Multi-Cloud Engineering Program

Parte 04 — Google Cloud: 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	04 de 23
Clases	049–060
Evaluaciones	12
Horas estimadas	52

Índice

Parte 04 — Google Cloud: plataforma esencial

← Parte 03 · Índice completo · Parte 05 →

Descargar: Esta parte en PDF · Recorrido de Google Cloud en PDF · Manual integral

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

Diseñar y operar una carga en Google Cloud con proyectos, identidades y datos bien delimitados.

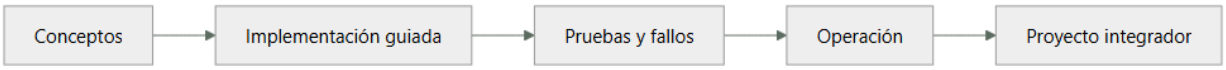
Resultados de la parte

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

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

Secuencia



Clases

ID	Clase	Laboratorio	Horas
049	Organización, folders, proyectos, billing y cuotas	governance	4
050	IAM, service accounts y Workload Identity Federation	iam	4
051	VPC global, subredes regionales, firewall y Cloud NAT	network	4
052	Compute Engine, managed instance groups y load balancing	compute	4
053	Cloud Storage, clases, lifecycle y replicación	storage	4
054	Cloud SQL, Spanner, Firestore y Memorystore	data	4
055	Cloud Run, Cloud Functions y API Gateway	serverless	4
056	Pub/Sub, Cloud Tasks y Workflows	messaging	4
057	Cloud Logging, Monitoring, Trace y Audit Logs	observability	4
058	Cloud KMS, Secret Manager y Security Command Center	security	4
059	Terraform y despliegues reproducibles en GCP	iac	4
060	Proyecto: aplicación de tres capas en Google Cloud	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

- Google Cloud Architecture Framework — Google.
- Official Google Cloud Certified Professional Cloud Architect Study Guide — Dan Sullivan.
- Data Engineering with Google Cloud Platform — Adi Wijaya.

049 — Organización, folders, proyectos, billing y cuotas

Parte: 04 — Google Cloud: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: governance · Estado: EXECUTABLECORE

Propósito

Montar la estructura de una organización en Google Cloud sabiendo que aquí el proyecto no equivale a la suscripción de Azure ni a la cuenta de AWS: es barato, desechable y la unidad de casi todo —API, cuotas, IAM, nombres—, y esa diferencia cambia el diseño en vez de solo cambiar los nombres. Es la clase donde se comprueba la primera parte de la hipótesis de la clase 048: qué contratos de las partes 02 y 03 reaparecen intactos y dónde aparece la primera asimetría propia.

Resultados de aprendizaje

Al finalizar podrás:

1. Diseñar una jerarquía de carpetas y proyectos sabiendo qué se hereda y qué se decide en cada nivel.
2. Usar el identificador correcto de un proyecto y explicar por qué el nombre no sirve para automatizar.
3. Dimensionar cuotas por proyecto y por región, y planificar los aumentos con antelación en vez de descubrirlos.
4. Separar el eje de facturación del eje jerárquico, y activar la exportación de costos antes de necesitarla.
5. Aplicar políticas de organización distinguiéndolas de IAM, con su prueba negativa.

Conceptos centrales

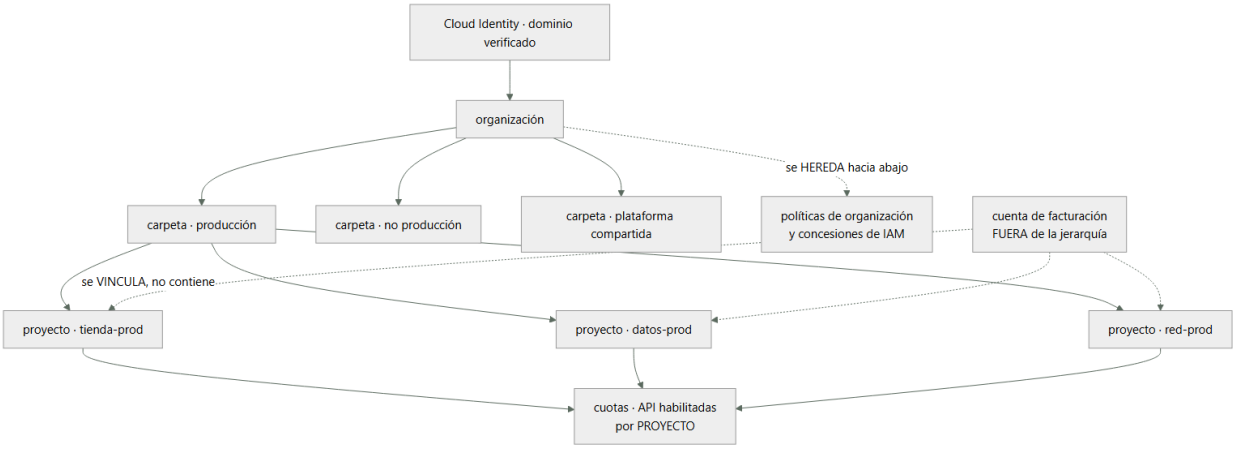
Concepto	Comprensión verificable
organización	Nodo raíz, vinculado a un dominio verificado en Cloud Identity o Workspace. Como el inquilino de Azure, la identidad precede a la jerarquía: sin dominio no hay organización.
proyecto	Unidad de API habilitadas, cuotas, IAM, red y nombres. Es barato y se crea en segundos, así que el idioma de Google Cloud es tener muchos, no pocos y grandes.
identificador de proyecto	Cadena global, única e inmutable que usan todas las API. Distinta del nombre, que se puede cambiar, y del número, que también es inmutable. Automatizar con el nombre es el error clásico.
cuenta de facturación	Objeto fuera de la jerarquía: se vincula a proyectos, no los contiene. Desvincularla no borra nada y detiene los recursos del proyecto.
cuota	Límite por proyecto y a menudo por región, de dos tipos: de tasa —por minuto— y de asignación —recursos existentes—. Ampliarlas es una solicitud con plazo, no un ajuste.
política de organización	Restricción sobre configuraciones permitidas, heredada por la jerarquía. No es IAM: acota lo que se puede configurar, con independencia de quién tenga permiso.
proyecto huérfano	Proyecto creado fuera de la organización. No hereda ninguna política, no aparece en la exportación de costos y nadie lo gobierna.

