funciones por perfil de carga: lo que responde a un usuario no convive con lo que procesa en segundo plano.
Un proceso largo se corta a los diez minutos y subir la memoria no lo arregla	El plan de consumo tiene un tope duro de diez minutos por ejecución	Parte el trabajo con Durable Functions o usa consumo flexible, premium o dedicado.
Una cola crece durante días sin que nadie reciba un error	El consumidor tiene una regla de escalado HTTP y se redujo a cero réplicas	Escala por longitud de cola, fija un mínimo de una réplica y alerta sobre la profundidad de la cola, no sobre la salud del proceso.

Seguridad, ética y costo

Trabaja con cuentas propias o sandboxes autorizados. No publiques secretos, identificadores reales ni datos personales. Antes de crear recursos pagos define presupuesto, etiquetas y comando de destrucción; después verifica que no queden recursos huérfanos. Los ejemplos locales enseñan contratos, pero no certifican cumplimiento ni disponibilidad de producción.

Preguntas de comprobación

1. ¿Cuál es la unidad de escalado en App Service, Functions y Container Apps, y qué se arrastra en cada caso?
2. Un servicio externo falla de forma intermitente solo en los picos. ¿Qué métrica confirma que el problema es tuyo?
3. ¿Qué viaja y qué se queda en un intercambio de espacios, y cuál es el valor por defecto peligroso?
4. ¿Por qué un proceso de catorce minutos no se arregla con más memoria en el plan de consumo?
5. Un consumidor de cola está en cero réplicas y no hay ningún error. ¿Qué señal debería haber existido?

Referencias

- Microsoft (2025). App Service plan overview — la unidad de escalado y facturación, y el efecto de compartirla. <https://learn.microsoft.com/en-us/azure/app-service/overview-hosting-plans>
- Microsoft (2025). Set up staging environments in App Service — intercambio, ajustes fijados al espacio y calentamiento. <https://learn.microsoft.com/en-us/azure/app-service/deploy-staging-slots>
- Microsoft (2025). Azure Functions hosting options — duración máxima, red virtual y arranque en frío por plan. <https://learn.microsoft.com/en-us/azure/azure-functions/functions-scale>
- Microsoft (2025). Set scaling rules in Azure Container Apps — reglas KEDA, réplicas mínimas y escaladores de cola. <https://learn.microsoft.com/en-us/azure/container-apps/scale-app>
- Microsoft (2025). Troubleshoot SNAT port exhaustion — cupo por instancia, métricas y mitigaciones. <https://learn.microsoft.com/en-us/azure/app-service/troubleshoot-intermittent-outbound-connection-errors>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 03 en PDF · Recorrido de Azure en PDF · Manual integral

Evaluación — 043 App Service, Functions y Container Apps

Preguntas

1. Explica el problema que resuelve esta capacidad sin mencionar un proveedor.
2. Identifica una frontera de responsabilidad y su propietario.
3. Compara dos alternativas mediante confiabilidad, seguridad, costo y operación.
4. ¿Qué demuestra el laboratorio y qué no puede demostrar?
5. Propón un caso límite o una prueba de fallo.

Reto verificable

Entrega api-plataforma-azure con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

- [ ] Resultado reproducible por una segunda persona.
- [ ] Evidencia vinculada a cada afirmación técnica.
- [ ] Prueba negativa o de fallo incluida.
- [ ] Costos y permisos expresados con unidades y alcance.
- [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.

044 — Service Bus, Event Grid y Event Hubs

Parte: 03 — Azure: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: messaging · Estado: EXECUTABLECORE

Propósito

Repartir el trabajo asíncrono entre los tres servicios de mensajería de Azure a partir de una sola pregunta que casi nunca se hace: ¿el mensaje se consume o se observa? Confundirla lleva a poner órdenes de pedido en un registro de eventos que nunca las borra, sin cola de mensajes fallidos y con un evento envenenado bloqueando una partición entera. La clase 033 dejó el criterio de la entrega y la idempotencia; aquí cambian las piezas y aparecen tres relojes —el bloqueo, la ventana de duplicados y el reintento— que hay que dimensionar juntos.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir entre Service Bus, Event Grid y Event Hubs según si el mensaje se consume, se enruta o se observa.
2. Dimensionar la duración del bloqueo frente al tiempo real de proceso y explicar qué ocurre cuando el primero es menor.
3. Configurar el traslado automático a mensajes fallidos y una alerta sobre él, en los tres servicios.
4. Distinguir la detección de duplicados —con ventana acotada— de la idempotencia del manejador.
5. Decidir el número de particiones sabiendo qué se rompe al cambiarlo después.

Conceptos centrales

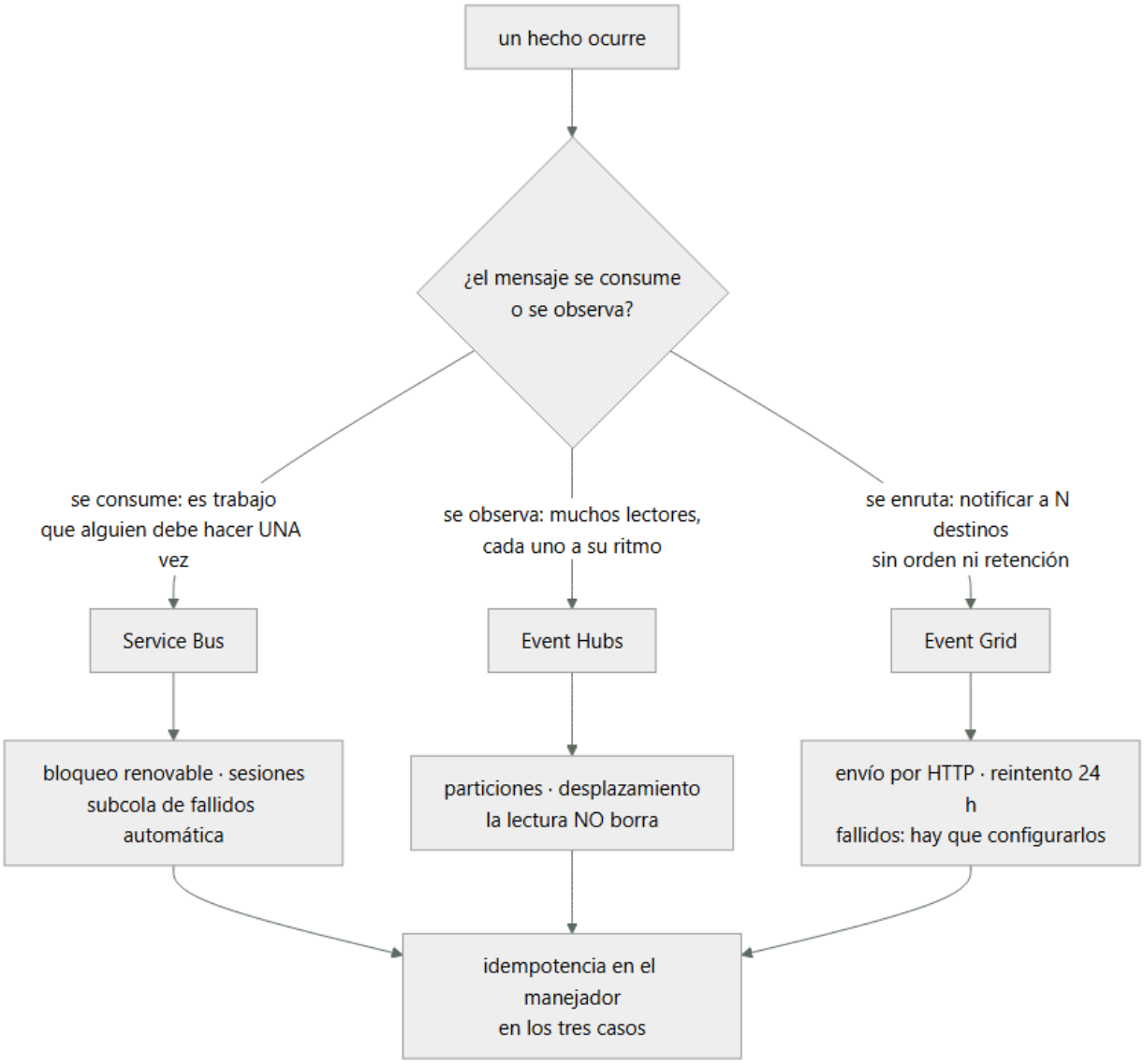
Concepto	Comprensión verificable
bloqueo por inspección	Modo de recepción en el que el mensaje queda reservado un tiempo en vez de borrarse. Equivale al tiempo de invisibilidad de la clase 033 y añade algo que aquel no tiene: se puede renovar.
subcola de mensajes fallidos	Destino automático de un mensaje que agota su cuenta de entregas. En Service Bus existe siempre, cuelga de la propia entidad y registra el motivo en DeadLetterReason.
sesión	Agrupación que garantiza orden dentro de un identificador. Una sesión la bloquea un único consumidor, así que una sesión con mucho tráfico serializa todo su trabajo.
detección de duplicados	Descarte de mensajes con el mismo identificador dentro de una ventana temporal acotada. Fuera de la ventana no descarta nada: no sustituye a la idempotencia.
grupo de consumidores y punto de control	En Event Hubs, la lectura no borra: cada grupo avanza su propio marcador. Si el punto de control se guarda poco, un reinicio reprocesa desde el último guardado.
unidad de rendimiento	Unidad de capacidad de Event Hubs: 1 MB/s de entrada y 2 MB/s de salida. El inflado automático sube y no baja: lo que escala una prueba de carga se paga todo el mes.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Aplicado a esta clase, separa siempre cuatro planos: intención (qué necesita el usuario), configuración (qué declaramos), estado observado (qué existe de verdad) y evidencia (cómo sabemos que cumple). Confundirlos produce diseños que se ven correctos en un diagrama pero fallan al operar.

■ Flujo de razonamiento



Desarrollo

1. Una pregunta decide el servicio: ¿se consume o se observa?

Los tres nombres contienen la palabra «evento» o «bus» y eso lleva a elegir por eufonía. La pregunta que separa de verdad es otra:

```
¿el mensaje representa TRABAJO que alguien debe hacer exactamente una vez?
   → Service Bus.  Se consume: al completarlo, desaparece.

¿el mensaje representa un HECHO que varios interesados leen a su ritmo?
   → Event Hubs.  Se observa: leerlo no lo borra, y otro puede releerlo.

¿el mensaje es una NOTIFICACIÓN que hay que repartir a destinos que no conozco?
   → Event Grid.  Se enruta: se entrega por HTTP y no se guarda.
```

La tabla que hace operativa esa elección:

	Service Bus	Event Hubs	Event Grid
Modelo	Cola y tema con suscripciones	Registro con particiones	Enrutador de sucesos
¿Leer borra?	Sí, al completar	No	No aplica: se envía

	Service Bus	Event Hubs	Event Grid
Retención	Hasta completar o caducar	1-7 días (90 en premium)	Solo reintentos
Orden	Por sesión	Por partición	Sin garantía
Mensajes fallidos	Automático	No existe	Configurable, apagado por defecto
Tamaño	256 KB (100 MB en premium)	1 MB	1 MB
Se paga por	Operación o unidad de mensajería	Unidad de rendimiento	Operación

La fila de mensajes fallidos es la que más consecuencias tiene. En Service Bus la subcola existe siempre y recoge lo que falla; en Event Hubs no hay tal cosa, porque el modelo no es de consumo: un evento que el consumidor no sabe procesar se queda en su sitio y, si el consumidor reintenta desde el mismo desplazamiento, bloquea la partición entera — nada posterior se procesa hasta que alguien intervenga. Poner órdenes de pedido en Event Hubs por creer que es «el servicio de eventos» es el error de diseño más caro de esta clase.

El reparto habitual en una plataforma como CloudShop, escrito para no rediscutirlo:

```
Service Bus   crear pedido · cobrar · emitir factura · enviar correo
              trabajo con dueño, con reintentos y con fallo visible

Event Hubs    clics, telemetría de la aplicación, eventos de stock
              volumen alto, varios consumidores, se puede releer

Event Grid    "se subió un blob" · "se creó un recurso" · "se desplegó"
              reacción a hechos de la plataforma y notificación a terceros
```

Y un uso de Event Grid que rentabiliza el trabajo de las clases anteriores: todos los recursos de Azure emiten eventos. Un blob nuevo en el contenedor de facturas dispara la función que lo indexa; la creación de un recurso sin etiquetas dispara la corrección de gobierno de la clase 037. Es la vía por la que la plataforma reacciona a sí misma sin sondeos periódicos.

2. El bloqueo, el reloj que casi nunca se dimensiona

Service Bus recibe de dos formas, y solo una es defendible en producción:

```
recibir y borrar    el mensaje desaparece al entregarse
                    si el proceso falla después, el trabajo se pierde
bloqueo por inspección  el mensaje queda reservado; hay que completarlo,
                    abandonarlo o trasladarlo a fallidos explícitamente
```

El bloqueo cumple la función del tiempo de invisibilidad de la clase 033 y añade una capacidad que aquel no tiene: se puede renovar. Esa diferencia cambia el diseño.

La duración máxima del bloqueo es de cinco minutos. Si el manejador tarda más y nadie renueva:

```
t+0:00  se entrega el mensaje, bloqueo de 5 min
t+5:00  el bloqueo expira; el mensaje vuelve a estar disponible
t+5:01  OTRO consumidor lo recibe y empieza a procesarlo
t+6:20  el primero termina y llama a completar
        → MessageLockLostException
t+7:40  el segundo también termina
        → el pedido se ha cobrado dos veces
```

La secuencia importa porque el error aparece al final, cuando el daño ya está hecho. Y el MessageLockLostException en el registro se suele leer como un fallo transitorio del SDK.

Tres respuestas, de mejor a peor:

```
1. sacar el trabajo largo del manejador
   el mensaje dispara el proceso, no lo ejecuta
2. renovar el bloqueo mientras se trabaja
   los SDK lo hacen automáticamente SI el hilo no está bloqueado
3. subir la duración del bloqueo
   tope de 5 minutos: solo mueve el problema un poco más allá
```

La segunda tiene una letra pequeña que produce incidentes reproducibles: la renovación automática ocurre en segundo plano y necesita que el hilo pueda ejecutarse. Un manejador que bloquea el hilo con una llamada síncrona impide su propia renovación, así que el bloqueo caduca justo en los casos lentos, que son los que más lo necesitan.

El traslado automático a fallidos es la otra mitad, y en Azure viene puesto:

```
MaxDeliveryCount = 10 por defecto
al superarlo → subcola de fallidos de la propia entidad
```

No hay que crear ninguna cola ni configurar ninguna política de reenvío: existe. Y como existe sin que nadie la pida, también se llena sin que nadie la mire. El motivo queda registrado y es la primera cosa que hay que consultar:

```
$ az servicebus queue show -g rg-msg --namespace-name sb-cloudshop -n pedidos \
    --query "countDetails.deadLetterMessageCount" -o tsv
8412
```

```
DeadLetterReason
  MaxDeliveryCountExceeded   el manejador falló 10 veces: mira su excepción
  TTLExpiredException        caducó sin que nadie lo consumiera
  HeaderSizeExceeded         propiedades demasiado grandes
  <motivo propio>            el manejador lo trasladó a propósito
```

La distinción entre las dos primeras es diagnóstica: la primera es un consumidor que falla, la segunda es un consumidor que no está. Y la alerta que hay que tener en los tres servicios es la misma y casi nunca está: cuenta de mensajes fallidos mayor que cero durante N minutos. Un sistema asíncrono sano no acumula fallidos; que los acumule en silencio es la forma en que se pierde trabajo sin que nadie reciba un error.

3. Detección de duplicados, sesiones y filtros: lo que garantizan y lo que no

La clase 033 separó el duplicado de entrega del fallo de idempotencia. Azure ofrece un mecanismo que parece resolverlo y solo cubre una parte:

```
detección de duplicados
  descarta mensajes con el mismo MessageId
  DENTRO de una ventana temporal
  por defecto 30 s; hasta 7 días en premium
```

La ventana es el detalle que decide. Un cliente que reintenta a los 45 segundos con la ventana en 30 crea un duplicado real, y el sistema lo entrega dos veces. Además la ventana consume almacenamiento y capacidad: fijarla en siete días para «asegurarse» tiene un precio y sigue sin cubrir el reintento manual de un operador tres semanas después.

La conclusión es la de la clase 033, reforzada: la detección de duplicados reduce el ruido; la idempotencia del manejador es la que garantiza corrección. Un manejador idempotente hace innecesaria la primera; la primera nunca hace innecesario al segundo.

```
def manejar(mensaje):
    clave = mensaje.application_properties[b"idempotency-key"].decode()
    if repositorio.ya_procesado(clave):        # la garantía real
        return                                  # completar sin reejecutar
    with repositorio.transaccion():
        aplicar_efecto(mensaje)
        repositorio.marcar_procesado(clave)     # en la MISMA transacción
```

La última línea es la que hace que funcione: marcar y aplicar el efecto en la misma transacción. Marcadas por separado, un fallo entre ambas reproduce exactamente el problema que se quería evitar.

Las sesiones dan orden dentro de un identificador —todos los mensajes de un pedido, en secuencia— y tienen un costo estructural que hay que anticipar: una sesión la bloquea un único consumidor, así que dentro de una sesión no hay paralelismo. Si el identificador de sesión es demasiado grueso, se ha construido una cola de un solo hilo:

```
sesión = idPedido      miles de sesiones, buen reparto            ✓
sesión = idTienda      una tienda grande serializa todo su trabajo ✗
sesión = "pedidos"     una sola sesión: concurrencia = 1          ✗✗
```

Es el mismo razonamiento de la clave de partición de la clase 042: la unidad de orden es también la unidad de serialización. Solo se pide orden donde el negocio lo exige, y con el identificador más fino que lo satisfaga.

Los temas con suscripciones y filtros sustituyen al abanico de la clase 033, con dos matices:

```
$ az servicebus topic subscription rule create -g rg-msg \
    --namespace-name sb-cloudshop --topic-name pedidos --subscription-name facturacion \
    --name solo-pagados --correlation-filter properties.estado=pagado
```

Primero: un filtro de correlación —igualdad sobre propiedades— se evalúa mucho más barato que uno con sintaxis SQL. Con volumen alto, la diferencia es de factura.

Segundo: una suscripción sin ninguna regla recibe todo, y una suscripción creada hoy no recibe lo publicado ayer. La primera produce consumidores que procesan mensajes que no les tocaban; la segunda produce el hueco de datos al añadir un consumidor nuevo, que hay que rellenar a mano si importaba.

Y una decisión de nivel que se toma tarde y duele: el nivel estándar es multiinquilino, factura por operación y puede limitar; el premium reserva unidades de mensajería, da latencia predecible, admite mensajes de 100 MB y es el único que acepta punto de conexión privado. Ese último punto conecta con la clase 039: si la plataforma cerró los accesos públicos, el nivel estándar no puede participar. El salto de precio es considerable, así que conviene decidirlo con el diseño de red delante y no cuando ya no hay alternativa.