Modelo mental

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

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 proyecto es barato, y eso cambia el diseño

La primera traducción que hace todo el mundo al llegar de AWS o de Azure es esta:

```
cuenta de AWS  ≈  suscripción de Azure  ≈  proyecto de Google Cloud
```

Es correcta en lo que concentra —aislamiento, cuotas, facturación, IAM— y falla en lo que cuesta. Abrir una cuenta de AWS implica un proceso organizativo; crear una suscripción de Azure, un acuerdo comercial. Crear un proyecto es una llamada de API que tarda segundos:

```
$ gcloud projects create cloudshop-tienda-prod-01 \
    --folder $CARPETA_PROD --labels entorno=prod,sistema=tienda
$ gcloud beta billing projects link cloudshop-tienda-prod-01 \
    --billing-account $CUENTA_FACTURACION
```

Esa diferencia de costo administrativo produce un idioma distinto. En AWS y en Azure la presión es a agrupar para no multiplicar cuentas o suscripciones; en Google Cloud la presión es la contraria, y por tres razones concretas:

```
1. las cuotas son POR PROYECTO
   más proyectos = más margen, sin pedir nada a nadie
2. las API habilitadas son POR PROYECTO
   un proyecto solo tiene encendido lo que usa: menos superficie
3. borrar un proyecto borra todo lo que contiene
   la limpieza de un entorno efímero es una operación, no un inventario
```

La tercera es la que más rinde en la práctica. Un entorno de pruebas completo se destruye con una llamada, y el borrado es reversible durante 30 días antes de hacerse definitivo, lo que da un margen que ni el grupo de recursos de Azure ni la cuenta de AWS ofrecen de la misma forma.

Y hay un detalle que rompe automatizaciones el primer día: un proyecto tiene tres identificadores y solo uno sirve.

```
nombre         "CloudShop tienda producción"   legible, MUTABLE
identificador  cloudshop-tienda-prod-01        lo usan las API, INMUTABLE
                                               y ÚNICO EN GOOGLE CLOUD ENTERO
número         418293047512                    inmutable, lo usan algunos servicios
```

Dos consecuencias. La primera: cualquier automatización debe usar el identificador; usar el nombre funciona hasta que alguien lo edita. La segunda sorprende más: el identificador es único globalmente, entre todos los clientes de Google Cloud. cloudshop-prod puede estar cogido por una empresa que no conoces, y además un identificador no se libera nunca, ni siquiera al borrar el proyecto. La convención de nombres tiene que asumirlo desde el principio:

```
cls-  tienda   -prod- euw1 -01
■     ■         ■      ■     ■■■ secuencia
■     ■         ■      ■■■■■■■■■■ región principal
■     ■         ■■■■■■■■■■■■■■■■■ entorno
■     ■■■■■■■■■■■■■■■■■■■■■■■■■■■ sistema
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ prefijo corto de la organización
```

El prefijo no es estética: es lo que hace improbable la colisión global. Y el límite de 30 caracteres del identificador obliga a abreviar, así que conviene fijarlo antes de crear el primero y no a mitad del segundo entorno.

2. Carpetas, herencia y el proyecto que nadie gobierna

La jerarquía tiene cuatro niveles y hereda hacia abajo, igual que en las dos plataformas anteriores:

```
organización        vinculada a un dominio verificado
  carpeta           anidable hasta 10 niveles
    carpeta
      proyecto      la unidad real
        recurso
```

Lo que se hereda es IAM y políticas de organización, y es acumulativo: un permiso concedido en la carpeta vale para todos los proyectos que contiene. Es el mismo comportamiento aditivo de Azure RBAC de la clase 038, y aquí Google Cloud ofrece dos mecanismos que restan en vez de uno: las políticas de denegación de IAM —que la clase 050 desarrolla— y las políticas de organización, que ocupan el mismo lugar que Azure Policy allí. Con tres plataformas ya se puede enunciar el patrón: el permiso suma, y lo que resta vive en otro sistema, con otro error y otras reglas de herencia. Lo que cambia entre proveedores es cuántos sistemas hay para restar, no que hagan falta.

El criterio para diseñar las carpetas es el mismo que la clase 037 estableció y conviene repetirlo porque la tentación es la misma: se ordena por lo que necesita una política distinta, no por el organigrama. Una estructura que sirve:

```
organización
  produccion/         políticas estrictas, acceso restringido
    tienda/  datos/  red/
  no-produccion/      políticas relajadas, presupuestos bajos
    desarrollo/  pruebas/
  plataforma/         red compartida, registro central, artefactos
  aislados/           terceros, pruebas de concepto, sin datos reales
```

Las carpetas se reorganizan sin recrear proyectos —mover un proyecto de carpeta es una operación— así que equivocarse aquí es barato. Lo que no es barato es no tener organización en absoluto.

Y ese es el problema propio de Google Cloud que no existe en las otras dos: un proyecto puede crearse fuera de cualquier organización. Cualquiera con una cuenta de Google puede crear uno, vincularlo a una tarjeta y desplegar. Ese proyecto:

```
no hereda ninguna política de organización
no aparece en la exportación de facturación de la empresa
no lo ve ningún panel de seguridad corporativo
si la persona se va, puede quedar sin propietario accesible
```

Es el equivalente al «TI en la sombra» con infraestructura real dentro, y suele descubrirse por una factura o por un incidente. Las dos medidas que lo cierran:

```
# 1. solo un grupo concreto puede crear proyectos en la organización
$ gcloud organizations add-iam-policy-binding $ORG_ID \
    --member "group:plataforma@cloudshop.example" \
    --role roles/resourcemanager.projectCreator

# 2. solo identidades del dominio propio pueden recibir permisos
$ gcloud resource-manager org-policies allow \
    constraints/iam.allowedPolicyMemberDomains $ID_CLIENTE_DOMINIO \
    --organization $ORG_ID
```

La segunda es una de las políticas de mayor rendimiento de toda la plataforma: impide conceder acceso a una cuenta personal de Gmail o al dominio de otra empresa. Sin ella, un roles/editor concedido a una dirección externa es una puerta permanente que ningún inventario de usuarios de la empresa muestra.

Y la parte que no cubre ninguna política: los proyectos que ya existen fuera de la organización. Se localizan por la facturación —una cuenta de facturación corporativa vinculada a un proyecto sin organización— y se migran, que es una operación posible y con requisitos. Es el mismo patrón de la clase 046: la política guarda la puerta y no limpia lo que ya está dentro.

3. API y cuotas: dos límites que se descubren tarde

Ningún servicio existe en un proyecto hasta que se habilita su API. No hay equivalente a esto en AWS ni en Azure, y produce dos efectos, uno bueno y uno molesto.

```
$ gcloud services enable run.googleapis.com sqladmin.googleapis.com \
    secretmanager.googleapis.com --project cls-tienda-prod-euw1-01
$ gcloud services list --enabled --project cls-tienda-prod-euw1-01 | wc -l
14
```

El efecto bueno es que la superficie de un proyecto es explícita y auditable: catorce API habilitadas es una afirmación verificable sobre lo que ese proyecto puede hacer. El molesto es que un despliegue falla con SERVICEDISABLED la primera vez, y que habilitar una API tiene una propagación de hasta unos minutos — así que habilitarla y usarla en la misma ejecución de una plantilla puede fallar de forma intermitente. La solución es habilitar las API en un paso previo, no en el mismo despliegue que las usa.

Las cuotas son la segunda sorpresa, y esta cuesta calendario. Hay dos tipos y se confunden:

```
de tasa        peticiones por minuto a una API
               se reinician solas; el síntoma es 429 RESOURCE_EXHAUSTED
de asignación  cantidad de recursos que pueden existir a la vez
               CPU por región, direcciones IP, instancias, reglas de firewall
               el síntoma es un despliegue que no puede crear más
```

Y el detalle que decide el diseño: casi todas son por proyecto, y muchas además por región. Eso convierte la granularidad de proyectos en una decisión de capacidad, no solo de orden:

```
un proyecto grande con todo
  → una sola bolsa de CPU por región para todos los servicios
  → una prueba de carga de un equipo agota el margen de otro

varios proyectos por servicio
  → bolsas independientes
  → el ruido de un equipo no llega al de al lado
```

Lo que hay que anticipar es el plazo. Un aumento de cuota es una solicitud que se revisa, y puede tardar de horas a días:

```
$ gcloud compute regions describe europe-west1 \
    --format="table(quotas.metric,quotas.limit,quotas.usage)" | grep CPUS
CPUS            72.0    58.0
```

Cincuenta y ocho de setenta y dos usadas antes de una campaña que triplica el tráfico: eso no es un problema técnico, es un problema de planificación, y solo se ve mirándolo. La regla que lo evita es sencilla y casi nunca está: revisar la cuota frente al pico previsto forma parte de la preparación de cualquier evento de carga, junto con la prueba de carga misma.

Y una alerta que rinde más que muchas otras: la cuota consumida por encima del 80 %, que en Cloud Monitoring existe como métrica y avisa antes de que el despliegue falle en vez de después.

4. La facturación es otro eje, y la exportación no es retroactiva

Aquí está la asimetría estructural de esta clase, la que no tiene equivalente en las dos plataformas anteriores.

```
AWS     la cuenta pagadora está DENTRO de la organización
Azure   la suscripción ES la frontera de facturación
Google  la cuenta de facturación está FUERA de la jerarquía
        y se VINCULA a los proyectos
```

Un proyecto puede moverse de carpeta sin cambiar de cuenta de facturación, y varias organizaciones pueden compartir una. La consecuencia práctica es que la estructura de costos y la estructura de gobierno son dos diseños distintos y hay que hacer los dos.

Y hay una operación con un efecto que sorprende: desvincular la facturación de un proyecto no lo borra, pero detiene sus recursos. Las máquinas se apagan, los servicios gestionados dejan de responder. Es a la vez un mecanismo de contención de emergencia y una forma muy eficaz de tumbar producción sin darse cuenta de lo que se estaba haciendo. Merece un bloqueo organizativo y figurar en la lista de operaciones peligrosas del equipo.

La exportación de facturación a BigQuery es el mecanismo real de atribución, y tiene una propiedad que hay que conocer antes y no después:

```
$ gcloud beta billing accounts describe $CUENTA_FACTURACION
# la exportación se configura en la consola de facturación → Exportación de BigQuery

la exportación NO es retroactiva
lo que no se exportó mientras ocurría, no se puede recuperar después
```

Activarla el primer día cuesta cinco minutos y una tabla de BigQuery; activarla el tercer mes deja dos meses sin detalle, con solo los informes agregados de la consola. Es exactamente el mismo error de la clase 045 con los registros de recurso apagados: una señal que no se recoge mientras ocurre no existe.

Lo mismo vale para las etiquetas. La exportación atribuye costo por etiqueta, y solo desde el momento en que la etiqueta estaba puesta:

```
SELECT
  (SELECT value FROM UNNEST(labels) WHERE key = 'sistema') AS sistema,
  (SELECT value FROM UNNEST(labels) WHERE key = 'entorno') AS entorno,
  ROUND(SUM(cost), 2) AS costo
FROM `cls-facturacion.billing.gcp_billing_export_v1_XXXXXX`
WHERE DATE(usage_start_time) BETWEEN '2026-07-01' AND '2026-07-31'
GROUP BY sistema, entorno
ORDER BY costo DESC
```

Un recurso sin etiqueta aparece con sistema nulo, y esa fila —«coste no atribuible»— es la métrica de gobierno que importa: mientras no sea cero, hay gasto del que nadie responde. Es el mismo indicador que la clase 025 definió para AWS, con otra consulta.

Y sobre los presupuestos, una precisión que evita una expectativa falsa:

```
un presupuesto NOTIFICA; no limita el gasto
```

Exactamente igual que en AWS. Limitar de verdad exige automatizar una reacción —notificación a Pub/Sub y una función que actúe— y la única acción realmente contundente es desvincular la facturación, que apaga el proyecto entero. Es una opción legítima para un entorno de pruebas y no lo es para producción, así que el control efectivo en producción sigue siendo el de siempre: alertar pronto, atribuir bien y revisar.

5. Políticas de organización: el gobierno que no es IAM

Es la tercera vez que aparece el mismo patrón, así que ya se puede enunciar como regla general del programa:

```
AWS     política de control de servicios (SCP)
Azure   Azure Policy
Google  política de organización

las tres:  acotan CONFIGURACIONES, no identidades
           se heredan por la jerarquía
           producen un error DISTINTO al de falta de permiso
           y no arreglan lo que ya existe
```

Hay dos formas de restricción:

```
booleana   se aplica o no
           constraints/storage.publicAccessPrevention
           constraints/compute.requireOsLogin
           constraints/iam.disableServiceAccountKeyCreation
de lista   permite o deniega valores concretos
           constraints/compute.vmExternalIpAccess
           constraints/gcp.resourceLocations
           constraints/iam.allowedPolicyMemberDomains
```

La que más rinde y conviene poner el primer día es iam.disableServiceAccountKeyCreation. Las claves de cuenta de servicio son ficheros JSON con credenciales de larga duración, y son la fuga de credenciales más habitual de Google Cloud: acaban en repositorios, en portátiles y en imágenes de contenedor. La clase 050 desarrolla la alternativa; la política que impide crearlas es lo que hace que la alternativa se use de verdad.

```
$ gcloud resource-manager org-policies enable-enforce \
    constraints/iam.disableServiceAccountKeyCreation --organization $ORG_ID
$ gcloud resource-manager org-policies enable-enforce \
    constraints/storage.publicAccessPrevention --organization $ORG_ID
```

Y dos comportamientos que conviene tener claros antes de asignarlas:

La herencia se puede romper deliberadamente. Una política definida en un proyecto puede sustituir a la de la carpeta si la restricción lo permite. Eso hace posible la excepción legítima —un proyecto que sí necesita un contenedor público para recursos estáticos— y también hace posible la excepción silenciosa. La comprobación es la misma que en la clase 046: buscar dónde se ha roto la herencia y exigir que cada caso tenga motivo escrito.

```
$ gcloud asset search-all-resources --scope organizations/$ORG_ID \
    --asset-types cloudresourcemanager.googleapis.com/Project \
    --format "value(name)" | while read p; do
    gcloud resource-manager org-policies describe \
      constraints/storage.publicAccessPrevention --project "$p" 2>/dev/null \
      | grep -q "enforce: false" && echo "excepción en $p"
  done
```

No limpia lo existente. Activar publicAccessPrevention impide abrir nuevos buckets al público y no cierra los que ya lo están. Es literalmente la lección de la clase 046 en otro proveedor, lo que confirma la parte de la hipótesis que decía que los contratos se conservan: la secuencia correcta vuelve a ser inventariar, corregir y después imponer.