4. Event Hubs: particiones, puntos de control y el inflado que no desinfla

Event Hubs es un registro particionado, no una cola. Las tres consecuencias que definen su operación:

```
el orden es POR PARTICIÓN, nunca global
leer NO borra: el consumidor avanza un marcador propio
cada grupo de consumidores tiene su propio marcador
```

La clave de partición decide a qué partición va cada evento, y por tanto qué queda ordenado respecto a qué. Es la misma decisión de la clase 042 con otro nombre. Y el número de particiones es, en la práctica, una decisión de un solo sentido: cambiarlo altera a qué partición corresponde cada clave, así que el orden por clave se rompe en el punto del cambio y los consumidores que dependían de él procesan eventos desordenados durante la transición.

```
regla útil    particiones ≥ consumidores concurrentes máximos previstos
              porque una partición la lee UN consumidor por grupo
              particiones de más: coste de gestión, no de dinero
              particiones de menos: techo de paralelismo que no se puede subir
                                    sin romper el orden
```

El punto de control es la otra fuente de incidentes. El consumidor guarda su posición en una cuenta de almacenamiento, y la frecuencia con la que lo hace define cuánto se reprocesa tras un reinicio:

```
punto de control cada 10.000 eventos, a 30.000 eventos/min
→ hasta 20 s de eventos reprocesados en cada reinicio

punto de control cada 10.000 eventos, a 500 eventos/min
→ hasta 20 MINUTOS reprocesados en cada reinicio
```

La misma configuración produce dos comportamientos incomparables según el caudal. Por eso el punto de control se fija por tiempo además de por cantidad, y por eso el consumidor tiene que ser idempotente igual que el de Service Bus: el reprocesamiento no es una anomalía, es el funcionamiento normal tras cualquier reinicio.

Y un fallo de operación con firma propia: dos instancias del mismo grupo de consumidores compitiendo por la misma partición. La segunda toma el arrendamiento y expulsa a la primera; la primera reintenta y expulsa a la segunda. El resultado es un vaivén de arrendamientos, ningún avance del marcador y un consumo de CPU alto sin trabajo hecho. Se reconoce en el registro por el ciclo continuo de adquisición y pérdida de particiones.

El rendimiento se compra en unidades y ahí está la trampa de costo de esta clase:

```
1 unidad de rendimiento ≈ 1 MB/s de entrada, 2 MB/s de salida, 1.000 eventos/s
al superarlo → ServerBusyException, que el SDK reintenta
inflado automático → SUBE solo hasta el máximo configurado
                  → y NO BAJA nunca
```

Una prueba de carga que empuje de 2 a 20 unidades deja el espacio de nombres en 20 hasta que una persona lo baje:

```
2 unidades de rendimiento       ~44
20 tras la prueba de carga     ~440
```

Cuatrocientos dólares al mes por un experimento de veinte minutos, sin ninguna alerta que lo señale. La medida preventiva es fijar el máximo del inflado en un valor que se pueda pagar y revisar el valor actual como parte del cierre de cualquier prueba de carga.

Dos capacidades que conviene conocer porque ahorran trabajo:

Captura. Archiva automáticamente en almacenamiento en formato Avro, por ventana de tiempo o de tamaño. Es la forma más barata de tener el flujo en bruto para reprocesar, y encaja con las reglas de ciclo de vida de la clase 041 sin escribir un solo consumidor.

Compatibilidad con Kafka. Event Hubs habla el protocolo de Kafka, así que un productor o consumidor existente funciona cambiando la cadena de conexión. Para un programa multinube esto es un dato de arquitectura, no una curiosidad: el contrato de la aplicación deja de ser específico del proveedor aunque el servicio lo sea.

5. Event Grid: entrega por HTTP, reintentos y el saludo que falla

Event Grid empuja, no se consulta. Eso simplifica el consumidor —una función HTTP y nada más— y traslada al emisor tres responsabilidades que en una cola no existían.

El saludo de validación. Antes de entregar nada a un extremo HTTP, Event Grid envía un evento de validación y espera que el extremo devuelva el código recibido. Es la protección contra usar el servicio para inundar a un tercero, y es el fallo de puesta en marcha número uno: la suscripción se crea, no entrega nada y el extremo no registra ningún error porque nunca llegó a llamarse para lo demás.

```
if tipo == "Microsoft.EventGrid.SubscriptionValidationEvent":
    return {"validationResponse": datos["validationCode"]}
```

Con funciones de Azure o colas como destino, el saludo lo resuelve la plataforma; con un extremo propio, hay que escribirlo.

El reintento y su final. Event Grid reintenta con retroceso exponencial hasta 24 horas por defecto. Pasado ese plazo, descarta el evento, y el destino de mensajes fallidos está apagado:

```
$ az eventgrid event-subscription update --name indexar-facturas \
    --source-resource-id $ST_ID \
    --deadletter-endpoint "$ST_ID/blobServices/default/containers/eventos-fallidos"
```

Sin esa línea, un extremo caído durante un día produce pérdida de eventos sin traza. Con ella, los eventos aterrizan en un contenedor con su motivo, y la lección de la clase 041 vuelve a aplicar: ese contenedor necesita su propia regla de ciclo de vida o crece para siempre.

Ni orden ni unicidad. Event Grid no garantiza el orden y puede entregar el mismo evento más de una vez. Un manejador que asume orden —«el evento de creación llega antes que el de actualización»— funciona en desarrollo y falla en producción de forma intermitente. La defensa es la misma de todo el resto de la clase: idempotencia, y usar la marca de tiempo o la versión del recurso para descartar lo viejo en vez de confiar en el orden de llegada.

Un cierre que une las tres piezas. Los tres servicios entregan al menos una vez, ninguno entrega exactamente una vez, y los tres exigen lo mismo del manejador:

```
Service Bus   duplica al expirar un bloqueo o al reintentar el cliente
Event Hubs    duplica al reprocesar desde el último punto de control
Event Grid    duplica al reintentar la entrega
```

La entrega exactamente una vez no se compra: se construye en el manejador. Es la misma conclusión de la clase 033, y después de tres servicios con tres mecanismos distintos ya no es una recomendación teórica: es la única propiedad que se conserva al cambiar de servicio, y por tanto la única que merece la pena escribir una sola vez y reutilizar.

Ejemplo trabajado

CloudShop monta su columna asíncrona en Azure. El equipo elige Event Hubs para todo «porque es el servicio de eventos y escala más». Cinco incidentes en un mes reparten cada flujo donde le corresponde.

Punto de partida:

```
eh-cloudshop (Event Hubs, 4 particiones, 2 unidades de rendimiento)
  pedidos · pagos · telemetria · notificaciones
todos los consumidores en el mismo grupo, punto de control cada 10.000 eventos
```

Incidente 1 — una partición deja de avanzar y 12.000 pedidos se detienen.

Un pedido con un campo nuevo hace fallar al deserializador. El consumidor reintenta desde el mismo desplazamiento, indefinidamente.

```
partición 2   desplazamiento 4.481.209   sin avanzar desde hace 3 h 40 min
pedidos detenidos detrás del envenenado   12.184
```

No hay subcola de fallidos: Event Hubs no la tiene, porque leer no consume. La solución de urgencia es saltar el desplazamiento a mano; la de fondo es mover el flujo:

```
pedidos          Event Hubs        Service Bus (cola con sesiones por idPedido)
```

```
pagos            Event Hubs        Service Bus (tema con suscripciones filtradas)
telemetria       Event Hubs        Event Hubs                      ← se queda
notificaciones   Event Hubs        Event Grid (reacción a blob y a estado)
```

El criterio queda escrito en una línea: lo que alguien debe hacer una vez va a Service Bus; lo que muchos observan a su ritmo se queda en Event Hubs.

Incidente 2 — dos pedidos cobrados dos veces.

Con la cola ya en Service Bus, el manejador de cobro tarda entre 4 y 7 minutos porque llama a la pasarela y espera la confirmación.

```
MessageLockLostException al completar    41 veces en 24 h
cobros duplicados detectados por conciliación   2
```

El bloqueo era de cinco minutos y el manejador bloqueaba el hilo con una llamada síncrona, impidiendo su propia renovación. Se corrige por el lado del diseño, no del reloj:

```
trabajo dentro del manejador    llamada de 4-7 min   registrar intención y salir
cobro efectivo                  en el manejador      proceso propio con reintento
duración del bloqueo                 5 min              30 s (sobra)
clave de idempotencia               ninguna          idPedido + intento,
                                                     marcada en la MISMA transacción
cobros duplicados                       2                  0
```

La clave de idempotencia es la que cierra el caso: aunque el mensaje vuelva a entregarse, el efecto no se repite.

Incidente 3 — 8.412 mensajes fallidos que nadie miró en tres semanas.

```
$ az servicebus queue show -g rg-msg --namespace-name sb-cloudshop -n facturacion \
    --query "countDetails.deadLetterMessageCount" -o tsv
8412
$ # motivo del primero
MaxDeliveryCountExceeded · System.NullReferenceException en importe_neto
```

Un cambio de esquema tres semanas atrás. Cada mensaje se reintentó diez veces y acabó en la subcola, que existía desde el principio y no tenía ninguna alerta.

```
alerta sobre mensajes fallidos       ninguna     > 0 durante 15 min → aviso
procedimiento de reproceso           ninguno     documentado y ensayado
validación de esquema en el emisor   no          sí, contrato versionado
mensajes recuperados                  —          8.412 reprocesados
```

Incidente 4 — eventos de facturas perdidos durante un día.

La función que indexa facturas estuvo caída 26 horas por un despliegue fallido. Al recuperarla, faltan 1.900 documentos.

```
$ az eventgrid event-subscription show --name indexar-facturas \
    --source-resource-id $ST_ID --query "deadLetterDestination"
null
```

Event Grid reintentó 24 horas y descartó el resto. Sin destino de fallidos, no queda rastro. Se configura, y se reconstruye el hueco releyendo el contenedor —posible porque los blobs siguen ahí, no porque el sistema de eventos lo permitiera.

```
destino de mensajes fallidos       ninguno    contenedor eventos-fallidos
regla de ciclo de vida sobre él        —      borrado a 30 días (clase 041)
alerta de entregas fallidas           no      > 50 en 10 min → aviso
```

Incidente 5 — la factura de Event Hubs se multiplica por diez sin más tráfico.

```
$ az eventhubs namespace show -g rg-msg -n eh-cloudshop \
    --query "{tu:sku.capacity,inflar:isAutoInflateEnabled,max:maximumThroughputUnits}" -o tsv
20   True   20
```

Veinte unidades de rendimiento con un tráfico que necesita dos. El inflado automático había subido durante una prueba de carga de veinte minutos, tres semanas atrás, y nunca bajó — porque no baja.

```
unidades de rendimiento               20            2
máximo del inflado automático         20            6
costo mensual de Event Hubs         ~440 USD      ~44 USD
revisión tras prueba de carga       ninguna    paso obligatorio del cierre
```

Y de paso se corrige el punto de control, que con el caudal real de telemetría reprocesaba seis minutos en cada reinicio:

```
punto de control   cada 10.000 eventos   →   cada 10.000 eventos o 30 s
reproceso tras reinicio   ~6 min          →   < 30 s
```

Resumen de la columna asíncrona:

```
servicios usados                        1 (para todo)    3, por criterio escrito
trabajo detenido sin señal              12.184 pedidos          0
cobros duplicados                             2                 0
mensajes fallidos sin vigilar             8.412           alertados a 15 min
eventos perdidos sin rastro               1.900     con destino de fallidos
unidades de rendimiento pagadas              20                 2
manejadores idempotentes                   0 de 4            4 de 4
costo mensual de mensajería               ~465 USD         ~118 USD
```

La lección que esta clase traslada al resto de la parte: los tres servicios entregan al menos una vez y ninguno entrega exactamente una vez. Lo que hace correcto a un sistema asíncrono no es el servicio elegido sino la propiedad que se escribe en el manejador, y esa propiedad es la única que sobrevive intacta cuando el servicio cambia — que es exactamente lo que este programa entiende por portabilidad.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-03-azure-core-platform/044-service-bus-event-grid-y-event-hubs/lab.py
```

El laboratorio selecciona el motor de práctica messaging y produce labresult.json. El escenario, sus comprobaciones y el artefacto esperado corresponden a esta clase; no requiere credenciales y deja explícito qué debe revalidarse en un sandbox real.

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa flujo-eventos-azure en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es un flujo que declara orden, entrega y manejo de errores. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye flujo-eventos-azure para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

Errores frecuentes

Síntoma	Causa probable	Corrección
Una partición deja de avanzar y el trabajo posterior se detiene sin errores visibles	Un evento envenenado en Event Hubs, que no tiene subcola de mensajes fallidos porque leer no consume	Usa Service Bus para lo que debe procesarse una vez; en Event Hubs, aparta el evento y avanza el desplazamiento con registro del descarte.
MessageLockLostException al completar y efectos duplicados	El manejador tarda más que el bloqueo y bloquea el hilo, impidiendo su propia renovación	Saca el trabajo largo del manejador, no bloquees el hilo, y aplica una clave de idempotencia en la misma transacción que el efecto.
Miles de mensajes en la subcola de fallidos sin que nadie se entere	El traslado automático existe desde el principio y no tiene alerta asociada	Alerta sobre la cuenta de mensajes fallidos mayor que cero y documenta un procedimiento de reproceso ensayado.
Se pierden eventos de Event Grid tras una caída prolongada del destino	El reintento dura 24 h y el destino de mensajes fallidos está apagado por defecto	Configura el destino de fallidos, ponle su regla de ciclo de vida y alerta sobre las entregas fallidas.
La factura de Event Hubs se multiplica sin que el tráfico haya cambiado	El inflado automático sube y nunca baja, y una prueba de carga dejó el valor alto	Fija un máximo asumible y revisa las unidades activas como paso obligatorio del cierre de toda prueba de carga.
Se activa la detección de duplicados y siguen llegando repetidos	La ventana es acotada —30 s por defecto— y el reintento cae fuera de ella	Trátala como reducción de ruido: la corrección la garantiza la idempotencia del manejador.

Seguridad, ética y costo

Trabaja con cuentas propias o sandboxes autorizados. No publiques secretos, identificadores reales ni datos personales. Antes de crear recursos pagos define presupuesto, etiquetas y comando de destrucción; después verifica que no queden recursos huérfanos. Los ejemplos locales enseñan contratos, pero no certifican cumplimiento ni disponibilidad de producción.

Preguntas de comprobación

1. ¿Qué pregunta separa Service Bus de Event Hubs, y qué ocurre exactamente si se elige mal para un flujo de pedidos?
2. Un manejador tarda siete minutos y el bloqueo dura cinco. Describe la secuencia completa hasta el efecto duplicado.
3. ¿Por qué la detección de duplicados no sustituye a la idempotencia, y qué hace idempotente a un manejador?
4. ¿Qué se rompe al cambiar el número de particiones de un centro de eventos, y qué regla usarías para elegirlo?
5. ¿Qué tres cosas hay que configurar en Event Grid que no vienen puestas, y qué se pierde sin cada una?

Referencias

- Microsoft (2025). Compare Azure messaging services — cuándo Service Bus, cuándo Event Grid y cuándo Event Hubs. <https://learn.microsoft.com/en-us/azure/service-bus-messaging/compare-messaging-services>
- Microsoft (2025). Service Bus dead-letter queues — traslado automático, motivos y reproceso. <https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dead-letter-queues>
- Microsoft (2025). Message sessions and duplicate detection — orden por sesión y ventana de duplicados. <https://learn.microsoft.com/en-us/azure/service-bus-messaging/message-sessions>
- Microsoft (2025). Event Hubs features — particiones, grupos de consumidores, puntos de control y captura. <https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-features>
- Microsoft (2025). Event Grid delivery and retry — saludo de validación, reintentos y mensajes fallidos. <https://learn.microsoft.com/en-us/azure/event-grid/delivery-and-retry>

• Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 03 en PDF · Recorrido de Azure en PDF · Manual integral

Anterior	Índice	Siguiente
← 043 · App Service, Functions y Container Apps	Parte 03 · Programa	045 · Azure Monitor, Log Analytics y Application Insights →

Evaluación — 044 Service Bus, Event Grid y Event Hubs

Preguntas

1. Explica el problema que resuelve esta capacidad sin mencionar un proveedor.
2. Identifica una frontera de responsabilidad y su propietario.
3. Compara dos alternativas mediante confiabilidad, seguridad, costo y operación.
4. ¿Qué demuestra el laboratorio y qué no puede demostrar?
5. Propón un caso límite o una prueba de fallo.

Reto verificable

Entrega flujo-eventos-azure con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

• [ ] Resultado reproducible por una segunda persona.

• [ ] Evidencia vinculada a cada afirmación técnica.

• [ ] Prueba negativa o de fallo incluida.

• [ ] Costos y permisos expresados con unidades y alcance.

• [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.

045 — Azure Monitor, Log Analytics y Application Insights

Parte: 03 — Azure: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: observability · Estado: EXECUTABLECORE

Propósito

Construir la evidencia operativa de una plataforma Azure sabiendo dos cosas que cambian todo lo demás: los registros de recurso están apagados por defecto, así que un incidente ocurrido antes de activarlos no tiene datos que analizar; y la misma señal puede almacenarse como métrica o como registro, con una diferencia de precio de dos órdenes de magnitud. La clase 034 dejó las cuatro preguntas de un incidente; aquí se responden con KQL y se paga la factura correcta por hacerlo.

Resultados de aprendizaje

Al finalizar podrás:

1. Distinguir métricas de registros por costo, latencia y capacidad de consulta, y elegir dónde va cada señal.
2. Activar registros de recurso mediante directiva en vez de recurso por recurso, y comprobar la cobertura.
3. Escribir consultas KQL que respondan tasa de error, percentil de latencia y reconstrucción de una traza.
4. Reducir la factura de ingesta con filtrado en la regla de recopilación y planes por tabla, sin perder la señal.
5. Elegir entre alerta de métrica y alerta de registro sabiendo qué latencia añade cada una.

Conceptos centrales

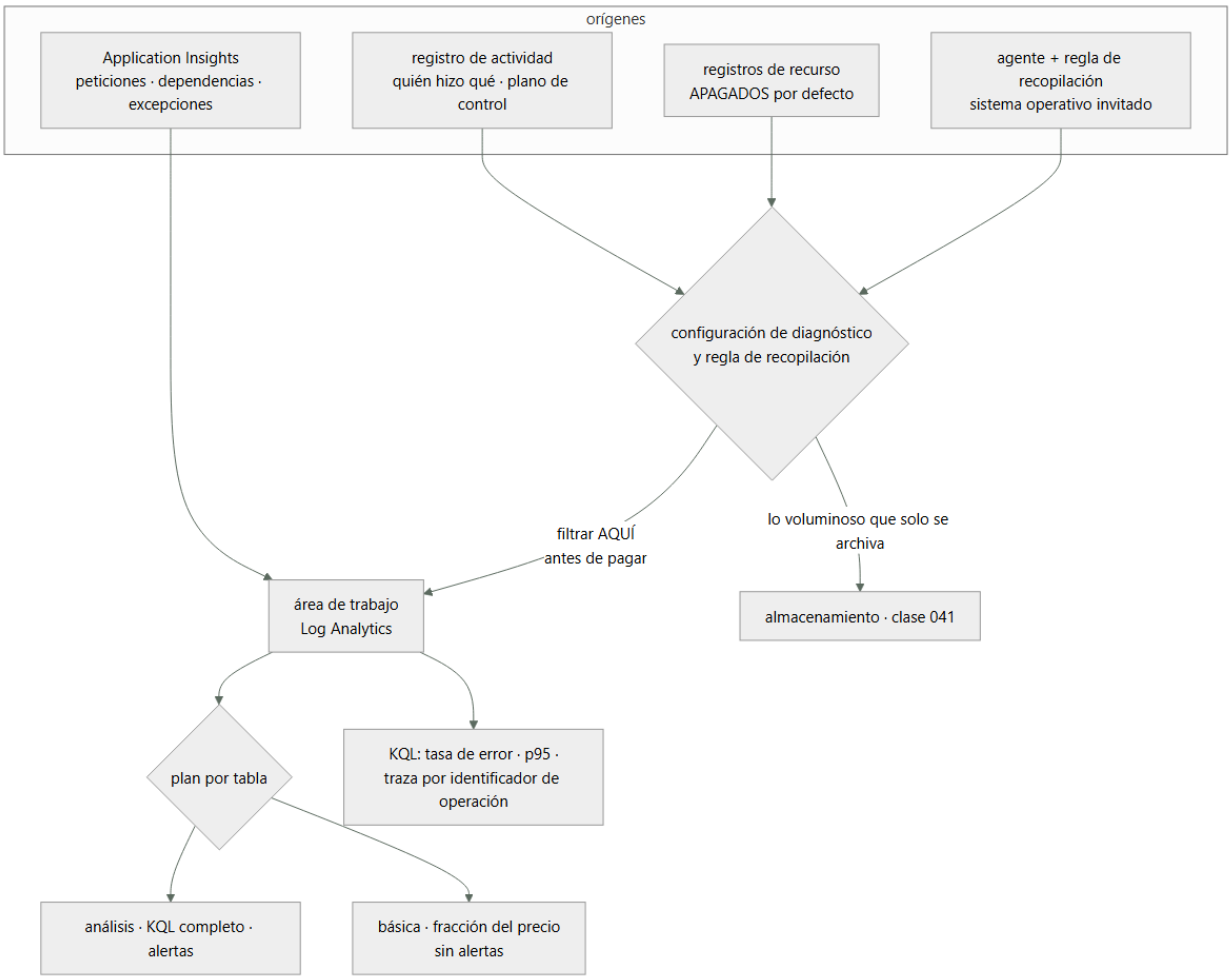
Concepto	Comprensión verificable
configuración de diagnóstico	Interruptor que envía los registros de un recurso a un destino. Está apagado por defecto en todos los recursos: sin él, el plano de datos no deja rastro.
regla de recopilación de datos	Define qué se recoge, cómo se transforma y adónde va. Es el punto donde se filtra antes de pagar la ingesta, no después.
plan por tabla	Cada tabla del área de trabajo puede ser de análisis, básica o auxiliar. La básica cuesta una fracción y renuncia a alertas y a parte de KQL.
muestreo adaptativo	Reducción automática del volumen que envía el SDK de Application Insights. Hace que count() mienta: hay que sumar itemCount.
identificador de operación	Correlador que atraviesa servicios siguiendo el estándar de traza del W3C. Es lo que convierte cuatro registros dispersos en una sola historia.
alerta de registro frente a alerta de métrica	La de métrica se evalúa en la canalización de métricas y avisa en cerca de un minuto; la de registro espera a la ingesta y a su ventana, y rara vez baja de cinco.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Aplicado a esta clase, separa siempre cuatro planos: intención (qué necesita el usuario), configuración (qué declaramos), estado observado (qué existe de verdad) y evidencia (cómo sabemos que cumple). Confundirlos produce diseños que se ven correctos en un diagrama pero fallan al operar.

■ Flujo de razonamiento



Desarrollo

1. Nada se registra hasta que alguien lo enciende

Esta es la frase que hay que aprender antes que cualquier consulta. En Azure hay cuatro orígenes de telemetría y solo uno funciona sin configurar nada:

```
registro de actividad     plano de control: quién creó, modificó o borró qué
                          ACTIVO siempre · 90 días · gratis
registros de recurso      plano de datos: consultas SQL, accesos a blobs,
                          peticiones de la pasarela
                          APAGADOS por defecto en TODOS los recursos
sistema operativo invitado agente + regla de recopilación
                          hay que instalarlo
aplicación                Application Insights
                          hay que instrumentarla
```

La segunda línea es la que produce la conversación más incómoda de un postmortem: hubo una caída, se abre el portal a buscar qué pasó y no hay nada que mirar, porque la configuración de diagnóstico de ese recurso nunca se creó. Y no se puede recuperar: los registros no existían.

Activarlo recurso por recurso no funciona a escala, por el mismo motivo que cualquier control manual: alguien despliega uno nuevo y nadie se acuerda. La solución es la de la clase 037 aplicada a la telemetría — una directiva que lo despliegue por sí sola:

```
$ az policy assignment create --name diagnostico-obligatorio \
    --scope "/subscriptions/$SUB" \
    --policy-set-definition "Deploy Diagnostic Settings to Azure Services" \
    --location westeurope --identity-scope "/subscriptions/$SUB" \
    --role Contributor -p "{\"logAnalytics\":{\"value\":\"$WORKSPACE_ID\"}}"
```

El efecto DeployIfNotExists corrige lo que existe y cubre lo que venga. Y la comprobación —que es el entregable, no la asignación— consiste en preguntar cuántos recursos siguen sin ella:

```
resources
| where type in~ ("microsoft.sql/servers/databases",
                  "microsoft.network/applicationgateways",
                  "microsoft.storage/storageaccounts")
| join kind=leftanti (
    insightsresources
    | where type =~ "microsoft.insights/diagnosticsettings"
    | project id = tolower(tostring(split(id, "/providers/microsoft.insights")[0]))
  ) on $left.id == $right.id
| summarize sin_diagnostico = count() by type
```

Sobre el registro de actividad hay un límite que sorprende tarde: se conserva 90 días y no más. Una investigación sobre algo ocurrido hace cuatro meses no encuentra nada. Si la organización necesita responder «quién borró esto» más allá de ese plazo —y las auditorías lo necesitan—, hay que exportarlo con una configuración de diagnóstico, y el destino natural es una cuenta de almacenamiento con inmutabilidad de la clase 041: un registro de auditoría que el propio administrador puede borrar no es un registro de auditoría.

2. Métricas y registros: la misma señal con dos precios

Azure Monitor guarda dos tipos de datos, y la elección entre ellos es una decisión de arquitectura con consecuencias de factura:

	Métricas	Registros
Forma	Serie temporal numérica	Registros estructurados
Latencia hasta ser consultable	1 min	1-3 min, más si hay limitación
Retención	93 días, incluida	Configurable, se paga
Dimensiones	Pocas y acotadas	Las que quieras
Costo de las de plataforma	Prácticamente nulo	2,30 USD por GB ingerido
Sirven para	Alertar rápido, ver tendencia	Investigar, correlacionar, auditar

La clase 034 avisaba de que la cardinalidad multiplica la factura. En Azure ese principio se concreta en una pregunta por cada señal: ¿necesito filtrar esto por un valor que no conozco de antemano? Si la respuesta es no —CPU, latencia media, profundidad de cola—, es una métrica y sale casi gratis. Si es sí —el identificador del cliente, la ruta exacta, el mensaje de error—, es un registro y se paga por GB.

El error caro consiste en emitir como registro lo que podía ser métrica: una línea de registro por petición para poder contar peticiones. Contar peticiones es exactamente lo que hace una métrica, por una fracción del precio y con menos latencia.

Y hay un tercer nivel entre ambos que casi nadie usa y resuelve el caso voluminoso: los planes por tabla.

```
análisis    ~2,30 USD/GB   KQL completo, alertas, 31 días incluidos
básica      ~0,65 USD/GB   consultas limitadas, SIN alertas, 30 días
auxiliar    ~0,15 USD/GB   volcado barato para búsquedas ocasionales
archivo     ~0,03 USD/GB/mes  hay que lanzar un trabajo de búsqueda para leerlo
```

La decisión se toma por tabla, no por área de trabajo, y ahí está la palanca: los registros de acceso de una pasarela o la salida estándar de decenas de contenedores suelen ser el 70-80 % del volumen y casi nunca son el origen de una alerta. Ponerlos en plan básico no cambia lo que se puede investigar durante un incidente; cambia el precio de guardarlos.

El otro punto de control está antes: la regla de recopilación puede transformar y descartar antes de la ingesta, que es cuando se paga.

```
source
| where TimeGenerated > ago(1h)
| where httpStatus_d >= 400 or timeTaken_d > 1.0
| project TimeGenerated, clientIP_s, requestUri_s, httpStatus_d, timeTaken_d
```

Dos efectos a la vez: se descarta el 95 % de las líneas —las peticiones correctas y rápidas, que ya están contadas en una métrica— y se recorta el ancho de las que quedan. Filtrar después, en la consulta, no ahorra un céntimo: el GB ya se pagó al entrar.

Y una regla de higiene que evita la sorpresa: poner un presupuesto y un límite diario al área de trabajo desde el primer día. El límite corta la ingesta al alcanzarlo, lo cual es malo; despertar un lunes con una factura de cuatro cifras por un componente que se volvió locuaz el viernes es peor, y sin límite no hay nada que lo impida.

3. KQL suficiente para responder las cuatro preguntas

La clase 034 planteaba cuatro preguntas durante un incidente: qué está roto, desde cuándo, qué cambió y quién lo hizo. Con un área de trabajo, las cuatro se responden con el mismo lenguaje.

Qué está roto y cuánto.

```
requests
| where timestamp > ago(1h)
| summarize total = sum(itemCount),
            fallidas = sumif(itemCount, success == false) by operation_Name
| extend tasa_error = round(100.0 * fallidas / total, 2)
| where total > 100
| order by tasa_error desc
```

Desde cuándo, y con qué latencia.

```
requests
| where timestamp > ago(6h) and operation_Name == "POST /api/pedidos"
| summarize p50 = percentile(duration, 50),
            p95 = percentile(duration, 95),
            p99 = percentile(duration, 99) by bin(timestamp, 5m)
| render timechart
```

Qué dependencia lo causa.

```
dependencies
| where timestamp > ago(1h) and success == false
| summarize fallos = sum(itemCount) by target, type, resultCode
| order by fallos desc
```

La historia completa de una petición concreta, que es lo que convierte cuatro paneles en un diagnóstico:

```
let op = "8f2a1c9e4b7d3a05";
union requests, dependencies, exceptions, traces
| where operation_Id == op
| project timestamp, itemType, name, target, resultCode, duration, message
| order by timestamp asc
```

Ese operationId viaja entre servicios en la cabecera traceparent del estándar del W3C. Es la pieza que hace que la traza cruce la pasarela, la aplicación web, la función y la cola. Y tiene una condición: cada salto debe propagar la cabecera. Un cliente HTTP escrito a mano que no la reenvíe corta la traza justo donde empieza a ser interesante.

Quién lo hizo, en el registro de actividad:

```
AzureActivity
| where TimeGenerated > ago(7d)
| where OperationNameValue endswith "/DELETE"
| project TimeGenerated, Caller, OperationNameValue, _ResourceId, ActivityStatusValue
| order by TimeGenerated desc
```

Y ahora la advertencia que hace inútiles todas las consultas anteriores si se ignora. El SDK de Application Insights aplica muestreo adaptativo por defecto: cuando el volumen sube, envía una fracción y anota en itemCount a cuántos representa cada registro.

```
count()                 cuenta REGISTROS ALMACENADOS      41.312
sum(itemCount)          cuenta peticiones REALES         413.120
```

Un factor de diez, silencioso, en cualquier panel escrito con count(). Y peor: una alerta con umbral sobre count() deja de dispararse justo cuando el tráfico sube, porque el muestreo se vuelve más agresivo precisamente entonces. La regla es sencilla y no admite excepción: en tablas de Application Insights se suma itemCount, nunca se cuenta.

Si para un flujo concreto —pagos, por ejemplo— hace falta el detalle completo, el muestreo se desactiva para ese tipo de telemetría en lugar de para toda la aplicación, y se acepta el costo de esa decisión sabiendo cuál es.

4. Alertas: la que llega en un minuto y la que llega en nueve

Hay tres clases de alerta y elegir mal añade minutos al tiempo de detección sin que nadie se dé cuenta:

	Se evalúa sobre	Latencia típica	Costo
Métrica	La canalización de métricas	1 min	Muy bajo por regla
Registro	Una consulta KQL programada	5-15 min	Por regla y frecuencia
Registro de actividad	Sucesos del plano de control	Minutos	Muy bajo

La latencia de una alerta de registro es acumulativa y hay que sumarla entera:

```
ingesta            1-3 min
frecuencia mínima  5 min
ventana            5-15 min según se defina
■■■■■■■■■■■■■■■■■■■■■■■■■■■■
detección real     de 7 a 20 minutos
```

Eso descalifica a la alerta de registro como señal rápida. La distribución correcta:

```
métrica    disponibilidad, tasa de error HTTP, latencia, profundidad de cola,
           CPU, mensajes fallidos  →  todo lo que despierta a alguien
registro   lo que solo se puede expresar consultando: una excepción concreta,
           una combinación de campos, una correlación entre tablas
actividad  cambios sensibles: borrado de recursos, cambios de rol,
           elevación de acceso (clase 038)
```

Dos ajustes que separan una alerta útil de un generador de ruido:

Umbrales dinámicos. Aprenden la estacionalidad de una métrica y avisan de la desviación. Son excelentes para tráfico con patrón diario y semanal —una tienda lo tiene— y son inútiles para una señal que nunca ha estado sana: el modelo aprende que la anomalía es lo normal.

Grupos de acción con la severidad bien puesta. Una alerta que no corresponde a una acción no debería despertar a nadie. El criterio de la clase 034 se mantiene: si la respuesta a una alerta es «ya, eso pasa a veces», la alerta está mal definida o el sistema está mal.

Y la alerta que casi nunca existe y que esta parte ha justificado tres veces —clase 043 con el consumidor en cero réplicas y clase 044 con la subcola de fallidos—: la que detecta trabajo que se acumula. Un proceso caído se nota; un proceso que se apagó ordenadamente y dejó de trabajar no.

```
profundidad de cola creciente durante N minutos
edad del mensaje más antiguo por encima de un umbral
cuenta de mensajes fallidos mayor que cero
retraso del consumidor de Event Hubs respecto al último desplazamiento
```

Las cuatro son métricas, así que son baratas y rápidas. No tenerlas no es una cuestión de presupuesto: es que nadie se acordó de que un sistema puede fallar sin caerse.

Sobre el diseño del área de trabajo, una regla que evita una discusión recurrente: se separa por control de acceso y residencia de datos, no por orden estético. Una sola área de trabajo con acceso en contexto de recurso permite que cada equipo vea los registros de sus propios recursos sin ver los de los demás, y hace posibles las consultas que cruzan servicios — que son justamente las que resuelven los incidentes interesantes. Varias áreas por «tenerlo ordenado» obligan a consultas entre áreas de trabajo para todo y multiplican el costo fijo.

5. Operar la flota sin abrir puertos, y saber qué cambió

La clase 034 cerraba con dos capacidades que aquí tienen otras piezas y el mismo objetivo: entrar sin exponer y saber qué se movió.

Entrar sin exponer. Tres mecanismos, en orden de preferencia:

```
no entrar             la mayoría de diagnósticos se resuelven con telemetría
                      si hace falta entrar a menudo, falta instrumentación
Bastion               sesión por el portal, sin IP pública en la máquina
                      subred AzureBastionSubnet, de la clase 039
comandos remotos      `az vm run-command` ejecuta sin sesión interactiva,
                      queda registrado en el registro de actividad
```

La primera línea es la importante y suele saltarse. Una plataforma en la que hay que abrir una sesión para saber qué ocurre es una plataforma con un problema de observabilidad, no de acceso.

Y una precisión sobre el segundo: los comandos remotos se ejecutan con los permisos del agente, es decir como administrador local, y aparecen en el registro de actividad como una única operación sin el contenido del comando. Conviene saberlo por lo que implica: es un camino de ejecución privilegiada que hay que restringir con RBAC igual que cualquier otro.

Saber qué cambió. Es la pregunta que más veces resuelve un incidente, y tiene dos fuentes:

```
// cambios de configuración en las últimas 6 horas sobre los recursos afectados
AzureActivity
| where TimeGenerated > ago(6h)
| where OperationNameValue has_any ("WRITE", "DELETE", "ACTION")
| where ActivityStatusValue == "Success"
| project TimeGenerated, Caller, OperationNameValue, _ResourceId
| order by TimeGenerated asc
```

El historial de cambios de Azure Resource Graph completa la foto con el antes y el después de cada propiedad, que es lo que distingue «alguien tocó la pasarela» de «alguien cambió el tiempo de espera de 30 a 3 segundos».

Juntar ambas con la ventana del incidente es un hábito que ahorra horas:

```
1. ¿cuándo empezó?              percentil de latencia o tasa de error
2. ¿qué cambió en esa ventana?  registro de actividad e historial de cambios
3. ¿qué traza lo demuestra?     una petición fallida por identificador de operación
4. ¿quién y con qué permiso?    autor del cambio y su rol (clase 038)
```

Y la conclusión que enlaza con el proyecto de la clase 048: estas cuatro consultas se escriben una vez, se guardan en el área de trabajo y se enlazan desde el manual de operación. Un incidente no es el momento de aprender KQL. La diferencia entre una plataforma observada y una instrumentada es que la primera tiene las preguntas escritas antes de necesitarlas.

Ejemplo trabajado

CloudShop lleva un mes en Azure con todo desplegado y nada instrumentado más allá de lo que viene puesto. Una caída de 48 minutos abre cinco frentes, y el primero es que no hay datos de la caída.

Frente 1 — el incidente sin evidencia.

La tienda devuelve 502 durante 48 minutos. Al abrir el portal a investigar:

```
$ az monitor diagnostic-settings list --resource $APPGW_ID --query "value[].name" -o tsv
(vacío)
$ az monitor diagnostic-settings list --resource $SQLDB_ID --query "value[].name" -o tsv
(vacío)
```

Ni la pasarela ni la base de datos habían registrado nada del plano de datos, porque nadie lo activó. La causa se termina deduciendo por eliminación, sin poder demostrarla. Se activa por directiva, no a mano:

```
recursos con diagnóstico configurado    3 de 61   61 de 61
mecanismo                             manual    DeployIfNotExists
recursos nuevos                    sin cobertura  cubiertos al crearse
```

Y el registro de actividad se exporta a almacenamiento con inmutabilidad, porque los 90 días por defecto no cubren una auditoría anual.

Frente 2 — la factura de la telemetría, a los once días.

```
Usage
| where TimeGenerated > ago(7d) and IsBillable
| summarize GB = sum(Quantity)/1000 by DataType
| order by GB desc
```

```
AzureDiagnostics (pasarela, registros de acceso)   154 GB   48 %
ContainerLogStdout                                  98 GB   30 %
AppRequests + AppDependencies                       43 GB   13 %
resto                                               27 GB    9 %
                                                   ■■■■■■
total                                              322 GB en 7 días → ~46 GB/día
```

```
46 GB/día × 30 × 2,30 USD = 3.174 USD/mes solo de ingesta
```

Las dos primeras filas son el 78 % del gasto y nunca habían originado una alerta. Se actúa en los dos puntos, no en uno:

```
registros de acceso de la pasarela        22 GB/día     3 GB/día
  filtrado en la regla de recopilación:
  solo estado >= 400 o duración > 1 s
  el volumen completo se archiva en almacenamiento (clase 041)
salida estándar de contenedores           14 GB/día     5 GB/día en plan básico
  filtrado de líneas de depuración
telemetría de aplicación                   6 GB/día     6 GB/día  ← intacta
resto                                      4 GB/día     2 GB/día

ingesta en plan de análisis (11 GB/día)      759
ingesta en plan básico (5 GB/día)             97
archivo en almacenamiento                     14
                                             ■■■■■■
                                             870      frente a 3.174   (−73 %)
```

Lo que se conserva íntegro es lo que se usa para investigar. Lo que se abarata es lo que solo se lee cuando alguien va a buscarlo, y sigue estando.

Frente 3 — el panel que llevaba un mes mintiendo.

El panel de tráfico marcaba 41.000 peticiones diarias. La facturación del proveedor de pago no cuadraba con ese volumen.

```
requests | where timestamp > ago(1d)
| summarize registros = count(), reales = sum(itemCount)

registros  41.312
reales    413.120
```

Muestreo adaptativo al 10 %. Todos los paneles y las dos alertas de volumen usaban count().

```
agregación en paneles y alertas       count()         sum(itemCount)
muestreo para el flujo de pago       adaptativo       desactivado
muestreo para el resto               adaptativo       adaptativo (correcto)
alerta por caída de tráfico       nunca disparó    verificada con simulacro
```

La alerta por caída de tráfico era la peor de las tres: con count() y muestreo adaptativo, una subida real de tráfico podía parecer una bajada.

Frente 4 — la alerta que llegó nueve minutos tarde.

En la caída, la primera notificación llegó a las 09:31 para un incidente que empezó a las 09:22.

```
inicio real                         09:22
ingesta del registro                 09:24
frecuencia de evaluación (5 min)    09:29
ventana de 5 min ya cumplida         09:29
notificación                         09:31
```

Nueve minutos, todos estructurales. Se redistribuyen las alertas:

```
disponibilidad y 5xx de la pasarela      alerta de registro  alerta de métrica
latencia p95                             alerta de registro  alerta de métrica
excepción concreta con correlación         —                 alerta de registro
borrado de recursos                        —                 alerta de actividad
tiempo hasta la notificación             9 min               70 s
```

Frente 5 — las alertas que faltaban y ya se habían pagado dos veces.

Los incidentes de las clases 043 y 044 —el consumidor en cero réplicas y los 8.412 mensajes fallidos— tienen la misma forma: el sistema no se cayó, dejó de trabajar. Se añaden las cuatro señales de acumulación:

```
profundidad de la cola de pedidos creciendo 15 min
edad del mensaje más antiguo > 10 min
mensajes fallidos > 0 durante 15 min
retraso del consumidor de Event Hubs > 100.000 eventos
```

Las cuatro son métricas: cuestan céntimos y avisan en un minuto.

Y las cuatro consultas del manual de operación, guardadas antes de necesitarlas:

```
1. tasa de error por operación en la última hora
2. percentiles de latencia por operación, en serie temporal
3. traza completa por identificador de operación
```

```
4. cambios de configuración en la ventana del incidente, con autor
```

Resumen de la observabilidad:

```
recursos con registros activos                3 de 61         61 de 61
mecanismo de activación                       manual        por directiva
ingesta diaria                              46 GB/día       16 GB/día
costo mensual de telemetría                3.174 USD        870 USD
exactitud del recuento de peticiones          ×0,1            exacta
tiempo hasta la notificación                  9 min            70 s
alertas sobre trabajo acumulado                 0               4
consultas de incidente escritas de antemano     0               4
```

La lección que esta clase traslada al resto de la parte: la observabilidad de Azure no falla por falta de herramientas sino por dos valores por defecto: los registros de recurso vienen apagados y el muestreo viene encendido. El primero deja un incidente sin datos; el segundo deja los datos con un factor de diez. Ninguno de los dos avisa, y los dos se comprueban en un minuto — el día que se despliega, no el día del incidente.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-03-azure-core-platform/045-azure-monitor-log-analytics-y-application-insights/lab.
py
```

El laboratorio selecciona el motor de práctica observability y produce labresult.json. El escenario, sus comprobaciones y el artefacto esperado corresponden a esta clase; no requiere credenciales y deja explícito qué debe revalidarse en un sandbox real.

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa evidencia-operativa-azure en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es telemetría correlacionada con una pregunta operativa. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye evidencia-operativa-azure para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

Errores frecuentes

Síntoma	Causa probable	Corrección
Tras una caída no hay registros que analizar	Las configuraciones de diagnóstico están apagadas por defecto y nunca se crearon	Actívalas por directiva con DeployIfNotExists y verifica la cobertura con una consulta sobre los recursos sin ella.
Los paneles muestran una fracción del tráfico real	El muestreo adaptativo de Application Insights está activo y las consultas usan count()	Suma itemCount en todas las agregaciones y desactiva el muestreo solo para los flujos que exijan detalle completo.
La factura de telemetría crece más rápido que la plataforma	Todo entra en plan de análisis, incluidos los registros de acceso y la salida estándar de los contenedores	Filtra en la regla de recopilación antes de la ingesta y asigna plan básico a las tablas voluminosas que no originan alertas.
Las alertas llegan varios minutos después del inicio del incidente	Son alertas de registro: suman ingesta, frecuencia y ventana	Pasa a alertas de métrica todo lo que despierte a alguien y reserva las de registro para lo que solo se puede expresar consultando.
Una traza se corta al llegar a un servicio interno	Ese salto no propaga la cabecera traceparent	Propaga la cabecera en todos los clientes HTTP y verifica la continuidad con una consulta por operationId.
No se puede saber quién borró un recurso hace cuatro meses	El registro de actividad se conserva 90 días	Expórtalo con una configuración de diagnóstico a almacenamiento inmutable desde el primer día.

Seguridad, ética y costo

Trabaja con cuentas propias o sandboxes autorizados. No publiques secretos, identificadores reales ni datos personales. Antes de crear recursos pagos define presupuesto, etiquetas y comando de destrucción; después verifica que no queden recursos huérfanos. Los ejemplos locales enseñan contratos, pero no certifican cumplimiento ni disponibilidad de producción.

Preguntas de comprobación

1. ¿Qué origen de telemetría funciona sin configurar nada y cuál está apagado en todos los recursos?
2. ¿Con qué criterio decides si una señal debe ser métrica o registro, y cuánto cambia el costo?
3. ¿Por qué count() sobre la tabla requests puede mentir, y qué se usa en su lugar?
4. Descompón los nueve minutos de latencia de una alerta de registro. ¿Cuáles de ellos se pueden reducir?
5. ¿Qué cuatro alertas detectan un sistema que dejó de trabajar sin caerse, y por qué son baratas?

Referencias

- Microsoft (2025). Azure Monitor overview — métricas, registros, orígenes y destinos. <https://learn.microsoft.com/en-us/azure/azure-monitor/overview>
- Microsoft (2025). Diagnostic settings in Azure Monitor — activación por recurso, destinos y despliegue por directiva. <https://learn.microsoft.com/en-us/azure/azure-monitor/essentials/diagnostic-settings>
- Microsoft (2025). Log Analytics workspace table plans and cost optimization — planes por tabla, retención y archivo. <https://learn.microsoft.com/en-us/azure/azure-monitor/logs/data-platform-logs>
- Microsoft (2025). Sampling in Application Insights — muestreo adaptativo, itemCount y sus efectos en las consultas. <https://learn.microsoft.com/en-us/azure/azure-monitor/app/sampling>
- Microsoft (2025). Types of Azure Monitor alerts — métrica, registro y registro de actividad, con sus latencias. <https://learn.microsoft.com/en-us/azure/azure-monitor/alerts/alerts-types>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Evaluación — 045 Azure Monitor, Log Analytics y Application Insights

Preguntas

1. Explica el problema que resuelve esta capacidad sin mencionar un proveedor.
2. Identifica una frontera de responsabilidad y su propietario.
3. Compara dos alternativas mediante confiabilidad, seguridad, costo y operación.
4. ¿Qué demuestra el laboratorio y qué no puede demostrar?
5. Propón un caso límite o una prueba de fallo.

Reto verificable

Entrega evidencia-operativa-azure con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

- [ ] Resultado reproducible por una segunda persona.
- [ ] Evidencia vinculada a cada afirmación técnica.
- [ ] Prueba negativa o de fallo incluida.
- [ ] Costos y permisos expresados con unidades y alcance.
- [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.

046 — Key Vault, Defender for Cloud y Azure Policy

Parte: 03 — Azure: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: security · Estado: EXECUTABLECORE

Propósito

Aplicar defensa en profundidad a la plataforma Azure con las tres piezas que la sostienen, y con la precisión que más cuesta aprender: una directiva con efecto Deny no arregla lo que ya existe, porque actúa sobre la petición y no sobre el inventario. Asignarla y ver el panel en verde es el error de gobierno más repetido de esta parte. La clase 035 dejó el criterio —control de claves, rotación sin cortar conexiones, prueba negativa por control—; aquí cambian los mecanismos y aparece un motor de gobierno que no tiene equivalente directo en AWS.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir el nivel de Key Vault y el modelo de permisos, y explicar qué escalada cierra cada decisión.
2. Dimensionar el acceso a secretos para no agotar el límite de transacciones del almacén.
3. Distinguir los efectos de Azure Policy y saber cuál corrige lo existente y cuál solo guarda la puerta.
4. Rotar claves y secretos sin cortar conexiones, con el patrón de dos credenciales y una reacción a evento.
5. Verificar cada control con su prueba negativa, incluida la del antimalware y la de la directiva.

Conceptos centrales

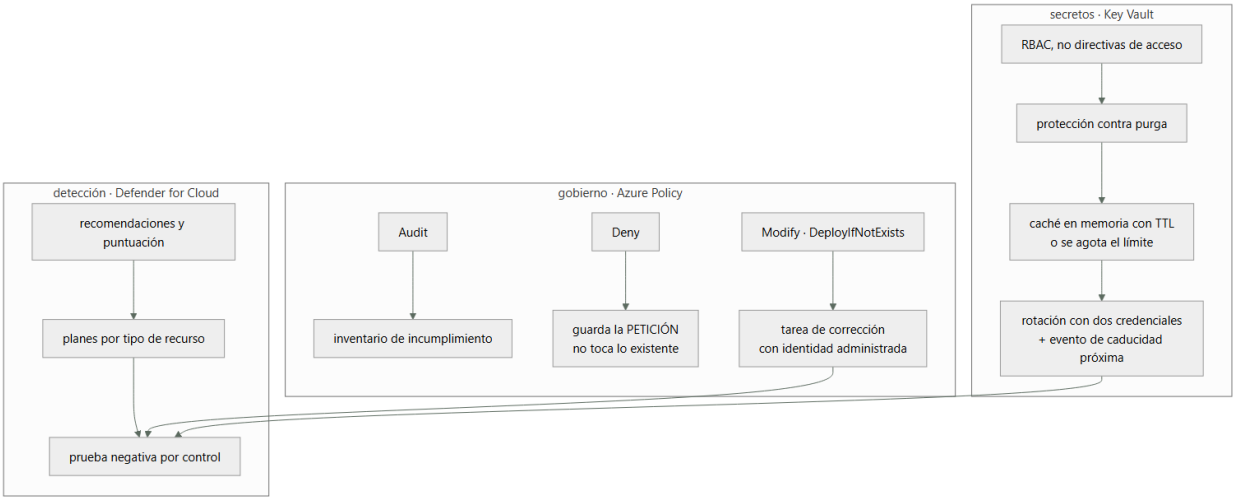
Concepto	Comprensión verificable
protección contra purga	Impide eliminar definitivamente un almacén o sus objetos durante el periodo de retención. Es irreversible al activarla, y sin ella quien puede borrar una clave puede destruir los datos que esa clave cifra.
modelo de permisos del almacén	Directivas de acceso —heredadas, del propio almacén— o Azure RBAC. Con las primeras, quien tiene Contributor sobre el almacén puede añadirse a sí mismo y leerlo todo.
clave gestionada por el cliente	Clave propia en Key Vault que cifra un servicio. Da control real y traslada una responsabilidad real: perder el acceso a la clave deja el recurso inaccesible.
efecto de directiva	Qué hace una directiva al evaluar: Audit informa, Deny rechaza la petición, Modify y DeployIfNotExists corrigen. Solo los dos últimos actúan sobre lo que ya existe, y mediante una tarea de corrección.
exención	Excepción explícita, con motivo, responsable y fecha de caducidad. Se distingue de excluir un ámbito en la asignación, que es invisible y no caduca.
puntuación de seguridad	Priorización relativa de las recomendaciones de Defender for Cloud. Es una herramienta para ordenar el trabajo, no un objetivo: llegar al 100 % no equivale a estar seguro.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Aplicado a esta clase, separa siempre cuatro planos: intención (qué necesita el usuario), configuración (qué declaramos), estado observado (qué existe de verdad) y evidencia (cómo sabemos que cumple). Confundirlos produce diseños que se ven correctos en un diagrama pero fallan al operar.

■ Flujo de razonamiento



Desarrollo

1. Key Vault: el modelo de permisos decide qué escalada existe

Antes de guardar el primer secreto hay dos decisiones que se toman una vez y se sufren siempre.

El nivel, que responde a un requisito, no a una preferencia:

```
estándar        claves protegidas por software
premium         claves respaldadas por HSM, módulo compartido
HSM administrado  módulo dedicado de un solo inquilino
                  ~3,20 USD/h ≈ 2.300 USD/mes
```

El último no se elige «por si acaso»: se elige cuando una obligación regulatoria exige aislamiento de inquilino y un nivel de certificación concreto. Su precio es un dato de arquitectura.

El modelo de permisos, que es donde está la escalada. Las directivas de acceso son el modelo original: una lista en el propio almacén que dice quién puede hacer qué. Su defecto es estructural:

```
quien tiene Contributor sobre el almacén
  → puede MODIFICAR la lista de directivas de acceso
  → puede añadirse a sí mismo con permisos de lectura de secretos
  → lee todos los secretos, y la operación es legítima en el registro
```

No hace falta ninguna vulnerabilidad: es cómo funciona. Con Azure RBAC eso se cierra, porque conceder acceso a los datos exige el rol correspondiente y conceder roles exige a su vez un permiso distinto —el de administrador de acceso de usuario—, que la clase 038 ya identificó como uno de los que deben ser elegibles y no permanentes.

```
$ az keyvault update -n kv-cloudshop -g rg-seg --enable-rbac-authorization true
$ az role assignment create --assignee $APP_ID --role "Key Vault Secrets User" \
    --scope "$KV_ID/secrets/db-password"
```

El ámbito de esa segunda línea importa: RBAC llega hasta el secreto concreto, mientras que una directiva de acceso solo llegaba al almacén entero. Es la diferencia entre «esta aplicación puede leer su contraseña» y «esta aplicación puede leer todas las contraseñas de la organización».

Y la tercera decisión, que solo se puede tomar en un sentido:

```
$ az keyvault update -n kv-cloudshop -g rg-seg --enable-purge-protection true
```

La eliminación temporal ya viene activa; la protección contra purga impide eliminar definitivamente durante el periodo de retención, y no se puede desactivar. El argumento que la justifica es directo: si el almacén guarda la clave que cifra una cuenta de almacenamiento, quien puede purgar esa clave puede destruir los datos sin tocarlos. Con protección contra purga, ese camino no existe.

Y el fallo operativo más frecuente de Key Vault no es de seguridad, es de límite de transacciones:

```
tope orientativo   ~2.000 transacciones de secreto por cada 10 s y almacén
aplicación que lee el secreto en CADA petición, a 900 peticiones/s
  → 9.000 transacciones cada 10 s → 429 Too Many Requests
  → que la aplicación devuelve como 500 al usuario
```

El síntoma engaña porque parece un fallo de la base de datos: la aplicación no consigue su contraseña y lo reporta como error de conexión. La corrección es la evidente y casi nunca está puesta:

```
leer el secreto al arrancar y guardarlo en memoria
refrescar con un TTL de minutos, no por petición
y refrescar además al recibir un error de autenticación
```

La última línea es la que permite rotar sin reiniciar: si la credencial falla, se relee del almacén antes de darse por vencido.

2. Claves propias: control real y responsabilidad real

La clase 035 estableció tres niveles de control sobre las claves. En Azure son estos:

	Quién gestiona	Costo	Qué obtienes
Clave de plataforma	Microsoft	Incluido	Cifrado en reposo, sin trabajo
Clave del cliente	Tú, en Key Vault	Almacén + operaciones	Control de rotación y de revocación
Clave en HSM administrado	Tú, módulo dedicado	2.300 USD/mes	Aislamiento de inquilino

La segunda fila es la que hay que entender bien, porque su ventaja y su riesgo son la misma propiedad: puedes revocar el acceso a la clave. Eso significa que puedes dejar un dato ilegible a voluntad, que es exactamente lo que se pide en una baja de servicio o en una respuesta a incidente. Y significa también que:

```
si borras la clave                → la cuenta de almacenamiento queda inaccesible
si retiras el permiso a la identidad → el servicio deja de arrancar
si el almacén queda inalcanzable por red → lo mismo, sin haber tocado la clave
```

No es un fallo del diseño: es el precio del control, y hay que aceptarlo por escrito antes de activarlo. Las tres protecciones que lo hacen sostenible son la protección contra purga, un bloqueo de recurso sobre el almacén y una identidad asignada por el usuario —de la clase 038— para que la relación con la clave sobreviva a recrear el recurso.

Sobre la rotación, la precisión de la clase 035 se mantiene entera: rota la clave, no lo ya cifrado. Y en Azure hay un detalle comprobable que decide si la rotación sirve de algo:

```
# referencia SIN versión: el servicio toma automáticamente la versión nueva
https://kv-cloudshop.vault.azure.net/keys/clave-almacenamiento

# referencia CON versión: la rotación no cambia nada, sigue usando la vieja
https://kv-cloudshop.vault.azure.net/keys/clave-almacenamiento/7f3a…
```

Una rotación automática configurada sobre una referencia con versión fija se ejecuta puntualmente cada 90 días, genera una versión nueva, y el recurso sigue cifrando con la de siempre. El panel dice que la rotación está activa; la comprobación honesta es preguntar qué versión está usando el recurso.

La rotación de secretos sin cortar conexiones es el patrón de dos credenciales, y aquí encaja con lo construido en la clase 044:

```
1. existen dos credenciales activas: primaria y secundaria
2. las aplicaciones usan la primaria
3. se rota la SECUNDARIA (nadie la está usando)
4. se promueve: las aplicaciones pasan a la secundaria ya rotada
5. se rota la antigua primaria, que ahora está libre
```

En ningún momento hay una ventana sin credencial válida. Y el disparador no tiene por qué ser un calendario: Key Vault emite un evento cuando un secreto se acerca a su caducidad.

```
SecretNearExpiry → Event Grid → función que ejecuta la rotación
```

Es la reacción a hechos de la plataforma que la clase 044 describía, aplicada al caso donde más rinde: la rotación deja de depender de que alguien se acuerde.

3. Azure Policy: Deny guarda la puerta y no limpia la casa

Este es el mecanismo de gobierno que no tiene equivalente directo en AWS, y su malentendido más caro cabe en una línea: Deny evalúa la petición, no el inventario.

```
efecto              qué hace                        ¿toca lo que ya existe?
Audit               marca incumplimiento             informa, no cambia
Deny                rechaza la creación o el cambio  NO
```

```
Modify              añade o cambia propiedades       sí, con tarea de corrección
DeployIfNotExists   despliega un recurso asociado    sí, con tarea de corrección
AuditIfNotExists    marca la falta de algo asociado  informa
Disabled            desactiva la regla               —
```

Asignar una directiva Deny de «las cuentas de almacenamiento no deben permitir acceso público» impide crear una nueva mal configurada y deja intactas las catorce que ya lo están. El panel puede mostrar que las creaciones recientes cumplen mientras el problema real sigue ahí. Corregir lo existente exige otro efecto y un paso explícito:

```
$ az policy remediation create --name corregir-red-almacenamiento \
    --policy-assignment $ASIGNACION_ID --resource-discovery-mode ReEvaluateCompliance
```

Y esa tarea necesita una identidad administrada con permiso suficiente para hacer el cambio, lo que a su vez es una decisión de seguridad: se le concede el rol mínimo, no Contributor sobre la suscripción.

Dos comportamientos temporales que evitan diagnósticos erróneos:

```
al crear o modificar un recurso   la directiva se evalúa al instante
recursos existentes               se reevalúan cada ~24 h
```

Un panel de cumplimiento puede estar un día desfasado. Después de corregir algo, se fuerza la evaluación en vez de concluir que la corrección no funcionó.

Las iniciativas agrupan directivas con parámetros comunes, y las integradas cubren marcos completos —ISO 27001, PCI DSS, CIS—. Empezar por una integrada en modo Audit produce en un día el inventario que de otro modo cuesta semanas, y sin bloquear nada. La secuencia sensata es siempre la misma:

```
1. Audit          descubre el tamaño real del problema
2. corregir       con tareas de corrección donde el efecto lo permita
3. Deny           solo cuando el inventario ya cumple
```

Hacerlo al revés —Deny primero— bloquea despliegues legítimos, produce una avalancha de excepciones y termina con la directiva desasignada «temporalmente».