Y la prueba negativa, que es la única evidencia aceptable:

```
$ gsutil iam ch allUsers:objectViewer gs://cls-tienda-publico
AccessDeniedException: 412 Request violates constraint
  constraints/storage.publicAccessPrevention                    ✓
```

El código de error importa por lo mismo que en las clases 038 y 046: no dice que falte un permiso. Añadir roles no habría cambiado nada, y saber leerlo es lo que evita una hora buscando en el sistema equivocado.

Ejemplo trabajado

CloudShop abre su tercera plataforma. El equipo llega con el contrato de once filas de la clase 048 y lo aplica en dos días — y luego encuentra tres cosas que ese contrato no cubría, todas propias de Google Cloud.

Lo que se reutilizó sin discusión (y confirma la primera parte de la hipótesis):

```
la identidad precede a la jerarquía        organización desde dominio verificado
la jerarquía hereda y el permiso suma      carpetas y proyectos
el gobierno acota configuraciones          políticas de organización
la frontera de aislamiento es la de cuotas proyecto
etiquetas obligatorias para atribuir costo labels
inventariar → corregir → imponer            misma secuencia
```

Seis de las once filas quedaron resueltas en el primer día, sin aprender nada nuevo salvo el nombre.

Sorpresa 1 — la traducción «suscripción → proyecto» sale cara en una prueba de carga.

Siguiendo el modelo de Azure, se crean tres proyectos: cls-dev, cls-stage, cls-prod, con todo dentro. La primera prueba de carga previa a campaña falla al escalar:

```
$ gcloud compute instances create ... --zone europe-west1-b
ERROR: Quota 'CPUS' exceeded. Limit: 72.0 in region europe-west1.
$ gcloud compute regions describe europe-west1 \
    --format="value(quotas.filter(metric:CPUS).usage)"
71.0
```

Las cuotas son por proyecto, y en cls-prod convivían la tienda, el procesamiento por lotes y el entorno de análisis. La solicitud de aumento tardó tres días, con la campaña a cinco.

Se rehace la estructura con el idioma correcto de la plataforma:

```
proyectos                                   3              19
bolsas de CPU independientes por región     1               7
API habilitadas en el proyecto de tienda   61              14
aumento de cuota necesario                  sí             no
borrado de un entorno de pruebas       inventario a mano  gcloud projects delete
```

La fila de las API es un efecto secundario que nadie buscaba: al separar, cada proyecto solo habilita lo que usa, y sesenta y una API encendidas «por si acaso» pasaron a catorce declaradas.

Sorpresa 2 — dos meses de gasto sin poder atribuirlo.

A los dos meses, finanzas pide el costo por sistema. La consola da totales por proyecto y por servicio, y no por etiqueta.

```
exportación de facturación a BigQuery   activada el día 61
datos disponibles                       desde el día 61
datos de los días 1 al 60               solo agregados, sin detalle
```

La exportación no es retroactiva. Y las etiquetas se habían añadido en la semana 4, así que incluso desde el día 61 una parte del gasto —recursos creados antes y nunca modificados— seguía sin sistema:

```
SELECT COUNT(*) AS lineas, ROUND(SUM(cost),2) AS costo
FROM `cls-facturacion.billing.gcp_billing_export_v1_XXXXXX`
WHERE (SELECT value FROM UNNEST(labels) WHERE key='sistema') IS NULL
```

```
costo no atribuible                2.140 USD/mes       0
etiquetas obligatorias           por convención   por política y en plantilla
exportación de facturación        día 61          día 1 en la organización nueva
```

La medida de fondo no fue etiquetar: fue hacer de «costo no atribuible = 0» un indicador con dueño, revisado cada semana. Una convención se erosiona; un número que alguien tiene que explicar, no.

Sorpresa 3 — cuatro proyectos que nadie sabía que existían.

Una revisión de la cuenta de facturación encuentra gasto en proyectos que no están en la organización:

```
$ gcloud beta billing projects list --billing-account $CUENTA \
    --format="value(projectId)" > facturados.txt
$ gcloud asset search-all-resources --scope organizations/$ORG_ID \
    --asset-types cloudresourcemanager.googleapis.com/Project \
    --format="value(displayName)" > en_organizacion.txt
$ comm -23 <(sort facturados.txt) <(sort en_organizacion.txt)
cloudshop-poc-ml
cloudshop-demo-cliente
prueba-integracion-2025
sandbox-jm
```

Cuatro proyectos creados con cuentas personales, vinculados a la facturación de la empresa. Sin políticas heredadas, sin registro central, sin aparecer en ningún panel de seguridad. Uno de ellos, cloudshop-demo-cliente, tenía un bucket público con datos de prueba que resultaron ser un volcado real anonimizado a medias.

```
proyectos fuera de la organización              4            0
quién puede crear proyectos                 cualquiera   grupo plataforma
dominios permitidos en IAM                  cualquiera   solo el propio
buckets públicos                                 1            0
comprobación de proyectos no gobernados       ninguna    semanal, automática
```

Y la prueba negativa que cierra el caso:

```
$ gsutil iam ch allUsers:objectViewer gs://cls-prueba
AccessDeniedException: 412 Request violates constraint
  constraints/storage.publicAccessPrevention                              ✓
$ gcloud projects add-iam-policy-binding cls-tienda-prod-euw1-01 \
    --member "user:externo@gmail.com" --role roles/viewer
ERROR: Request violates constraint constraints/iam.allowedPolicyMemberDomains ✓
```

Resumen de la puesta en marcha:

```
proyectos                                       3            19
proyectos fuera de la organización              4             0
API habilitadas en el proyecto principal       61            14
aumentos de cuota necesarios                    1             0
costo no atribuible                      2.140 USD/mes       0
exportación de facturación                  día 61         día 1
políticas de organización aplicadas             0             6
controles con prueba negativa                   0             6
```