Y sobre las excepciones, una distinción de gobierno que separa una plataforma auditable de una que aparenta serlo:

```
exclusión de ámbito en la asignación
  invisible en los paneles, sin motivo, sin fecha, sin responsable
exención
  objeto propio, con categoría, justificación, responsable y CADUCIDAD
```

Las dos consiguen que un recurso deje de incumplir. Solo una de ellas se puede revisar. Una plataforma con cero incumplimientos y treinta exclusiones no documentadas está peor que una con treinta incumplimientos conocidos.

Un último enlace con la clase 038: cuando una directiva bloquea algo, el error no es de autorización, y leerlo ahorra buscar en el sitio equivocado:

```
AuthorizationFailed          falta el rol
RequestDisallowedByPolicy    hay rol y la directiva lo impide
```

4. Defender for Cloud: dos mitades y una puntuación que no es la meta

Defender for Cloud hace dos cosas distintas que se facturan distinto:

```
gestión de la postura (CSPM)   recomendaciones, puntuación de seguridad,
                               cumplimiento normativo
                               nivel gratuito incluido
protección de cargas (CWP)     detección de amenazas por tipo de recurso
                               planes de pago, uno por tipo
```

La segunda se activa plan a plan, y ahí está la decisión de costo. Encender todo en toda la suscripción es cómodo y caro; encender lo que corresponde al inventario real es trabajo de una hora:

```
Defender for Servers P2      ~15 USD por servidor y mes
Defender for Storage         ~10 USD por cuenta y mes (+ análisis por GB)
Defender for SQL             ~15 USD por instancia y mes
Defender for Containers      ~7 USD por vCPU y mes
Defender for Key Vault       por transacciones
```

La puntuación de seguridad ordena el trabajo por impacto relativo, y conviene decir con claridad qué no es: no es un objetivo. Subirla del 68 % al 74 % cerrando veinte recomendaciones de bajo impacto consume el mismo tiempo que cerrar la única que permitía movimiento lateral, y no reduce el mismo riesgo. La puntuación sirve para priorizar, y la priorización se corrige con criterio propio: qué explota un atacante primero en esta plataforma concreta.

El panel de cumplimiento normativo merece una nota, porque explica la arquitectura del producto: está construido sobre iniciativas de Azure Policy. Defender y Policy no son dos herramientas, son la misma evaluación presentada de dos formas. Lo que se corrige en una aparece en la otra.

Dos capacidades que enlazan con clases anteriores y rinden más de lo que cuestan:

Acceso justo a tiempo a máquinas virtuales. Los puertos de administración permanecen cerrados en el NSG y se abren, para una IP concreta y durante un plazo, previa solicitud registrada. Es exactamente el modelo de privilegio mínimo en el tiempo de la clase 038, aplicado a la red en lugar de a los roles. Y la aritmética es la misma: un puerto abierto 3 horas al mes en vez de 720 reduce la superficie un 99,6 %.

Análisis antimalware en almacenamiento. Analiza cada blob al subirlo. Tiene un costo por GB analizado, así que se activa en los contenedores que reciben ficheros de usuarios y no en los que guardan telemetría.

Y aquí vuelve, con toda su fuerza, la regla que cerraba la clase 035: cada control necesita su prueba negativa. Activar un plan de Defender no demuestra nada; lo que lo demuestra es provocar la detección:

```
# fichero de prueba estándar EICAR — inofensivo, reconocido por todo antimalware
$ printf 'X5O!P%%@AP[4\\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*' \
    > eicar.txt
$ az storage blob upload -c cargas -f eicar.txt -n eicar.txt \
    --account-name stcloudshopprod --auth-mode login
# la alerta debe aparecer en Defender en minutos
```

Si no aparece, el control no está donde se creía —muy a menudo porque el plan se activó en la suscripción y esa cuenta de almacenamiento estaba excluida—. Y esa comprobación hay que repetirla, porque un control verificado una vez es un control verificado en el pasado.

5. La prueba negativa de cada control de esta parte

La clase 035 dejó el principio y la clase 036 lo aplicó a AWS. Cerrar la parte 03 exige la lista equivalente, porque un control sin prueba negativa es una afirmación, no un control.

control	prueba negativa que lo demuestra
acceso público del almacenamiento deshabilitado	petición desde fuera → 403 (clase 041)
punto de conexión privado activo	nslookup desde la subred devuelve 10.x y desde fuera falla (clase 039)
salida de la subred cerrada	curl con tiempo límite → agota (clase 039)
clave compartida deshabilitada	autenticación con la clave → KeyBasedAuthenticationNotPermitted (041)
federación acotada por sujeto	petición desde otro repositorio → AADSTS70021 (clase 038)
directiva Deny asignada	crear el recurso prohibido → RequestDisallowedByPolicy
protección contra purga	intentar purgar → operación rechazada
antimalware en almacenamiento	subir el fichero EICAR → alerta
recuperación de contenedor	borrar y restaurar uno de prueba (041)
restauración de base de datos	restaurar a una copia y contar filas (042)
rotación de secreto	rotar con tráfico en curso y medir errores: deben ser cero (035)

La columna derecha es el entregable. Un informe de seguridad que dice «se ha habilitado X» no es comparable a uno que dice «se ha habilitado X y aquí está el error que devuelve al intentar lo que impide».

Y hay una prueba negativa de segundo orden que casi nunca se hace y descubre lo que las demás no: comprobar que el control sigue activo pasado un tiempo. Los controles se apagan solos por caminos perfectamente normales —alguien recrea el recurso desde una plantilla vieja, un despliegue sobrescribe la configuración, una excepción temporal se queda—. La defensa contra eso no es la memoria de nadie:

```
directiva en modo Audit sobre cada control crítico
  → el panel muestra cualquier recurso que se salga
  → y la desviación aparece en horas, no en la siguiente auditoría
```

Es el mismo principio que cierra toda la parte: lo que no tiene una señal automática, se degrada en silencio. Los registros apagados de la clase 045, el consumidor en cero réplicas de la 043, la subcola de fallidos de la 044 y el control de seguridad revertido son el mismo problema con cuatro caras: sistemas que dejan de hacer su trabajo sin producir ningún error.

Ejemplo trabajado

CloudShop cierra la seguridad de su plataforma Azure. El equipo activa Key Vault, asigna una iniciativa de cumplimiento y enciende Defender en toda la suscripción. A las seis semanas hay un panel en verde, una factura alta y cinco cosas que no eran ciertas.

Hecho 1 — la aplicación devuelve 500 en los picos y la culpa parecía de la base de datos.

```
09:40  errores 500 al 6 % · el registro dice "no se pudo conectar a la base de datos"
09:41  la base de datos está sana: CPU 22 %, ninguna conexión rechazada
```

```
AzureDiagnostics
| where ResourceProvider == "MICROSOFT.KEYVAULT" and httpStatusCode_d == 429
| summarize limitadas = count() by bin(TimeGenerated, 1m)
```

```
09:38   4.117
09:39  11.902
09:40  12.455
```

La aplicación leía la contraseña de Key Vault en cada petición. A 900 peticiones por segundo, el almacén limitaba, y el fallo se reportaba como error de base de datos porque es donde se manifestaba.

```
lecturas del almacén            1 por petición   1 cada 30 min
transacciones en el pico        ~9.000/10 s        ~2/10 s
relectura ante fallo de auth        no               sí
errores 500 en el pico             6,0 %            0,0 %
```

Hecho 2 — un ingeniero leyó todos los secretos y todo estaba permitido.

Una revisión rutinaria encuentra una modificación de la lista de directivas de acceso seguida de cuarenta lecturas de secretos, todas correctas.

```
AzureActivity
| where _ResourceId has "kv-cloudshop"
| where OperationNameValue endswith "VAULTS/WRITE"
| project TimeGenerated, Caller, ActivityStatusValue
```

No hubo intrusión: con Contributor sobre el almacén, añadirse a la lista es una operación legítima. Se cierra el camino:

```
modelo de permisos          directivas de acceso     Azure RBAC
ámbito de la aplicación       almacén entero        un secreto concreto
quién puede conceder acceso  cualquier Contributor  administrador de acceso
                                                    de usuario, y elegible (038)
protección contra purga           no                    sí
alerta sobre cambios de rol       no                    sí, alerta de actividad
```

Hecho 3 — el certificado se renovó y el sitio siguió sirviendo el viejo.

A los cuatro días de la renovación automática, los clientes empiezan a ver avisos de certificado caducado.

```
$ az keyvault certificate show --vault-name kv-cloudshop -n cloudshop-tls \
    --query "attributes.expires" -o tsv
2027-07-30T00:00:00Z                      # renovado, correcto
$ echo | openssl s_client -connect cloudshop.example:443 2>/dev/null \
    | openssl x509 -noout -enddate
notAfter=Jul 31 09:00:00 2026 GMT         # lo que ve el cliente: el viejo
```

El enlace de la aplicación apuntaba a una versión concreta del certificado. La renovación creó otra versión y nadie la usaba.

```
referencia al certificado      versión fija       sin versión
vigilancia del vencimiento     desde el almacén   desde FUERA, con una sonda TLS
aviso                          ninguno            30 días antes, sobre lo servido
```

La segunda fila es la lección: el vencimiento se vigila donde lo ve el cliente, no donde se guarda. El almacén decía la verdad y era irrelevante.

Hecho 4 — cumplimiento en verde con catorce cuentas públicas.

```
$ az policy state summarize --query "value[0].results.{cumple:nonCompliantResources}" -o tsv
0
$ az storage account list --query "[?allowBlobPublicAccess].name" -o tsv | wc -l
14
```

La contradicción se explica leyendo la asignación: efecto Deny, que solo evalúa peticiones nuevas, y un ámbito con tres exclusiones heredadas de la puesta en marcha, sin motivo ni fecha.

```
efecto de la directiva                    Deny        Audit → corregir → Deny
cuentas con acceso público                 14                 0
mecanismo de corrección                 ninguno      tarea de corrección
exclusiones de ámbito                       3                 0
exenciones documentadas                     0        2, con motivo y caducidad
```

Las dos exenciones que quedan son reales —un contenedor de recursos estáticos públicos por diseño— y ahora caducan en noventa días, así que alguien tendrá que volver a justificarlas.

Hecho 5 — 3.100 USD al mes de Defender y un control que no estaba donde se creía.

```
Defender for Servers P2                todo          22 servidores    330
Defender for Storage                   todo           3 cuentas         30
Defender for SQL                       todo           2 instancias      30
Defender for Containers                todo          16 vCPU           112
Defender for App Service, DNS, Resource Manager, APIs…                 2.598
                                                                      ■■■■■■
                                                                       3.100
```

Se conservan los planes que corresponden al inventario y a la superficie real de ataque, y se retiran los que protegían servicios que CloudShop no usa:

```
planes conservados (servidores, almacenamiento, SQL,
contenedores, Key Vault, Resource Manager)                ~640
```

Y la prueba negativa descubre lo importante:

```
$ az storage blob upload -c cargas -f eicar.txt -n eicar.txt \
    --account-name stcloudshopcargas --auth-mode login
# 20 minutos después: ninguna alerta
```

La cuenta que recibe los ficheros de los clientes —la única que de verdad importaba— se había creado después de activar el plan y estaba en otra suscripción sin cobertura. Se corrige y se repite la prueba:

```
subida del fichero EICAR → alerta "Malicious file uploaded" en 4 min      ✓
```

Resumen del cierre de seguridad:

```
modelo de permisos del almacén        directivas de acceso     RBAC
protección contra purga                       no                sí
errores 500 por limitación del almacén     6,0 %               0,0 %
cuentas de almacenamiento públicas           14                  0
exclusiones de ámbito sin justificar          3                  0
exenciones con motivo y caducidad             0                  2
planes de Defender activos                todos (3.100 USD)   los del inventario (640)
controles con prueba negativa ejecutada     0 de 11             11 de 11
```

La lección que esta clase traslada al proyecto de la clase 048: el panel en verde y el control efectivo son dos cosas distintas, y solo una de ellas se puede demostrar. Deny guardaba la puerta mientras catorce puertas ya estaban abiertas; el plan de antimalware protegía todo menos la cuenta que recibía ficheros de desconocidos. La prueba negativa no es la parte opcional del trabajo: es la única parte que produce evidencia.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-03-azure-core-platform/046-key-vault-defender-for-cloud-y-azure-policy/lab.py
```

El laboratorio selecciona el motor de práctica security y produce labresult.json. El escenario, sus comprobaciones y el artefacto esperado corresponden a esta clase; no requiere credenciales y deja explícito qué debe revalidarse en un sandbox real.

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa controles-azure en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es un control con amenaza, mitigación y evidencia. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye controles-azure para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
La aplicación falla al conectar con la base de datos y la base de datos está sana	Se lee el secreto de Key Vault en cada petición y el almacén limita con 429	Cachea el secreto en memoria con TTL y reléelo solo al recibir un error de autenticación.
Alguien con permisos administrativos sobre el almacén lee todos los secretos y todo aparece como legítimo	El modelo de directivas de acceso permite modificar la propia lista	Cambia a Azure RBAC, acota el ámbito al secreto y haz elegible el rol que concede acceso.
El certificado se renueva en el almacén y el cliente sigue viendo el caducado	El enlace apunta a una versión concreta, no al certificado	Referencia sin versión y vigila el vencimiento con una sonda TLS desde fuera, no desde el almacén.
El panel de cumplimiento está en verde y hay recursos incumpliendo	El efecto Deny solo evalúa peticiones nuevas, y había exclusiones de ámbito sin documentar	Empieza en Audit, corrige lo existente con tareas de corrección, y sustituye exclusiones por exenciones con motivo y caducidad.
La rotación automática de la clave se ejecuta y el recurso sigue usando la versión antigua	El recurso referencia una versión concreta en vez de la clave sin versión	Usa la referencia sin versión y comprueba qué versión está usando realmente el recurso.
Un plan de Defender está activo y no detecta nada al provocarlo	El recurso que importaba estaba fuera del ámbito donde se activó el plan	Ejecuta la prueba negativa —el fichero EICAR— contra el recurso real y repítela periódicamente.

■ Seguridad, ética y costo

Trabaja con cuentas propias o sandboxes autorizados. No publiques secretos, identificadores reales ni datos personales. Antes de crear recursos pagos define presupuesto, etiquetas y comando de destrucción; después verifica que no queden recursos huérfanos. Los ejemplos locales enseñan contratos, pero no certifican cumplimiento ni disponibilidad de producción.

■ Preguntas de comprobación

1. ¿Qué escalada permite el modelo de directivas de acceso y cómo la cierra Azure RBAC?
2. ¿Por qué una directiva con efecto Deny puede convivir con catorce recursos incumpliendo?
3. ¿Qué diferencia hay entre excluir un ámbito y crear una exención, y por qué importa para una auditoría?
4. ¿Qué ganas y qué asumes al cifrar con una clave propia en vez de con la de la plataforma?
5. Enumera tres controles de esta parte y la prueba negativa concreta que demuestra cada uno.

Referencias

• Microsoft (2025). Azure Key Vault security overview — modelos de permisos, protección contra purga y límites de servicio. <https://learn.microsoft.com/en-us/azure/key-vault/general/security-features>

• Microsoft (2025). Customer-managed keys — control, revocación y consecuencias de perder el acceso a la clave. <https://learn.microsoft.com/en-us/azure/storage/common/customer-managed-keys-overview>

• Microsoft (2025). Understand Azure Policy effects — Audit, Deny, Modify, DeployIfNotExists y su alcance. <https://learn.microsoft.com/en-us/azure/governance/policy/concepts/effects>

• Microsoft (2025). Azure Policy exemption structure — exenciones con motivo, responsable y caducidad. <https://learn.microsoft.com/en-us/azure/governance/policy/concepts/exemption-structure>

• Microsoft (2025). Microsoft Defender for Cloud plans — postura frente a protección de cargas y precios por tipo de recurso. <https://learn.microsoft.com/en-us/azure/defender-for-cloud/defender-for-cloud-introduction>

• Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 03 en PDF · Recorrido de Azure en PDF · Manual integral

Anterior	Índice	Siguiente
← 045 · Azure Monitor, Log Analytics y Application Insights	Parte 03 · Programa	047 · Bicep, plantillas y despliegues por alcance →

Evaluación — 046 Key Vault, Defender for Cloud y Azure Policy

Preguntas

1. Explica el problema que resuelve esta capacidad sin mencionar un proveedor.
2. Identifica una frontera de responsabilidad y su propietario.
3. Compara dos alternativas mediante confiabilidad, seguridad, costo y operación.
4. ¿Qué demuestra el laboratorio y qué no puede demostrar?
5. Propón un caso límite o una prueba de fallo.

Reto verificable

Entrega controles-azure con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

• [ ] Resultado reproducible por una segunda persona.

• [ ] Evidencia vinculada a cada afirmación técnica.

• [ ] Prueba negativa o de fallo incluida.

- [ ] Costos y permisos expresados con unidades y alcance.
- [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.

047 — Bicep, plantillas y despliegues por alcance

Parte: 03 — Azure: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: iac · Estado: EXECUTABLECORE

Propósito

Declarar la infraestructura de Azure con Bicep entendiendo lo que lo hace distinto de todo lo que viene después en la parte 07: no hay archivo de estado, porque el estado deseado se compara contra el propio Azure. Eso elimina una clase entera de problemas y crea otra, y en medio está el alcance del despliegue —grupo de recursos, suscripción, grupo de administración o inquilino—, que es la decisión que más plantillas rompe al empezar.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir el alcance de un despliegue y saber qué recursos solo pueden crearse en cada uno.
2. Distinguir el modo incremental del completo y explicar exactamente qué borra el segundo.
3. Leer la salida de what-if sabiendo qué diferencias son reales y cuáles son ruido del proveedor.
4. Evitar la desaparición de recursos hijo al mezclar declaración en línea y declaración independiente.
5. Manejar secretos en plantillas sin dejarlos en el historial de despliegues.

Conceptos centrales

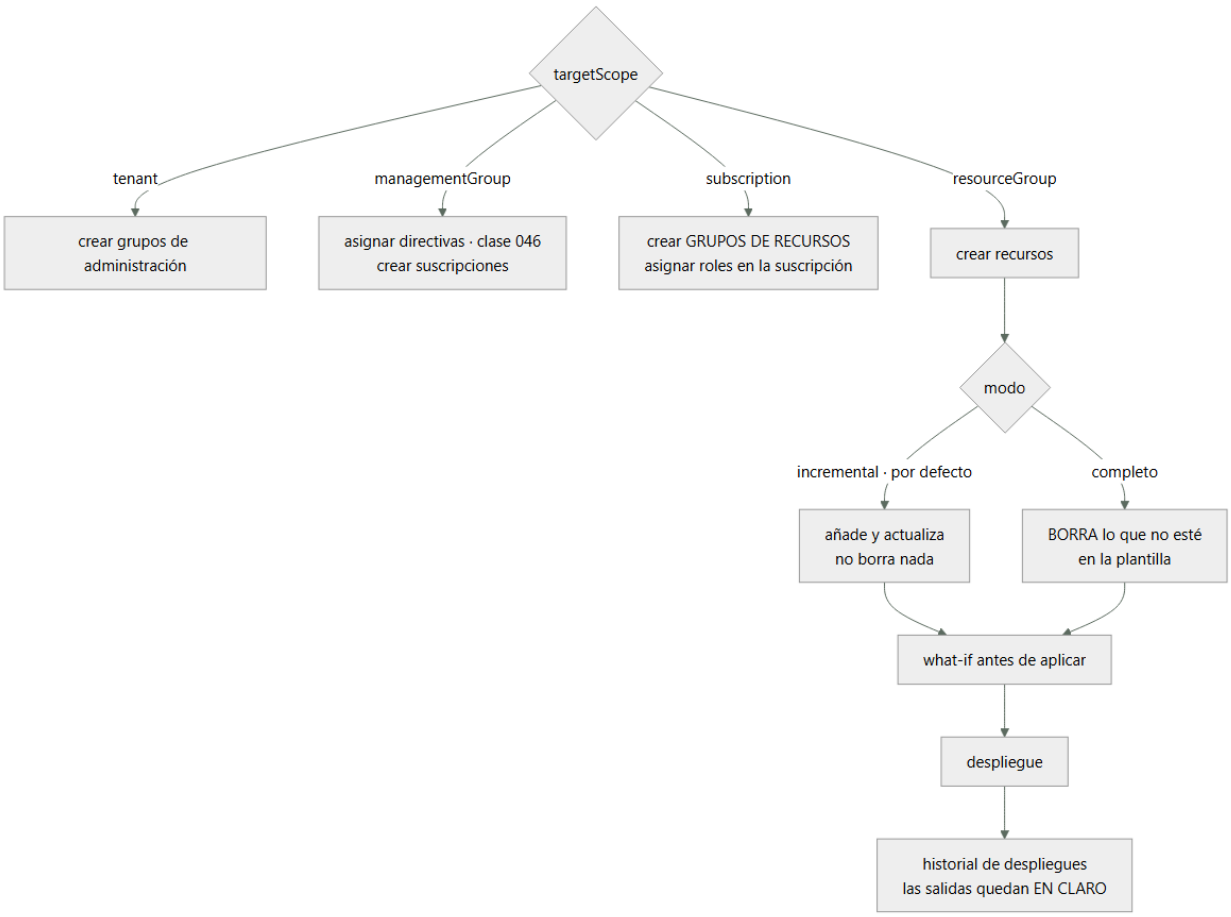
Concepto	Comprensión verificable
alcance de despliegue	Nivel en el que se ejecuta la plantilla: resourceGroup, subscription, managementGroup o tenant. Determina qué se puede crear: un grupo de recursos solo se crea desde el alcance de suscripción.
modo completo	Modo de despliegue que elimina del grupo de recursos todo lo que no aparece en la plantilla. No avisa por separado: es parte de la operación.
what-if	Simulación del cambio antes de aplicarlo. Es imprescindible y no es infalible: algunos proveedores informan de modificaciones que no ocurren, y ese ruido enseña a ignorarla.
recurso hijo en línea o independiente	Las reglas de un NSG, los ajustes de una aplicación o las entradas de DNS pueden declararse dentro del padre o como recurso propio. Mezclar ambas formas hace desaparecer las que no están en la plantilla que se despliega.
salida de despliegue	Valor devuelto por la plantilla. Se guarda en el historial de despliegues en claro y lo lee cualquiera con permiso de lectura: nunca debe contener un secreto.
pila de despliegue	Agrupación de recursos gestionada como una unidad, con control de ciclo de vida y ajustes de denegación. Es la respuesta moderna a la brusquedad del modo completo.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Aplicado a esta clase, separa siempre cuatro planos: intención (qué necesita el usuario), configuración (qué declaramos), estado observado (qué existe de verdad) y evidencia (cómo sabemos que cumple). Confundirlos produce diseños que se ven correctos en un diagrama pero fallan al operar.

■ Flujo de razonamiento



Desarrollo

1. Sin archivo de estado: qué se gana y qué se pierde

Bicep es una sintaxis que compila a la plantilla JSON del Azure Resource Manager. La correspondencia es uno a uno: no hay nada que Bicep pueda expresar que ARM no acepte, ni al revés. Eso lo convierte en una abstracción transparente, y explica por qué se puede adoptar sin apuesta: se puede volver al JSON en cualquier momento, y el resultado de compilarlo es exactamente lo que se despliega.

La diferencia estructural con la herramienta de la parte 07 está en otro sitio:

```
Terraform   mantiene un archivo de ESTADO con lo que cree que existe
Bicep/ARM   no mantiene estado: pregunta a Azure qué hay ahora mismo
```

Lo que se gana con eso es una lista concreta de problemas que dejan de existir:

```
no hay estado que bloquear entre dos despliegues simultáneos
no hay estado que se corrompa ni que haya que migrar
no hay secretos guardados en el estado
no hay que importar un recurso creado a mano: la próxima ejecución lo ve
```

Ese último punto es más importante de lo que parece. Un recurso creado desde el portal no rompe el siguiente despliegue: la plantilla declara lo que quiere y Azure concilia. Un recurso creado a mano fuera de una herramienta con estado, en cambio, exige un paso de importación explícito.

Y lo que se pierde también es concreto:

```
no hay una lista de "lo que este código gestiona"
  → nadie sabe qué recursos pertenecen a esta plantilla y cuáles no
no se detecta que alguien BORRÓ algo
  → la plantilla lo vuelve a crear en el siguiente despliegue,
    sin decir que había desaparecido
no hay destrucción ordenada del conjunto
  → borrar lo desplegado es borrar el grupo de recursos, o usar modo completo
```

La primera pérdida es la que causa incidentes. Sin una lista de pertenencia, la única forma de preguntar «¿qué gestiona esta plantilla?» es leerla, y la única forma de limpiar lo que sobra es el modo completo, que es un instrumento romo. Las pilas de despliegue existen precisamente para cubrir ese hueco: agrupan los recursos de un despliegue como una unidad con ciclo de vida propio y ajustes que impiden modificarlos fuera de la pila. Es lo más parecido a una lista de pertenencia que ARM ofrece, y conviene usarlas donde el conjunto tiene identidad —un entorno, un servicio completo—.

Una consecuencia práctica que conviene tener presente desde el principio: la unidad natural de aislamiento en Azure es el grupo de recursos (clase 037), y también lo es aquí. Un grupo de recursos por unidad desplegable evita casi todos los problemas que siguen, porque hace que la pregunta «qué pertenece a esta plantilla» tenga una respuesta obvia.

2. El alcance decide qué se puede crear

Es lo primero que rompe una plantilla al empezar, y el mensaje de error rara vez lo dice con claridad. Cada plantilla declara en qué nivel se ejecuta:

```
targetScope = 'subscription'
```

Y el nivel determina qué recursos son válidos:

Alcance	Se puede crear
tenant	Grupos de administración, asignaciones en la raíz
managementGroup	Asignaciones de directiva e iniciativas, suscripciones, asignaciones de rol
subscription	Grupos de recursos, directivas y roles de la suscripción
resourceGroup	Todo lo demás: la infraestructura propiamente dicha

El caso que aparece siempre: un grupo de recursos no se puede crear desde una plantilla de alcance resourceGroup, porque el grupo tendría que existir ya para poder ejecutarla. La estructura correcta es una plantilla de suscripción que cree los grupos y llame a módulos dentro de cada uno:

```
targetScope = 'subscription'

param entorno string
param ubicacion string = 'westeurope'

resource rgRed 'Microsoft.Resources/resourceGroups@2024-03-01' = {
  name: 'rg-cloudshop-red-${entorno}'
  location: ubicacion
}

module red 'modulos/red.bicep' = {
  name: 'despliegue-red'
  scope: rgRed                      // el módulo se ejecuta DENTRO del grupo
  params: { entorno: entorno, ubicacion: ubicacion }
}
```

La palabra clave es scope en el módulo: permite que una plantilla de un nivel despliegue en otro, que es como se construye una plataforma completa desde una sola ejecución. Y para referirse a algo que ya existe en otro sitio, sin crearlo:

```
resource almacen 'Microsoft.KeyVault/vaults@2023-07-01' existing = {
  name: 'kv-cloudshop'
  scope: resourceGroup('rg-cloudshop-seg-prod')
}
```

existing no crea ni modifica nada: solo obtiene una referencia. Es la vía correcta para leer un identificador o un secreto de un recurso gestionado por otro equipo, y evita el antipatrón de pasar identificadores completos como parámetros de texto, que se rompen en cuanto algo cambia de nombre.

Dos precisiones que ahorran depuración:

Los nombres de despliegue deben ser únicos por alcance y se conservan. Reutilizar el mismo nombre sobrescribe la entrada del historial, que es justo lo que no se quiere cuando hay que investigar qué se desplegó cuando. Un nombre con marca de tiempo o con el identificador de la ejecución de la canalización resuelve el problema.

Las funciones dependen del alcance. resourceGroup() no existe en una plantilla de suscripción, y subscription() sí. Un error de «función no reconocida» casi siempre significa que la plantilla está en el nivel equivocado, no que la función esté mal escrita.

3. Modo completo y recursos hijo: las dos formas de borrar sin querer

El modo de despliegue tiene dos valores y una asimetría de consecuencias:

```
incremental (por defecto)  añade lo que falta, actualiza lo que cambió,
                           NO toca lo que no aparece en la plantilla
completo                   además, ELIMINA del grupo de recursos
                           todo lo que no esté en la plantilla
```

El modo completo tiene un uso legítimo: garantizar que un entorno contiene exactamente lo declarado, sin residuos. Y tiene un riesgo que se materializa cuando el grupo de recursos contiene algo que otra persona creó por una razón válida — un disco de diagnóstico, una instantánea de una migración, un recurso de otro equipo. El despliegue no pregunta.

La protección no es la prudencia sino el diseño:

```
1. un grupo de recursos por unidad desplegable
   → nada ajeno puede estar dentro
2. `what-if` obligatorio antes de aplicar en modo completo
   → las eliminaciones aparecen marcadas y hay que leerlas
3. bloqueo de recurso sobre lo que jamás debe borrarse (clase 042)
   → un bloqueo hace fallar el despliegue en vez de perder el recurso
```

La segunda forma de borrar sin querer es más sutil y no depende del modo. Muchos recursos de Azure admiten declarar sus hijos de dos maneras:

```
// (a) en línea, dentro del padre
resource nsg 'Microsoft.Network/networkSecurityGroups@2024-01-01' = {
  name: 'nsg-app'
  location: ubicacion
  properties: {
    securityRules: [ { name: 'allow-lb-probe', properties: { /* … */ } } ]
  }
}

// (b) como recurso independiente
resource regla 'Microsoft.Network/networkSecurityGroups/securityRules@2024-01-01' = {
  parent: nsg
  name: 'allow-appgw-8080'
  properties: { /* … */ }
}
```

Cada forma es correcta por separado. Mezclarlas destruye datos: la declaración en línea es la lista completa de reglas, así que al desplegar el padre, Azure sustituye el conjunto entero por lo que hay en línea y las reglas declaradas aparte desaparecen. En el siguiente despliegue de la plantilla que las declaraba, vuelven. El resultado es un juego de reglas que aparecen y desaparecen según qué canalización se ejecutó la última, con incidentes de conectividad intermitentes de causa muy difícil de ver.

La regla de equipo es de una línea: para cada tipo de recurso hijo, se elige una de las dos formas y se documenta. Ocurre igual con los ajustes de aplicación de App Service, las reglas de firewall de una base de datos y los registros de una zona DNS.

Y el mismo mecanismo explica otra sorpresa habitual: un despliegue que «borra» la configuración que alguien puso a mano en el portal. No la borra por malicia — la plantilla declara el estado completo de esa propiedad, y lo que no está declarado vuelve a su valor por omisión. Es el comportamiento correcto de una herramienta declarativa, y es la razón por la que la configuración manual y la declarada no pueden convivir sobre el mismo recurso.

4. what-if, validación y el ruido que enseña a no mirar

what-if compara el estado deseado con el actual y muestra el cambio antes de aplicarlo:

```
$ az deployment group what-if -g rg-cloudshop-app-prod \
    --template-file main.bicep --parameters @prod.bicepparam

Resource and property changes are indicated with these symbols:
  + Create   ~ Modify   - Delete   = NoChange
```

```
~ Microsoft.Web/sites/app-tienda
  ~ properties.siteConfig.alwaysOn:  false → true
+ Microsoft.Insights/components/appi-tienda
```

Es obligatorio antes de cualquier aplicación y especialmente antes del modo completo, donde las líneas con - son las que hay que leer con atención.

Y tiene una limitación que hay que conocer para que no la desactive de hecho: algunos proveedores informan de modificaciones que no van a ocurrir. Devuelven propiedades calculadas o normalizan valores, y what-if las presenta como diferencias. El resultado predecible es que una salida con treinta líneas de ruido en cada ejecución enseña al equipo a hojearla, y el día que aparece un cambio real pasa desapercibido.

Las medidas que lo mantienen legible:

```
módulos pequeños       una salida corta se lee; una de 300 líneas se hojea
lista de exclusiones   `--exclude-change-types NoChange Ignore`
tipos ruidosos aparte  desplegar por separado lo que siempre informa cambios
revisar el ruido       si una propiedad aparece siempre, o falta en la plantilla
                       o es calculada: en el primer caso, se añade y desaparece
```

La última es la más rentable: buena parte del ruido de what-if es real y señala propiedades que el proveedor fija y la plantilla no declara. Declararlas explícitamente elimina la línea y, de paso, documenta el valor.

La validación previa completa la red:

```
$ bicep lint main.bicep                    # reglas del linter, en el editor y en CI
$ az deployment group validate -g rg-cloudshop-app-prod \
    --template-file main.bicep --parameters @prod.bicepparam
```

validate comprueba sintaxis, referencias y permisos sin crear nada. Es rápida y detecta el error de alcance, el nombre inválido y el parámetro que falta — los tres fallos que si no aparecen en CI aparecen a mitad de un despliegue, con parte de los recursos ya creados.

Y eso lleva al punto que más tranquiliza saber de antemano: un despliegue fallido no se revierte solo. Si diez de quince recursos se crearon y el undécimo falla, los diez se quedan. ARM no tiene transacción. La consecuencia de diseño es que las plantillas deben ser idempotentes y reejecutables: la respuesta a un fallo a medias es corregir y volver a desplegar, no limpiar a mano. Una plantilla que solo funciona sobre un grupo de recursos vacío es una plantilla que no sirve para operar.

5. Secretos, módulos y el historial que todo el mundo puede leer

Tres prácticas cierran la clase, y la primera es la que más veces está mal.

Las salidas quedan en claro en el historial de despliegues. Cualquiera con permiso de lectura sobre el grupo de recursos puede consultarlas:

```
$ az deployment group show -g rg-cloudshop-app-prod -n despliegue-app \
    --query "properties.outputs" -o json
```

Devolver una cadena de conexión, una clave o una contraseña como salida es publicarla. Y no basta con dejar de hacerlo: el historial conserva las anteriores, así que un secreto que estuvo en una salida hay que considerarlo comprometido, rotarlo y borrar esa entrada del historial.

La forma correcta de que una plantilla use un secreto sin verlo:

```
resource almacen 'Microsoft.KeyVault/vaults@2023-07-01' existing = {
  name: 'kv-cloudshop'
  scope: resourceGroup('rg-cloudshop-seg-prod')
}

module baseDatos 'modulos/sql.bicep' = {
  name: 'despliegue-sql'
  params: {
    contrasena: almacen.getSecret('sql-admin-password')   // no pasa por la plantilla
  }
}
```

getSecret() solo funciona pasando el valor a un parámetro marcado como seguro dentro de un módulo, y esa restricción es deliberada: impide usarlo en una expresión que acabe en un registro o en una salida.

```
@secure()
@description('Contraseña del administrador. No se registra ni se devuelve.')
```

```
param contrasena string
```

Un parámetro @secure() se omite en el historial y en los registros. Sin la anotación, el valor del archivo de parámetros queda registrado igual que cualquier otro.

Y la mejor opción sigue siendo no tener el secreto: si el recurso admite identidad administrada, la clase 038 ya dio la respuesta — no hay contraseña que pasar.

Los módulos se versionan como código. Un módulo publicado en un registro de contenedores se referencia por versión, y eso permite que un equipo actualice cuando le convenga en vez de cuando otro despliegue:

```
module red 'br:acrcloudshop.azurecr.io/bicep/modules/red:1.4.0' = {
  name: 'despliegue-red'
  params: { entorno: 'prod' }
}
```

Sin versión —referencias a una ruta local compartida o a latest— un cambio en el módulo se aplica a todos los que lo usan en su siguiente despliegue, que puede ser el de producción de otro equipo un viernes. Es el mismo argumento de fijar versiones de dependencias, aplicado a la infraestructura.

Y la plantilla es el sitio donde vive el gobierno de la parte. Las decisiones de las clases anteriores se declaran, no se recuerdan:

```
resource cuenta 'Microsoft.Storage/storageAccounts@2023-05-01' = {
  name: nombreCuenta
  location: ubicacion
  sku: { name: 'Standard_ZRS' }                // redundancia decidida (041)
  kind: 'StorageV2'
  properties: {
    allowBlobPublicAccess: false               // acceso público cerrado (041)
    allowSharedKeyAccess: false                // sin clave compartida (041)
    minimumTlsVersion: 'TLS1_2'
    publicNetworkAccess: 'Disabled'            // solo punto privado (039)
  }
  tags: etiquetas                              // atribución de costo (037)
}
```

Seis decisiones de cuatro clases distintas, en un recurso. Esa es la función real de la infraestructura como código en este programa: convertir lo que se aprendió en algo que no se puede olvidar al crear el siguiente recurso. Y lo que la plantilla no puede garantizar por sí sola —que nadie lo cambie después— lo cubre la directiva de la clase 046. Las dos capas juntas son el mecanismo; ninguna de las dos basta sola.

Ejemplo trabajado

CloudShop pasa a declarar su plataforma Azure en Bicep. La primera versión funciona y produce cuatro incidentes en dos meses, todos por comportamientos de ARM que la plantilla no controlaba.

Punto de partida:

```
un único main.bicep de 1.100 líneas, alcance resourceGroup
un grupo de recursos compartido: rg-cloudshop-prod
despliegue en modo completo "para que quede limpio"
los parámetros incluyen la contraseña de SQL en texto
```

Incidente 1 — el despliegue borra un disco de otro equipo.

El equipo de datos había creado una instantánea en el mismo grupo de recursos para una migración. El siguiente despliegue en modo completo la eliminó, junto con una cuenta de almacenamiento temporal con 900 GB.

```
What-if habría mostrado:
  - Microsoft.Compute/snapshots/migracion-2026-06
  - Microsoft.Storage/storageAccounts/stmigraciontmp
pero no se ejecutó: el modo completo estaba en la canalización sin paso previo
```

Se corrige por diseño y no por prudencia:

```
grupos de recursos              1 compartido      4, uno por unidad desplegable
modo de despliegue              completo          incremental + pila de despliegue
`what-if` en la canalización    no                sí, obligatorio y revisado
bloqueos de recurso             ninguno           sobre almacén y servidor SQL
```

La pila de despliegue da lo que faltaba: una lista de qué pertenece a este despliegue, y un borrado ordenado cuando se retire el entorno.

Incidente 2 — reglas de NSG que aparecen y desaparecen.

Durante tres semanas hay fallos de conectividad intermitentes hacia la base de datos. La regla que permite el puerto 5432 desde snet-app está unas veces y otras no.

```
$ az network nsg rule list -g rg-cloudshop-red --nsg-name nsg-datos \
    --query "[].name" -o tsv
allow-lb-probe
deny-lateral
# falta allow-app-5432
```

La causa estaba repartida entre dos plantillas: la de red declaraba securityRules en línea dentro del NSG, y la de la aplicación declaraba allow-app-5432 como recurso hijo independiente. Cada despliegue de red sustituía la lista completa y se llevaba la regla de la otra.

```
forma de declarar reglas         mezclada (dos plantillas)  solo recurso hijo
convención documentada                    no                      sí
fallos de conectividad en 3 semanas       11                       0
```

La regla escrita para el resto de la plataforma es la que evita la repetición: para cada tipo de recurso hijo, una sola forma, y se decide una vez — reglas de NSG, ajustes de aplicación, reglas de firewall de base de datos y registros de DNS.

Incidente 3 — una cadena de conexión legible por veinte personas.

Una auditoría de accesos revisa el historial de despliegues:

```
$ az deployment group show -g rg-cloudshop-app-prod -n despliegue-app \
    --query "properties.outputs.cadenaConexion.value" -o tsv
Server=tcp:sql-cloudshop-prod.database.windows.net;…;Password=…
```

La plantilla la devolvía como salida para que la usara el paso siguiente de la canalización. Cualquiera con permiso de lectura sobre el grupo de recursos —veintiuna personas— podía consultarla, y estaba en las 47 entradas del historial.