La lección que esta clase traslada al resto de la parte 04: seis de los once contratos de la clase 048 se reutilizaron el primer día, y las tres sorpresas fueron todas de la misma familia — cosas que en las otras plataformas eran caras y aquí son gratis, y que por eso se usan de otra manera. El proyecto barato cambia la granularidad, la creación libre de proyectos abre un agujero de gobierno que no existía, y la facturación fuera de la jerarquía obliga a diseñar dos estructuras en vez de una. Ninguna de las tres es una carencia de la plataforma: son consecuencias de una decisión de diseño distinta, y confundirlas con carencias es lo que produce una arquitectura que pelea contra su proveedor.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-04-gcp-core-platform/049-organizacion-folders-proyectos-billing-y-cuotas/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-gcp 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-gcp 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 prueba de carga falla por cuota con margen aparente en la organización	Las cuotas son por proyecto y por región, y varios servicios compartían proyecto	Reparte en más proyectos —son baratos— y revisa la cuota frente al pico previsto antes de cualquier evento de carga.
La automatización deja de funcionar tras renombrar un proyecto	Se usó el nombre en vez del identificador, que es el inmutable	Usa siempre el identificador de proyecto y fija una convención de nombres con prefijo, porque el identificador es único en todo Google Cloud.
No se puede atribuir el gasto de los primeros meses	La exportación de facturación a BigQuery no es retroactiva y las etiquetas se pusieron tarde	Actívala el primer día y convierte «costo no atribuible = 0» en un indicador con responsable.
Aparecen proyectos con gasto que no están en la organización	Cualquiera con una cuenta de Google puede crear proyectos fuera de ella	Restringe la creación a un grupo, aplica iam.allowedPolicyMemberDomains y compara periódicamente los proyectos facturados con los gobernados.
Un despliegue falla con SERVICEDISABLED de forma intermitente	La API se habilita en la misma ejecución que la usa y la propagación tarda	Habilita las API en un paso previo e independiente del despliegue que las consume.
Se activa una política de organización y siguen existiendo recursos que la incumplen	Las políticas acotan configuraciones nuevas y no corrigen lo existente	Inventaria, corrige y después impón; y busca dónde se ha roto la herencia con una excepción no documentada.

■ 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. ¿En qué se parece un proyecto a una suscripción de Azure y en qué cambia el diseño el hecho de que sea barato?

2. ¿Cuál de los tres identificadores de un proyecto debe usar la automatización, y qué implica que sea único globalmente?
3. ¿Por qué la granularidad de proyectos es una decisión de capacidad y no solo de orden?
4. ¿Qué se pierde por activar la exportación de facturación en el mes tres en vez de en el día uno?
5. ¿Qué distingue una política de organización de una concesión de IAM, y cómo lo confirma el mensaje de error?

Referencias

• Google Cloud (2025). Resource hierarchy — organización, carpetas, proyectos y herencia. <https://cloud.google.com/resource-manager/docs/cloud-platform-resource-hierarchy>

• Google Cloud (2025). Creating and managing projects — identificador, nombre, número y borrado con retención. <https://cloud.google.com/resource-manager/docs/creating-managing-projects>

• Google Cloud (2025). Working with quotas — cuotas de tasa y de asignación, por proyecto y por región. <https://cloud.google.com/docs/quotas/overview>

• Google Cloud (2025). Export Cloud Billing data to BigQuery — configuración, esquema y ausencia de retroactividad. <https://cloud.google.com/billing/docs/how-to/export-data-bigquery>

• Google Cloud (2025). Organization policy constraints — restricciones booleanas y de lista, herencia y excepciones. <https://cloud.google.com/resource-manager/docs/organization-policy/org-policy-constraints>

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

Descargar: Parte 04 en PDF · Recorrido de Google Cloud en PDF · Manual integral

Anterior	Índice	Siguiente
← 048 · Proyecto: aplicación de tres capas en Azure	Parte 04 · Programa	050 · IAM, service accounts y Workload Identity Federation →

Evaluación — 049 Organización, folders, proyectos, billing y cuotas

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-gcp 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.

050 — IAM, service accounts y Workload Identity Federation

Parte: 04 — Google Cloud: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: iam · Estado: EXECUTABLECORE

Propósito

Dominar el modelo de identidad de Google Cloud, cuya pieza central no tiene equivalente exacto en las otras dos plataformas: la cuenta de servicio es a la vez una identidad y un recurso, así que hay dos preguntas de permiso en vez de una —qué puede hacer y quién puede usarla—. De ahí sale el camino de escalada más habitual de la plataforma, la fuga de credenciales más repetida y la razón por la que aquí el privilegio mínimo se revisa con datos de uso en vez de con criterio.

Resultados de aprendizaje

Al finalizar podrás:

1. Distinguir los permisos de una cuenta de servicio de los permisos para suplantarla, y detectar la escalada que produce la segunda.
2. Sustituir claves de cuenta de servicio por identidad adjunta, suplantación y federación de identidad de carga de trabajo.
3. Acotar una federación con una condición de atributo y demostrar con una prueba negativa que la acotación funciona.
4. Aplicar políticas de denegación de IAM y condiciones con caducidad, y saber en qué orden se evalúa todo.
5. Reducir privilegios a partir del uso observado en vez de por estimación.

Conceptos centrales

Concepto	Comprensión verificable
cuenta de servicio	Identidad y recurso a la vez. Tiene permisos propios sobre otros recursos y una política propia que dice quién puede usarla. Confundir las dos caras es la causa de la escalada más común de la plataforma.
suplantación	Obtener un testigo de corta duración de una cuenta de servicio. Sustituye a las claves para personas y para automatización, y deja rastro de quién suplantó a quién.
clave de cuenta de servicio	Fichero JSON con credenciales de larga duración. Es la fuga más frecuente de Google Cloud: acaba en repositorios, portátiles e imágenes de contenedor.
rol básico	Owner, Editor y Viewer. Editor permite modificar casi todo en el proyecto, y es lo que las cuentas de servicio por defecto reciben automáticamente.
condición de atributo	Filtro sobre las afirmaciones del testigo externo en una federación. Sin ella, cualquier repositorio del proveedor externo puede obtener credenciales.
política de denegación de IAM	Regla que quita permisos y se evalúa antes que las concesiones. Es lo que resta en un modelo aditivo, y convive con las políticas de organización sin sustituirlas.

Modelo mental

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

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 de servicio tiene dos caras y solo se mira una

En AWS un rol se asume; en Azure una identidad administrada se asocia a un recurso. En Google Cloud la cuenta de servicio es las dos cosas a la vez, y de ahí sale casi todo lo interesante de esta clase:

```
cara 1 · es una IDENTIDAD
  tiene una dirección: despliegue@cls-tienda-prod.iam.gserviceaccount.com
  tiene roles sobre otros recursos: qué PUEDE HACER

cara 2 · es un RECURSO
  tiene su propia política de IAM: quién puede USARLA
```

La revisión de permisos habitual mira la primera cara —qué roles tiene cada persona— y se salta la segunda. Y la segunda es la que concede de verdad:

```
$ gcloud projects get-iam-policy cls-tienda-prod-euw1-01 \
    --flatten="bindings[].members" --filter="bindings.members:ana@cloudshop.example" \
    --format="value(bindings.role)"
roles/viewer
```

Parece que Ana solo puede leer. Pero:

```
$ gcloud iam service-accounts get-iam-policy \
    despliegue@cls-tienda-prod-euw1-01.iam.gserviceaccount.com \
    --format="value(bindings.role,bindings.members)"
roles/iam.serviceAccountTokenCreator   user:ana@cloudshop.example
```

```
$ gcloud storage ls --impersonate-service-account \
    despliegue@cls-tienda-prod-euw1-01.iam.gserviceaccount.com
# lista todo lo que puede la cuenta de despliegue, que tiene roles/editor
```

Ana tiene, en la práctica, los permisos de la cuenta de despliegue. Su rol propio es irrelevante. Los dos permisos que producen esto:

```
roles/iam.serviceAccountTokenCreator   genera testigos: acceso completo a la cuenta
roles/iam.serviceAccountUser           permite ADJUNTARLA a un recurso nuevo
                                       → crear una máquina con esa cuenta y
                                         ejecutar código con sus permisos
```

El segundo es más sutil y no menos potente: quien puede crear una máquina virtual y adjuntarle una cuenta de servicio privilegiada puede ejecutar lo que quiera con esos permisos. Por eso serviceAccountUser sobre una cuenta con Editor equivale a Editor.

La regla operativa que se deduce, y que cambia cómo se audita:

```
el permiso efectivo de una persona es la UNIÓN de:
  sus propios roles
  + los roles de TODA cuenta de servicio que pueda suplantar o adjuntar
  + (recursivamente) las que esas cuentas puedan suplantar a su vez
```

La palabra «recursivamente» no es teórica: una cadena de dos saltos —Ana suplanta a la cuenta A, que puede suplantar a la cuenta B, que es Owner— es un camino real y ninguna revisión que mire una sola tabla lo encuentra. La herramienta que lo resuelve es el analizador de políticas, que responde a la pregunta inversa: quién puede llegar a este recurso, por cualquier camino.

```
$ gcloud asset analyze-iam-policy --organization=$ORG_ID \
    --analyze-service-account-impersonation \
    --identity="user:ana@cloudshop.example"
```

2. Las cuentas por defecto ya vienen con Editor

Este es el hecho más importante de la clase para una organización recién abierta, porque el riesgo está puesto de fábrica.

Al habilitar Compute Engine o App Engine, el proyecto recibe cuentas de servicio por defecto y —salvo que se impida— se les concede automáticamente roles/editor sobre el proyecto:

```
$ gcloud projects get-iam-policy cls-tienda-prod-euw1-01 \
    --flatten="bindings[].members" --filter="bindings.role:roles/editor" \
    --format="value(bindings.members)"
serviceAccount:418293047512-compute@developer.gserviceaccount.com
```

Y esa cuenta es la que se adjunta por omisión a cualquier máquina virtual creada sin especificar otra. Es decir:

```
cualquier máquina creada sin pensar
  → ejecuta con una identidad que puede modificar casi todo el proyecto
  → una dependencia comprometida en esa máquina hereda ese poder
  → y el servicio de metadatos entrega el testigo sin pedir nada
```

La comprobación desde dentro de la máquina, que conviene hacer una vez para entender la magnitud:

```
$ curl -s -H "Metadata-Flavor: Google" \
    "http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/email"
418293047512-compute@developer.gserviceaccount.com
```

Las dos medidas, en este orden:

```
# 1. que las cuentas por defecto dejen de recibir Editor automáticamente
$ gcloud resource-manager org-policies enable-enforce \
    constraints/iam.automaticIamGrantsForDefaultServiceAccounts --organization $ORG_ID
```

```
# 2. una cuenta dedicada por carga de trabajo, con los roles mínimos
$ gcloud iam service-accounts create sa-tienda-web --project cls-tienda-prod-euw1-01
$ gcloud projects add-iam-policy-binding cls-tienda-prod-euw1-01 \
    --member "serviceAccount:sa-tienda-web@cls-tienda-prod-euw1-01.iam.gserviceaccount.com" \
    --role roles/secretmanager.secretAccessor --condition=None
```

Y hay una complicación heredada que confunde a quien depura permisos en máquinas antiguas: los ámbitos de acceso. Son un mecanismo previo a IAM que sigue existiendo en Compute Engine, y el permiso efectivo de una máquina es la intersección de ambos:

```
permiso efectivo = roles de la cuenta de servicio  ∩  ámbitos de la máquina
```

Eso produce el desconcierto clásico: la cuenta tiene el rol, la llamada falla, y añadir más roles no cambia nada porque el ámbito no lo incluye. La guía vigente es fijar el ámbito en cloud-platform y controlar exclusivamente con IAM, que es donde hay granularidad y auditoría:

```
$ gcloud compute instances create web-01 --zone europe-west1-b \
    --service-account sa-tienda-web@cls-tienda-prod-euw1-01.iam.gserviceaccount.com \
    --scopes cloud-platform
```

Y los roles básicos merecen una frase propia. Editor incluye modificar prácticamente cualquier recurso del proyecto; Owner añade gestionar los permisos, que es lo que convierte cualquier acceso en permanente. Ninguno de los tres debería aparecer en una concesión nueva. Su equivalente correcto son los roles predefinidos —hay cientos, muy específicos— y, cuando ninguno encaja, un rol personalizado:

```
$ gcloud iam roles create tiendaOperador --organization $ORG_ID \
    --permissions run.services.get,run.services.update,logging.logEntries.list \
    --stage GA
```

3. Las claves: por qué desaparecen y por qué no

Una clave de cuenta de servicio es un fichero JSON con una credencial que no caduca. Es la fuga de credenciales más frecuente de Google Cloud, y no por descuido excepcional: es que resulta cómoda.

```
{"type": "service_account", "project_id": "cls-tienda-prod-euw1-01",
 "private_key_id": "…", "private_key": "-----BEGIN PRIVATE KEY-----\n…",
 "client_email": "despliegue@cls-tienda-prod-euw1-01.iam.gserviceaccount.com"}
```

Cuatro sitios donde acaban, en orden de frecuencia: un repositorio, la máquina de alguien, una imagen de contenedor y un gestor de secretos de otro sistema desde el que se copia a mano.