```
secreto en salidas                   sí                    no
parámetro de contraseña          texto plano       @secure() + getSecret()
credencial de la aplicación        contraseña      identidad administrada (038)
historial con el secreto          47 entradas       purgado y contraseña rotada
```

La tercera fila es la corrección de fondo: el paso siguiente de la canalización no necesitaba la cadena, necesitaba poder conectarse. Con identidad administrada no hay ninguna cadena que pasar.

Incidente 4 — el cambio real que nadie vio entre el ruido.

Un despliegue rutinario deja el Always On de la aplicación de administración en false, y el servicio empieza a tardar ocho segundos en la primera petición de la mañana (clase 043). El what-if de esa ejecución lo mostraba:

```
~ Microsoft.Web/sites/app-admin
  ~ properties.siteConfig.alwaysOn: true → false
```

Entre otras 214 líneas, casi todas propiedades calculadas que aparecían en todas las ejecuciones. Nadie las leía desde hacía semanas.

```
líneas de what-if por ejecución        214              9
módulos                            1 de 1.100 líneas   7 módulos
propiedades calculadas declaradas       no          sí, 31 añadidas
revisión del what-if en el pull request  informal    obligatoria, con aprobación
```

La reducción de 214 a 9 no se consiguió filtrando: se consiguió declarando explícitamente las propiedades que el proveedor fijaba y la plantilla no mencionaba. El ruido era información sobre lo que faltaba en la plantilla.

Resumen del paso a infraestructura declarada:

```
tamaño de la plantilla principal           1.100 líneas    7 módulos versionados
grupos de recursos                         1 compartido    4 por unidad
modo de despliegue                         completo       incremental + pila
recursos borrados por accidente                 2               0
fallos de conectividad por reglas volátiles    11               0
secretos en el historial de despliegues        47               0
líneas de what-if por ejecución               214               9
tiempo de despliegue completo               22 min          14 min
```

```
reversión ensayada                          no        sí, versión anterior
                                                        del módulo, 6 min
```

La lección que esta clase traslada al proyecto de la clase 048 y a la parte 07: una plantilla no es un guion de creación, es la declaración del estado completo de lo que toca. Todo lo que no declara vuelve a su valor por omisión, todo lo que declara dos veces se lo disputan dos canalizaciones, y todo lo que devuelve queda escrito donde muchos pueden leerlo. Entender esas tres consecuencias es la diferencia entre automatizar la infraestructura y automatizar los incidentes.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-03-azure-core-platform/047-bicep-plantillas-y-despliegues-por-alcance/lab.py
```

El laboratorio selecciona el motor de práctica iac y produce labresult.json. El escenario, sus comprobaciones y el artefacto esperado corresponden a esta clase; no requiere credenciales y deja explícito qué debe revalidarse en un sandbox real.

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa infraestructura-bicep en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es un plan reproducible sin secretos ni cambios inesperados. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye infraestructura-bicep para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

Errores frecuentes

Síntoma	Causa probable	Corrección
Un despliegue elimina recursos que nadie tocó en la plantilla	El modo completo borra todo lo que no aparece en ella, y el grupo de recursos era compartido	Un grupo de recursos por unidad desplegable, modo incremental con pila de despliegue y what-if obligatorio antes de aplicar.
Unas reglas de red aparecen y desaparecen según qué canalización se ejecutó la última	El mismo tipo de recurso hijo está declarado en línea en una plantilla y como recurso independiente en otra	Elige una sola forma por tipo de recurso hijo, documéntala y aplícala en todas las plantillas.
Una contraseña aparece en el historial de despliegues	Se devolvió como salida o se pasó a un parámetro sin @secure()	Marca el parámetro como seguro, obtén el valor con getSecret() y, mejor aún, sustituye la credencial por identidad administrada; rota lo que ya se expuso.
Nadie revisa la salida de what-if porque siempre muestra decenas de cambios	La plantilla no declara propiedades que el proveedor fija, y aparecen como diferencias en cada ejecución	Declara esas propiedades explícitamente y divide en módulos: el objetivo es una salida corta que se lea de verdad.
Un despliegue falla a medias y deja recursos creados	ARM no tiene transacción: no revierte lo ya aplicado	Escribe plantillas idempotentes y reejecutables; la respuesta a un fallo parcial es corregir y volver a desplegar.
Error de función no reconocida o de recurso no válido al desplegar	La plantilla está en un alcance que no permite ese recurso o esa función	Ajusta targetScope y despliega dentro del nivel correcto con módulos y el parámetro scope.

Seguridad, ética y costo

Trabaja con cuentas propias o sandboxes autorizados. No publiques secretos, identificadores reales ni datos personales. Antes de crear recursos pagos define presupuesto, etiquetas y comando de destrucción; después verifica que no queden recursos huérfanos. Los ejemplos locales enseñan contratos, pero no certifican cumplimiento ni disponibilidad de producción.

Preguntas de comprobación

1. ¿Qué problemas desaparecen al no haber archivo de estado, y qué capacidad se pierde a cambio?
2. ¿Desde qué alcance se crea un grupo de recursos, y por qué no se puede hacer desde el de grupo de recursos?
3. Describe exactamente cómo desaparece una regla de NSG al mezclar declaración en línea y recurso independiente.
4. ¿Por qué una salida de despliegue no puede contener un secreto, y qué hay que hacer si ya lo contuvo?
5. Tu what-if muestra 200 líneas en cada ejecución. ¿Qué te dice eso sobre la plantilla y cómo lo reduces?

Referencias

- Microsoft (2025). What is Bicep? — relación con ARM, módulos y compilación transparente. <https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/overview>
- Microsoft (2025). Deployment scopes in Bicep — targetScope, módulos con scope y qué se crea en cada nivel. <https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/deploy-to-subscription>
- Microsoft (2025). Azure Resource Manager deployment modes — incremental frente a completo y qué elimina cada uno. <https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/deployment-modes>
- Microsoft (2025). Deployment what-if operation — símbolos, limitaciones y ruido de proveedores. <https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/deploy-what-if>
- Microsoft (2025). Use Azure Key Vault to pass secure parameter values — @secure(), getSecret() e historial de despliegues. <https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/key-vault-parameter>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Evaluación — 047 Bicep, plantillas y despliegues por alcance

Preguntas

1. Explica el problema que resuelve esta capacidad sin mencionar un proveedor.
2. Identifica una frontera de responsabilidad y su propietario.
3. Compara dos alternativas mediante confiabilidad, seguridad, costo y operación.
4. ¿Qué demuestra el laboratorio y qué no puede demostrar?
5. Propón un caso límite o una prueba de fallo.

Reto verificable

Entrega infraestructura-bicep con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

- [ ] Resultado reproducible por una segunda persona.
- [ ] Evidencia vinculada a cada afirmación técnica.
- [ ] Prueba negativa o de fallo incluida.
- [ ] Costos y permisos expresados con unidades y alcance.
- [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.

048 — Proyecto: aplicación de tres capas en Azure

Parte: 03 — Azure: plataforma esencial Nivel: intermedio · Horas estimadas: 8 Laboratorio: capstone · Estado: EXECUTABLECORE

Propósito

Integrar las once clases anteriores en una plataforma Azure que funcione, se observe, falle de forma controlada y cueste algo explicable — y responder con evidencia la pregunta que abrió la parte: de todo lo aprendido en AWS, qué era arquitectura y ha reaparecido, y qué era mecanismo del proveedor. La respuesta a esa pregunta, escrita y con las excepciones señaladas, es lo que se lleva a la parte 04 y lo que hace que la tercera plataforma cueste una fracción de esfuerzo que la segunda.

Resultados de aprendizaje

Al finalizar podrás:

1. Trazar cada componente de la arquitectura hasta la clase que lo decidió y la alternativa descartada.
2. Separar el contrato portable del mecanismo específico del proveedor, señalando dónde no hubo equivalencia.
3. Declarar la línea base en dos capas: lo que la plantilla garantiza y lo que la directiva vigila.
4. Provocar tres fallos y medir detección, impacto y recuperación en cada uno.
5. Comparar dos plataformas con una unidad común, sabiendo por qué comparar facturas no sirve.

Conceptos centrales

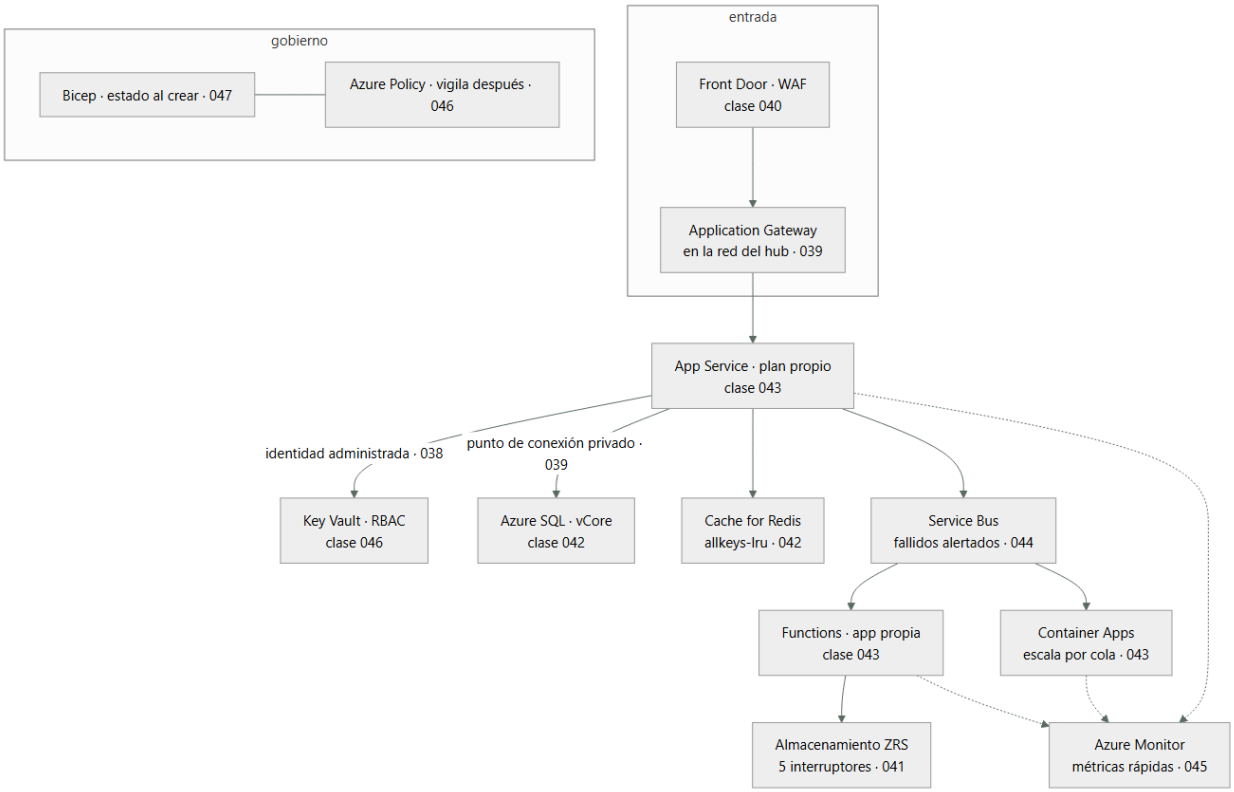
Concepto	Comprensión verificable
trazabilidad de decisión	Cada componente debe poder responder tres preguntas: qué requisito lo exige, qué alternativa se descartó y qué evidencia demuestra que cumple. Un componente sin las tres es una elección, no una decisión.
contrato portable	Lo que sobrevive al cambio de proveedor: el requisito, el patrón y la propiedad exigida al manejador. Cambia el nombre del servicio, no la obligación.
línea base en dos capas	La plantilla garantiza el estado al crear; la directiva vigila que siga así. Ninguna de las dos basta sola: la primera no impide un cambio posterior y la segunda no crea nada.
hallazgo de prueba de fallo	Lo que aparece al romper algo y no aparecía en ninguna configuración ni en ningún panel. Es el entregable real de un simulacro; que «todo aguantó» significa que el simulacro fue demasiado suave.
riesgo residual declarado	Lo que se decide no cubrir, con su motivo, su responsable y la condición que obligaría a revisarlo. Un riesgo no escrito no está aceptado: está ignorado.
costo por unidad de negocio	Única forma útil de comparar dos plataformas. Comparar facturas totales compara arquitecturas y tamaños distintos, no proveedores.

Modelo mental

En Azure, identidad y jerarquía organizacional preceden al recurso: tenant, management group, suscripción y resource group determinan el alcance de control.

Aplicado a esta clase, separa siempre cuatro planos: intención (qué necesita el usuario), configuración (qué declaramos), estado observado (qué existe de verdad) y evidencia (cómo sabemos que cumple). Confundirlos produce diseños que se ven correctos en un diagrama pero fallan al operar.

■ Flujo de razonamiento



Desarrollo

1. De dónde sale cada decisión, y qué trampa evitó

La arquitectura de la entrega no es una elección de servicios: es el resultado de once decisiones con su alternativa descartada. La tabla que hay que poder defender:

Componente	Requisito que lo exige	Alternativa descartada	Trampa de Azure que evita
Grupos de recursos por unidad	Aislamiento de despliegue y borrado (037)	Un grupo compartido	El modo completo borrando lo ajeno (047)
Identidad administrada asignada por el usuario	Sin secretos y flota compartida (038)	Secreto de aplicación con dos años	La rotación que nadie hace
Roles de datos además de gestión	Leer blobs y secretos (038)	Owner sobre la suscripción	AuthorizationPermissionMismatch con Owner
NAT Gateway + subred sin salida	Salida declarada y auditable (039)	Confiar en la salida implícita	La «subred privada» que no lo era
Puntos de conexión privados + zona DNS	Datos fuera de internet (039)	Endpoints de servicio	Funcionar por el camino público sin saberlo
App Service en planes separados	Radio de impacto (043)	Un plan para todo	El vecino ruidoso que degrada la tienda
Aplicación de funciones por perfil de carga	Aislar lo interactivo (043)	Una sola aplicación	El escalado por aplicación, no por función
Service Bus para el trabajo	Entrega con fallo visible (044)	Event Hubs «porque escala»	La partición bloqueada sin cola de fallidos
Azure SQL vCore	Diagnóstico por recurso (042)	Modelo DTU	El 100 % que no dice de qué
Alertas de métrica para lo urgente	Detección en un minuto (045)	Alertas de registro	Nueve minutos estructurales
Bicep + directiva	Estado al crear y vigilancia después (046, 047)	Solo plantillas	El control que alguien revierte

Cada fila responde a las tres preguntas de una decisión trazable: qué requisito, qué alternativa y qué evidencia. La cuarta columna es la que solo se puede escribir después de haber recorrido la parte: es la lista de errores que esta plataforma ya no puede cometer, porque su diseño los excluye en vez de confiar en que nadie los repita.

Y hay cuatro decisiones que se tomaron en contra de lo que sugería la traducción desde AWS, que conviene destacar porque son las que un revisor cuestionará:

1. No se usó el grupo de recursos como equivalente de la cuenta de AWS.
   Las cuotas son de la suscripción y no hay aislamiento de red entre
   grupos (037): la frontera de aislamiento es la suscripción.

2. No se buscó un equivalente del `Deny` de IAM.
   Azure RBAC es aditivo; lo que cumple esa función es Azure Policy,
   y opera en otro plano y con otro código de error (038, 046).

3. No se replicó el patrón de subred pública y privada por tabla de rutas.
   No existe: hay que declarar la salida y desactivarla explícitamente (039).

4. No se eligió redundancia geográfica para los datos de facturación.
   La región emparejada la asigna Microsoft y quedaba fuera de la
   jurisdicción aprobada (041): se usó ZRS más replicación explícita.

Las cuatro tienen la misma forma: el objetivo se conserva y el mecanismo no tiene equivalente. Reconocer esa forma es lo que hace útil la comparación entre proveedores; buscar la traducción literal es lo que produce los incidentes de esta parte.

2. Lo portable y lo del proveedor, con las excepciones señaladas

Este es el entregable intelectual de la parte, y sirve exactamente para una cosa: hacer que la parte 04 cueste una fracción del esfuerzo.

Lo que se repitió sin cambios —el contrato— y solo cambió de nombre:

Contrato que se conserva	En AWS (parte 02)	En Azure (parte 03)
Identidad de carga sin secretos	Rol de instancia y OIDC	Identidad administrada y federación
Sujeto de federación acotado	sub del rol federado	subject de la credencial federada
Frontera de aislamiento del entorno	Cuenta	Suscripción
Salida declarada y con costo por GB	NAT gateway	NAT Gateway
Acceso privado a servicios gestionados	Endpoint de interfaz	Punto de conexión privado
Entrega al menos una vez	SQS y sus reintentos	Service Bus y su bloqueo
Cola de mensajes fallidos con alerta a cero	Explícita	Automática, y sin alerta
Idempotencia en el manejador	Obligatoria	Obligatoria
Clave de partición como decisión irreversible	DynamoDB	Cosmos DB, con tope duro de 20 GB
Prueba negativa por control	Obligatoria	Obligatoria
Costo por unidad de negocio	Etiquetas y atribución	Etiquetas y atribución

Once filas. Ninguna de las once tuvo que volver a aprenderse, y todas se implementaron más rápido la segunda vez.

Lo que no tuvo equivalencia y hubo que aprender de cero:

```
dos planos de identidad          roles de directorio y roles de recurso
                                 son sistemas distintos (038)
actions frente a dataActions     gestionar un recurso no es leer sus datos (038)
sin subred privada por defecto   la salida es el estado inicial (039)
dos NSG en cadena                subred y NIC, ambos deben permitir (039)
dos límites de disco             el de la máquina suele mandar (040)
la cuenta como unidad            redundancia, red y claves son de la cuenta (041)
región emparejada asignada       no se elige el destino (041)
la unidad que escala             plan, aplicación de funciones, revisión (043)
Deny no toca lo existente        el gobierno actúa sobre la petición (046)
alcance de despliegue            qué se puede crear depende del nivel (047)
```

Diez conceptos. Ese es el tamaño real de la diferencia entre dos plataformas grandes: once contratos que se reutilizan y diez mecanismos que se aprenden. No es una proporción intuitiva, y explica por qué la segunda nube cuesta mucho menos de lo que la gente teme y la primera mucho más de lo que espera.

Y la conclusión operativa que se lleva a la parte 04: al abrir un proveedor nuevo, las once preguntas de la primera tabla ya están escritas y solo hay que averiguar cómo se llama la respuesta. El trabajo real está en detectar las excepciones —los sitios donde la respuesta es «aquí no existe»—, porque son las que producen incidentes cuando se asume equivalencia.

3. La línea base en dos capas, que es el entregable que más dura

La configuración de una plataforma se degrada. No por descuido excepcional, sino por caminos normales: alguien recrea un recurso desde una plantilla vieja, una excepción temporal se queda, un despliegue sobrescribe un ajuste. Un control verificado una vez es un control verificado en el pasado.

Por eso la línea base se declara en dos capas que hacen cosas distintas:

```
capa 1 · Bicep      fija el estado en el momento de crear
                    lo que no declara vuelve a su valor por omisión (047)
capa 2 · Policy     vigila que siga así después
                    y en modo Audit lo hace sin bloquear nada (046)
```

Ninguna basta sola. La plantilla no impide que alguien cambie algo mañana desde el portal; la directiva no crea nada y solo constata. Juntas, un cambio fuera de la plantilla aparece como incumplimiento en horas.