Las cuatro alternativas, por caso de uso, ordenadas de mejor a peor:

```
carga dentro de Google Cloud     cuenta de servicio ADJUNTA al recurso
                                 no hay credencial: el servicio de metadatos
                                 entrega un testigo corto
persona                          suplantación con el propio inicio de sesión
                                 gcloud --impersonate-service-account
CI externo o nube ajena          federación de identidad de carga de trabajo
caso sin alternativa             clave, con caducidad y rotación automatizada
                                 y con la justificación escrita
```

La suplantación para personas cambia además lo que dice la auditoría, que es la mitad del valor:

```
con clave         el registro dice: actuó despliegue@…
                  quién la usó: se desconoce
con suplantación  el registro dice: ana@cloudshop.example actuó
                  COMO despliegue@…
```

La primera forma hace imposible responder «quién hizo esto» durante un incidente. La segunda lo responde sola.

```
$ gcloud config set auth/impersonate_service_account \
    despliegue@cls-tienda-prod-euw1-01.iam.gserviceaccount.com
$ gcloud storage ls gs://cls-tienda-facturas
```

Y el interruptor que hace que todo lo anterior se cumpla de verdad, en lugar de quedarse en una recomendación:

```
$ gcloud resource-manager org-policies enable-enforce \
    constraints/iam.disableServiceAccountKeyCreation --organization $ORG_ID
```

Con él activo, la vía cómoda deja de existir y los equipos adoptan las otras tres. Sin él, siempre habrá un caso urgente que justifique una clave más. Es el mismo argumento de la clase 046 sobre por qué una directiva vale más que una norma: el control que no se puede saltar no depende de la disciplina de nadie.

Y para las claves que ya están en circulación, el inventario primero y la prueba negativa después:

```
$ gcloud asset search-all-resources --scope organizations/$ORG_ID \
    --asset-types iam.googleapis.com/ServiceAccountKey \
    --query "NOT name:*/keys/system-managed*" --format="value(name)" | wc -l
14
```

Las gestionadas por el sistema se excluyen del recuento porque son internas y rotan solas. Las catorce restantes son las creadas por personas, y cada una necesita un destino: sustituirla, o quedar documentada con responsable y fecha.

4. Federación: la tercera vez que aparece la misma condición

La federación de identidad de carga de trabajo permite que un sistema externo —una canalización de GitHub, una carga en otra nube, un servidor propio— obtenga credenciales sin ninguna clave. Es el mismo contrato de las clases 026 y 038, así que aquí interesa lo que cambia y lo que no.

Lo que no cambia: la pieza crítica sigue siendo acotar quién puede usarla.

```
$ gcloud iam workload-identity-pools create cls-ci --location global \
    --display-name "CI de CloudShop"

$ gcloud iam workload-identity-pools providers create-oidc github \
    --workload-identity-pool cls-ci --location global \
    --issuer-uri "https://token.actions.githubusercontent.com" \
    --attribute-mapping "google.subject=assertion.sub,\
attribute.repository=assertion.repository,\
attribute.ref=assertion.ref" \
    --attribute-condition "assertion.repository == 'cloudshop/tienda' && \
assertion.ref == 'refs/heads/main'"
```

Sin --attribute-condition, cualquier repositorio de GitHub del mundo puede obtener credenciales de tu organización. Es exactamente el mismo fallo que en AWS con un sub sin acotar y en Azure con un subject vacío, y es la tercera confirmación del mismo contrato: la parte peligrosa de una federación nunca es el emisor, es el sujeto.

Lo que sí cambia es la forma de conceder, que aquí es más expresiva. En vez de asociar la federación a una única identidad, se conceden roles a un conjunto de principales:

```
$ gcloud iam service-accounts add-iam-policy-binding \
    despliegue@cls-tienda-prod-euw1-01.iam.gserviceaccount.com \
    --role roles/iam.workloadIdentityUser \
    --member "principalSet://iam.googleapis.com/projects/$NUM/locations/global/\
workloadIdentityPools/cls-ci/attribute.repository/cloudshop/tienda"
```

Ese formato permite conceder por atributo —todo lo que venga de un repositorio, o de una rama— sin enumerar identidades una a una. Es cómodo y es la misma cuerda de la que colgarse: un principalSet con un atributo demasiado amplio concede a más de lo previsto.

Y la prueba negativa, que es la única evidencia aceptable y ya se ha ejecutado en tres plataformas:

```
desde otro repositorio                  permiso denegado    ✓
desde cloudshop/tienda, rama de trabajo  permiso denegado    ✓
desde cloudshop/tienda, rama main        testigo emitido     ✓
```

Dos mecanismos más completan el modelo, y conviene situarlos:

Condiciones en las concesiones. Una concesión puede llevar una expresión que la limite en el tiempo o por nombre de recurso. Es el privilegio mínimo en el tiempo de la clase 038, más simple que la activación con aprobación de allí y suficiente para un acceso temporal:

```
$ gcloud projects add-iam-policy-binding cls-tienda-prod-euw1-01 \
    --member "user:ana@cloudshop.example" --role roles/run.admin \
    --condition "expression=request.time < timestamp('2026-08-15T00:00:00Z'),\
title=incidente-4821,description=acceso temporal"
```

La concesión caduca sola. Eso elimina la deuda de accesos que nadie retira, que es de donde salen la mitad de los permisos excesivos de cualquier organización con dos años de vida.

Políticas de denegación. Se evalúan antes que las concesiones y ganan siempre, y se adjuntan a la organización, la carpeta o el proyecto:

```
orden de evaluación
  1. políticas de denegación   ¿hay una regla que quite este permiso? → fin
  2. concesiones               ¿hay alguna que lo dé? → permitido
  3. políticas de organización ¿la configuración resultante está permitida?
```

Son la pieza que faltaba para acotar un rol amplio sin rehacerlo: negar iam.serviceAccounts.getAccessToken sobre las cuentas de producción a todo el mundo salvo a un grupo, por ejemplo. Y conviven con las políticas de organización sin sustituirlas, porque responden preguntas distintas: la denegación dice qué permisos no tienes; la política de organización dice qué configuraciones no son válidas, tengas el permiso que tengas.

5. Reducir privilegios con datos en vez de con criterio

Todas las plataformas del programa han pedido privilegio mínimo y ninguna ha ofrecido hasta ahora una forma no subjetiva de conseguirlo. Aquí sí la hay, y merece ser el cierre de la clase porque cambia la conversación.