La línea base concreta de esta entrega, con la clase que la justifica:

```
red
  ninguna subred con salida no declarada                          039
  todo acceso a datos por punto de conexión privado con zona DNS  039
  segmentación explícita entre subredes                           039
datos
  acceso público deshabilitado en almacenamiento y base de datos  041, 042
  acceso con clave compartida deshabilitado                       041
  cinco interruptores de protección activos y probados            041
  retención a largo plazo y bloqueo sobre el servidor lógico      042
identidad
  cero secretos de aplicación; identidad administrada en todo     038
  Key Vault con RBAC y protección contra purga                    046
  roles privilegiados elegibles, no permanentes                   038
cómputo
  un plan por radio de impacto; una app de funciones por perfil   043
  sonda de estado que ejercita dependencias                       040
asíncrono
  manejadores idempotentes                                        044
  alerta de mensajes fallidos con umbral cero                     044
  alerta de acumulación en cada consumidor                        045
observabilidad
  diagnóstico activado por directiva en el 100 % de los recursos  045
  agregaciones con `sum(itemCount)`, nunca `count()`              045
  alertas de métrica para lo urgente                              045
gobierno
  cero exclusiones de ámbito; excepciones como exenciones con caducidad 046
  etiquetas de atribución de costo obligatorias                   037
```

Veintitrés afirmaciones. Cada una tiene su prueba negativa en el guion de verificación, y cada una tiene además una directiva en modo Audit que la vigila. Esa duplicidad es deliberada: el guion demuestra el estado hoy, la directiva detecta la desviación mañana.

Y el criterio para decidir qué entra en la línea base, que evita que crezca hasta ser inmanejable: entra lo que, si se revierte, produce un incidente o una brecha. Lo demás es una preferencia y se documenta en otro sitio.

4. Tres fallos provocados y lo que enseñó cada uno

Un simulacro en el que todo aguanta no aporta información: significa que fue demasiado suave. Los tres de esta entrega se eligieron porque atacan tres supuestos distintos.

Fallo 1 — caída de una zona. Se detienen las instancias de la zona 1 en el plan de App Service y se fuerza la conmutación de la base de datos.

```
detección por alerta de métrica             52 s
p95 durante el episodio                    212 ms  (SLO 500 ms)
errores HTTP observados por el cliente     1.104
recuperación completa                      3 min 40 s
```

Y el hallazgo, que ninguna configuración mostraba. Los 1.104 errores no vinieron del frontal ni del balanceador: vinieron de la base de datos durante los 38 segundos de su conmutación entre réplicas. Azure SQL devuelve errores transitorios identificables durante ese intervalo —40613, 40501, 49918— y la aplicación no los reintentaba: los propagaba como error 500.

```
síntoma observable      ninguno en los paneles de infraestructura
consecuencia real       1.104 peticiones perdidas, 47 pedidos sin registrar
causa                   sin manejo de fallos transitorios en el acceso a datos
```

La corrección no es de infraestructura: es de código, y es obligatoria en Azure SQL porque la conmutación es parte del funcionamiento normal, no un incidente.

```
1. reintento con retroceso exponencial para los códigos transitorios conocidos
2. tiempo de espera de conexión menor que el plazo de la petición
3. la prueba de fallo pasa a incluir una conmutación forzada, no solo
   la caída de las instancias de cómputo
```

Fallo 2 — revocación de la clave gestionada por el cliente. Se retira el permiso de la identidad sobre la clave que cifra el almacenamiento, para comprobar que el control funciona y medir su radio.

```
tiempo hasta que el almacenamiento queda inaccesible   11 min (caché de la clave)
servicios afectados esperados                          1 (subida de facturas)
servicios afectados reales                             3
recuperación tras restaurar el permiso                 6 min
```

El hallazgo: la misma cuenta de almacenamiento recibía la exportación del registro de actividad y el destino de mensajes fallidos de Event Grid. Revocar la clave no solo detuvo la subida de facturas: detuvo la evidencia de auditoría y el destino de los eventos que fallaban, justo cuando más falta hacían. El control funcionaba; el radio de impacto estaba mal estimado porque una cuenta servía a tres propósitos.

```
corrección   separar por propósito: facturas, auditoría y eventos fallidos
             en cuentas distintas, con claves distintas
registro     el orden de las operaciones de revocación y restauración
             queda documentado como procedimiento, no improvisado
```

Fallo 3 — consumidor detenido en silencio. Se detiene el consumidor de notificaciones sin apagar nada más, que es la forma en que este sistema falló dos veces durante la parte.

```
detección                          ninguna (2 días)         4 min 20 s
mensajes acumulados al detectar    41.000                     680
señal que lo detectó                 —              profundidad de cola
```

El hallazgo: la alerta se disparó correctamente y apuntaba a un procedimiento que describía cómo reiniciar el consumidor, sin decir cómo comprobar si los mensajes acumulados se habían procesado o duplicado tras el reinicio. La detección funcionaba y la respuesta estaba incompleta.

```
corrección   el procedimiento incluye la verificación posterior:
             recuento de notificaciones enviadas frente a pedidos del intervalo,
             apoyándose en la idempotencia del manejador (044)
```

Los tres hallazgos tienen la misma forma, y merece la pena decirlo explícitamente porque es la lección de la parte: la infraestructura aguantó los tres fallos y en los tres se perdió algo. Lo que falló fue el manejo de errores transitorios, la estimación del radio de impacto y la completitud del procedimiento. Ninguna de las tres cosas aparece en una revisión de configuración.

5. La entrega, la comparación honesta y la pregunta que abre la parte 04

La entrega, sin conocimiento tácito. El criterio es el de la clase 036 y no ha cambiado: otra persona debe poder repetir el recorrido sin preguntar nada.

```
infraestructura     Bicep en 7 módulos versionados, con `what-if` legible
línea base          23 afirmaciones, cada una con su prueba negativa
verificar.sh        ejecuta las 23 y devuelve código de salida
directivas          la iniciativa que vigila las 23 después
ADR                 11 decisiones con su alternativa descartada
riesgos residuales  5, con responsable y condición de revisión
consultas KQL       4, guardadas y enlazadas desde el manual
procedimientos      3 fallos ensayados, con su verificación posterior
```

```
línea base medida   rendimiento, costo y costo por pedido
```

Los cinco riesgos residuales merecen figurar porque declararlos es parte del trabajo:

```
1. una sola región: el RTO ante caída regional es de horas, aceptado
2. la conmutación de la cuenta de almacenamiento la deja en LRS (041)
3. el paso por el concentrador cuesta 32 USD/mes en inspección (039)
4. dos cuentas de emergencia con acceso permanente, documentadas (038)
5. el nivel estándar de Service Bus no admite punto de conexión privado;
   se acepta hasta que el volumen justifique el nivel premium (044)
```

La comparación con la parte 02, hecha de forma que signifique algo.

La comparación ingenua es inútil y conviene enseñarla antes de descartarla:

```
plataforma AWS (clase 036)     688,40 USD/mes
plataforma Azure, primera medida  1.389 USD/mes
```

Eso no compara proveedores: compara dos arquitecturas distintas y dos conjuntos de capacidades distintos. Al desglosarlo, casi todo el hueco tenía nombre y apellidos:

```
                                          primera      las palancas
base de datos aprovisionada frente a elástica   390            390
pasarela de aplicación con WAF                  185            185
telemetría en plan de análisis para todo        276            110   (045)
planes de Defender en todo                      230             96   (046)
cómputo, red, mensajería y almacenamiento       308            159
escalado automático de Cosmos mal dimensionado   —              —    (042)
                                              ■■■■■■■        ■■■■■■■
                                              1.389            940
```

Y la única cifra que compara de verdad, porque normaliza por trabajo hecho:

```
AWS    0,000702 USD por pedido
Azure  0,000959 USD por pedido    (+37 %)
```

Ese 37 % es real y tiene dos partidas identificadas: la pasarela con WAF y la ingesta de telemetría, ambas más caras que sus equivalentes de la parte 02 al mismo volumen. No es una penalización del proveedor: es el precio de dos capacidades concretas, y con ese desglose se puede decidir si compensan. Sin él, la conversación se convierte en «Azure es más caro», que no es una afirmación accionable.

El rendimiento, medido con la misma carga y el mismo método:

```
rps sostenidos          987,4         984,1
p50                    38,4 ms       41,2 ms
p95                    91,2 ms       96,8 ms
p99                   184,7 ms      203,5 ms
errores                    0             0
p99/p50                  4,8×          4,9×
```

La conclusión honesta es que no hay diferencia significativa, y esa es una información valiosa: descarta el rendimiento como criterio de elección a esta escala y devuelve la decisión a donde corresponde —capacidades, costo por unidad, competencias del equipo y requisitos de residencia—.

Y la pregunta que abre la parte 04. Con dos plataformas construidas y comparadas, la pregunta ya no es cuál es mejor. Es esta:

> De las once filas del contrato portable, ¿cuántas seguirán siendo válidas en Google Cloud, y cuáles de los diez conceptos sin equivalencia volverán a aparecer con otra forma?

La hipótesis que se lleva escrita —para poder equivocarse de forma comprobable— es que el contrato se conserva entero y que las excepciones serán distintas: donde Azure separó identidad de directorio e identidad de recurso, Google Cloud tiene una jerarquía de proyectos y carpetas con herencia propia; donde Azure no tiene subred privada, otro proveedor tendrá otra asimetría. Lo que se comprueba en la parte 04 no es una lista de servicios: es si el método de esta parte —contrato primero, excepciones señaladas, prueba negativa siempre— funciona a la tercera.

Ejemplo trabajado

Entrega del capstone de la parte 03, con las cifras que se llevan a la parte 04.

Verificación completa. Las 23 afirmaciones de la línea base, cada una con su prueba negativa:

```
$ ./verificar.sh
```

```
✓ federación deniega desde otro repositorio        AADSTS70021
✓ almacenamiento sin acceso público                403 desde fuera
✓ clave compartida deshabilitada                   KeyBasedAuthenticationNotPermitted
✓ nombre de blob resuelve a IP privada             10.20.6.5
✓ subred de datos sin salida a internet            tiempo agotado a los 5 s
✓ base de datos sin acceso público                 conexión rechazada
✓ tráfico lateral bloqueado salvo 5432             Deny · deny-lateral
✓ sonda del balanceador permitida                  Allow · allow-lb-probe
✓ recuperación de contenedor                       50 blobs restaurados
✓ restauración de base de datos                    1.284.391 filas · 11 min
✓ protección contra purga del almacén              operación rechazada
✓ directiva de red de almacenamiento               RequestDisallowedByPolicy
✓ antimalware en cuenta de cargas                  alerta en 4 min
✓ rotación de secreto con tráfico en curso         0 errores
✓ mensajes fallidos alertan a cero                 aviso en 15 min
✓ acumulación en cola alerta                       aviso en 4 min
✓ diagnóstico en el 100 % de recursos              61 de 61
✓ recuento de peticiones exacto                    sum(itemCount) verificado
✓ salud del servicio                               200
✓ pedido con comilla simple y punto y coma         201
✓ despliegue en modo incremental                   0 eliminaciones en what-if
✓ sin secretos en el historial de despliegues      0 coincidencias
✓ roles Owner permanentes                          2, ambos de emergencia
23/23 correctas
```

Línea base medida:

```
rps 984,1 · p50 41,2 ms · p95 96,8 ms · p99 203,5 ms · errores 0
L = 984 × 0,0412 = 40,5 de 50 solicitados → medida válida
p99/p50 = 4,9×
costo 940 USD/mes · 0,000959 USD/pedido
```

La comprobación de la ley de Little es la misma disciplina de la clase 036: si la concurrencia observada no cuadra con la solicitada, el generador de carga no estaba aplicando lo que decía y la medida no sirve.

Los tres fallos, con lo aprendido en cada uno:

```
caída de zona                  52 s        212 ms      1.104 peticiones
                                                       47 pedidos     los errores transitorios
                                                                      de SQL son normales:
                                                                      hay que reintentarlos

revocación de clave propia     11 min         —        3 servicios,    el radio de impacto
                                                       no 1            se estima mal cuando
                                                                       una cuenta sirve a tres
                                                                       propósitos

consumidor detenido          4 min 20 s    sin cambio   680 mensajes   detectar no es responder:
                                                       retrasados     el procedimiento no
                                                                      verificaba el resultado
```

El hallazgo que justificó el capstone. Todo estaba configurado correctamente, las 23 pruebas negativas pasaban y los paneles estuvieron en verde durante la caída de zona:

```
requests
| where timestamp between (datetime(2026-07-28 10:14) .. datetime(2026-07-28 10:18))
| summarize peticiones = sum(itemCount),
            fallidas = sumif(itemCount, success == false)

peticiones  58.402
fallidas     1.104        ← 1,9 %, por debajo del umbral de alerta del 3 %

exceptions
| where timestamp between (datetime(2026-07-28 10:14) .. datetime(2026-07-28 10:18))
| summarize por = count() by outerMessage

1.104 × "Database 'pedidos' on server '…' is not currently available" (40613)
```

Mil ciento cuatro errores con todas las alarmas en verde, porque el umbral estaba por encima y porque la infraestructura hizo exactamente lo que debía: conmutar en 38 segundos. La pérdida no la causó el fallo, la causó el código que no esperaba el fallo.

```
síntoma observable   ninguno: alarmas en verde, conmutación correcta,
                     p95 dentro del SLO
consecuencia real    47 pedidos no registrados
```

```
causa                sin reintento de errores transitorios documentados
qué lo destapó       provocar el fallo, no revisar la configuración
```

Se corrige y se amplía el alcance del simulacro:

```
1. reintento con retroceso para los códigos transitorios de Azure SQL
2. umbral de alerta por tasa de error bajado del 3 % al 1 %, con ventana corta
3. la prueba de fallo incluye conmutación forzada de base de datos
4. la comprobación posterior cuenta pedidos registrados frente a peticiones
   aceptadas: la diferencia debe ser cero
```

Repetido el simulacro con las cuatro correcciones:

```
errores durante la conmutación    1.104            0
pedidos no registrados               47            0
detección                       no hubo         38 s
```

Se entrega a la parte 04 con:

```
contrato portable    11 filas verificadas en dos proveedores
excepciones          10 conceptos sin equivalencia, con su síntoma
línea base           23 afirmaciones y su guion de verificación
ADR                  11 decisiones con alternativa descartada
riesgos residuales   5, con responsable y condición de revisión
comparación          costo por pedido y percentiles, con el desglose
                     que explica el 37 % de diferencia
```

Y la hipótesis escrita, para poder equivocarse de forma comprobable en la parte 04:

> El contrato de once filas se conservará entero en Google Cloud. Las excepciones serán otras diez, distintas de estas, y la que más incidentes producirá volverá a ser una en la que el sistema funcione por el camino equivocado en lugar de dar un error.

La lección que esta parte deja al programa: las tres pruebas de fallo mostraron infraestructura que aguantó y sistemas que perdieron datos. Configurar bien es necesario y no es suficiente; lo que separa una plataforma operable de una bien configurada es haber roto cada supuesto a propósito y haber medido qué se perdió.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-03-azure-core-platform/048-proyecto-aplicacion-de-tres-capas-en-azure/lab.py
```

El laboratorio selecciona el motor de práctica capstone y produce labresult.json. El escenario, sus comprobaciones y el artefacto esperado corresponden a esta clase; no requiere credenciales y deja explícito qué debe revalidarse en un sandbox real.

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa plataforma-azure en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es un incremento integrado, demostrable y documentado. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye plataforma-azure para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.

- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
Un simulacro en el que todo aguanta y no se aprende nada	El fallo elegido no atacaba ningún supuesto real del diseño	Elige fallos que rompan un supuesto distinto cada uno y mide pérdida real, no solo disponibilidad.
Se pierden peticiones durante una conmutación de base de datos con todas las alarmas en verde	Los errores transitorios de Azure SQL son parte del funcionamiento normal y la aplicación no los reintenta	Reintento con retroceso para los códigos transitorios, umbral de alerta acorde y conmutación forzada dentro del simulacro.
Revocar una clave afecta a más servicios de los previstos	Una misma cuenta de almacenamiento servía a tres propósitos distintos	Separa por propósito y clave, y documenta el orden de las operaciones de revocación y restauración.
La alerta se dispara, el procedimiento se ejecuta y nadie sabe si se perdió trabajo	El procedimiento describe cómo restablecer y no cómo verificar el resultado	Todo procedimiento termina con una comprobación cuantitativa apoyada en la idempotencia del manejador.
Se comparan dos plataformas por su factura total y la conclusión no es accionable	Las facturas comparan arquitecturas y capacidades distintas, no proveedores	Normaliza por unidad de negocio y desglosa la diferencia hasta que cada partida tenga nombre.
Un control desaparece semanas después de haberse verificado	La plantilla fija el estado al crear y nada vigila lo que ocurre después	Línea base en dos capas: la plantilla garantiza y una directiva en modo Audit detecta la desviación en horas.

■ Seguridad, ética y costo

Trabaja con cuentas propias o sandboxes autorizados. No publiques secretos, identificadores reales ni datos personales. Antes de crear recursos pagos define presupuesto, etiquetas y comando de destrucción; después verifica que no queden recursos huérfanos. Los ejemplos locales enseñan contratos, pero no certifican cumplimiento ni disponibilidad de producción.

■ Preguntas de comprobación

1. Elige tres componentes de la arquitectura y di qué requisito los exige, qué alternativa se descartó y qué trampa de Azure evitan.
2. ¿Cuántos contratos se reutilizaron de la parte 02 y cuántos conceptos hubo que aprender de cero? ¿Qué implica esa proporción para la parte 04?
3. ¿Por qué la línea base necesita dos capas, y qué falla si solo existe una de ellas?
4. Durante la caída de zona hubo 1.104 errores con las alarmas en verde. ¿Qué falló exactamente y por qué no lo mostraba ninguna revisión de configuración?
5. ¿Qué hace inútil comparar las facturas de dos plataformas, y qué comparación sí es accionable?

Referencias

- Microsoft (2025). Azure Well-Architected Framework — los cinco pilares como rejilla de revisión de la entrega. <https://learn.microsoft.com/en-us/azure/well-architected/>
- Microsoft (2025). Troubleshoot transient connection errors in Azure SQL — códigos transitorios y lógica de reintento obligatoria. <https://learn.microsoft.com/en-us/azure/azure-sql/database/troubleshoot-common-connectivity-issues>

• Microsoft (2025). Reliability in Azure: availability zones — comportamiento de la conmutación y qué mide un simulacro de zona. <https://learn.microsoft.com/en-us/azure/reliability/availability-zones-overview>
• Microsoft (2025). Cloud Adoption Framework: landing zone design areas — línea base declarada y vigilada, y sus áreas de decisión. <https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/landing-zone/design-areas>
• Microsoft (2025). Chaos Studio experiment design — provocar fallos con alcance acotado y medir el resultado. <https://learn.microsoft.com/en-us/azure/chaos-studio/chaos-studio-overview>
• Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 03 en PDF · Recorrido de Azure en PDF · Manual integral

Anterior	Índice	Siguiente
← 047 · Bicep, plantillas y despliegues por alcance	Parte 03 · Programa	049 · Organización, folders, proyectos, billing y cuotas →

Evaluación — 048 Proyecto: aplicación de tres capas en Azure

Preguntas

1. Explica el problema que resuelve esta capacidad sin mencionar un proveedor.
2. Identifica una frontera de responsabilidad y su propietario.
3. Compara dos alternativas mediante confiabilidad, seguridad, costo y operación.
4. ¿Qué demuestra el laboratorio y qué no puede demostrar?
5. Propón un caso límite o una prueba de fallo.

Reto verificable

Entrega plataforma-azure con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

• [ ] Resultado reproducible por una segunda persona.
• [ ] Evidencia vinculada a cada afirmación técnica.
• [ ] Prueba negativa o de fallo incluida.
• [ ] Costos y permisos expresados con unidades y alcance.
• [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.