El recomendador observa el uso real de los últimos 90 días y propone el rol ajustado:

```
$ gcloud recommender recommendations list \
    --project cls-tienda-prod-euw1-01 --location global \
    --recommender google.iam.policy.Recommender \
    --format "table(content.overview.member, content.overview.removedRole,
                    content.overview.addedRole)"

miembro                          rol retirado    rol propuesto
sa-tienda-web@…                  roles/editor    roles/run.invoker
                                                 roles/secretmanager.secretAccessor
sa-informes@…                    roles/editor    roles/bigquery.dataViewer
user:carlos@cloudshop.example    roles/owner     roles/viewer
```

Esto convierte una discusión —«¿de verdad necesitas Editor?»— en un dato: durante 90 días esa identidad usó tres permisos de los cientos que tenía. Y tiene dos límites que hay que decir en voz alta para no crear una falsa confianza:

```
1. una acción que solo ocurre una vez al trimestre no aparece en 90 días
   → aplicar la recomendación puede romper el cierre anual
2. mide lo USADO, no lo NECESARIO
   → un permiso usado por error sigue apareciendo como usado
```

La forma responsable de aplicarlo es por pasos y con vuelta atrás preparada:

```
1. aplicar en no producción y observar dos semanas
2. en producción, empezar por identidades de carga de trabajo
   (su comportamiento es más predecible que el de las personas)
3. conservar el permiso para procesos periódicos conocidos aunque
   no aparezcan en la ventana
4. registrar los errores de permiso en un panel: son la señal
   de que se recortó de más, y aparecen en minutos
```

El punto 4 es el que hace segura la operación. Un PERMISSIONDENIED es visible, inmediato y reversible; un permiso de más es invisible durante años. La asimetría favorece recortar, y esa es la razón por la que conviene hacerlo con datos y no esperar a tener certeza.

Y para diagnosticar un acceso denegado concreto, el orden que acota el problema, con la misma estructura de las clases 038 y 026:

```
# 1. ¿por qué se deniega ESTA llamada a ESTE recurso para ESTA identidad?
$ gcloud policy-intelligence troubleshoot-policy iam \
    --principal-email sa-tienda-web@cls-tienda-prod-euw1-01.iam.gserviceaccount.com \
    --resource-name //storage.googleapis.com/projects/_/buckets/cls-tienda-facturas \
    --permission storage.objects.get
```

La respuesta indica si falta la concesión, si hay una política de denegación que gana o si una condición no se cumple. Y si nada de eso explica el fallo, el error no es de IAM: es de una política de organización, y el mensaje lo dirá con otro texto. Tres plataformas, tres vocabularios y la misma pregunta: ¿en qué sistema está la regla que me impide esto?

Ejemplo trabajado

CloudShop configura la identidad de su plataforma en Google Cloud. Llega con el contrato de las clases 026 y 038 y lo aplica en un día. Después aparecen cuatro cosas que ese contrato no cubría, y tres de ellas son la misma pieza vista desde ángulos distintos: la cuenta de servicio como recurso.

Hallazgo 1 — todas las máquinas ejecutan con Editor.

La revisión inicial de permisos sobre el proyecto de tienda:

```
$ gcloud projects get-iam-policy cls-tienda-prod-euw1-01 \
    --flatten="bindings[].members" --filter="bindings.role:roles/editor" \
    --format="value(bindings.members)"
```

```
serviceAccount:418293047512-compute@developer.gserviceaccount.com
```

Seis máquinas y dos grupos de instancias usaban esa cuenta por omisión. Cualquier dependencia comprometida en cualquiera de ellas podía leer todos los secretos del proyecto, modificar la base de datos y crear recursos.

```
cuentas con roles/editor                        2               0
cuentas de servicio dedicadas                   0               7
roles por carga de trabajo                  editor      3,1 roles de media
concesiones automáticas por defecto         activas    desactivadas por política
ámbito de acceso de las máquinas           heredado      cloud-platform + IAM
```

Hallazgo 2 — Ana era «Viewer» y podía desplegar en producción.

Una auditoría de accesos cruzada con el analizador de políticas:

```
$ gcloud asset analyze-iam-policy --organization=$ORG_ID \
    --analyze-service-account-impersonation \
    --identity="user:ana@cloudshop.example" --format="value(…)"
ana@cloudshop.example → despliegue@cls-tienda-prod (tokenCreator)
                      → despliegue tiene roles/editor
```

El rol propio de Ana era roles/viewer. Su permiso efectivo era Editor sobre producción, por un camino que ninguna tabla de roles de usuario mostraba. Y no era un caso aislado:

```
personas con acceso efectivo a producción       11              3
caminos de suplantación no previstos             7              0
concesiones de tokenCreator                     11    3, con caducidad de 8 h
revisión de permisos                  tabla de roles   analizador de políticas
```

La corrección de fondo no fue quitar permisos: fue cambiar qué se revisa. Una revisión que mira solo los roles directos no puede encontrar esto.

Hallazgo 3 — catorce claves, una en un repositorio público.

```
$ gcloud asset search-all-resources --scope organizations/$ORG_ID \
    --asset-types iam.googleapis.com/ServiceAccountKey \
    --query "NOT name:*/keys/system-managed*" --format="value(name)" | wc -l
14
```

Una de ellas apareció en un repositorio público de un antiguo becario, con la cuenta informes@, que tenía roles/bigquery.dataViewer sobre el conjunto de datos de ventas. Estuvo accesible cuatro meses.

La migración, por casos de uso y en este orden:

```
canalizaciones de CI            5 claves      federación con condición de atributo
cargas en Compute y Cloud Run   6 claves      cuenta de servicio adjunta
herramientas de personas        3 claves      suplantación con el propio acceso
claves restantes                  14                  0
creación de claves nuevas      permitida      bloqueada por política
```

Y la prueba negativa de la federación, la tercera vez que se ejecuta la misma prueba en este programa:

```
desde otro repositorio                       permiso denegado    ✓
desde cloudshop/tienda, rama de trabajo      permiso denegado    ✓
desde cloudshop/tienda, rama main            testigo emitido     ✓
```

Hallazgo 4 — el recomendador dice que sobra el 96 % de los permisos.

Tras 90 días de operación con las cuentas dedicadas ya creadas:

```
identidad             permisos concedidos   usados en 90 días
sa-tienda-web                  412                 7
sa-procesador                  412                11
sa-informes                    189                 4
```

Las recomendaciones se aplican por pasos, empezando por las cuentas de carga de trabajo, y con el panel de errores de permiso vigilando:

```
permisos concedidos (suma)        1.013            34
errores PERMISSION_DENIED
  en las 2 semanas siguientes        —              3
de los cuales, recortes de más       —      2, restaurados en 20 min
el tercero                           —      un proceso trimestral no visible
                                            en la ventana de 90 días
```

El tercero es el interesante y confirma el límite que había que declarar: un cierre trimestral no aparece en una ventana de noventa días. Se restauró su permiso con una condición que lo limita a los cinco primeros días de cada trimestre.

Resumen del modelo de identidad:

```
cuentas con rol básico                          2              0
claves de cuenta de servicio                   14              0
personas con acceso efectivo a producción      11              3
caminos de suplantación no previstos            7              0
permisos concedidos (suma de las tres cuentas) 1.013           34
concesiones con caducidad                       0              6
prueba negativa de la federación               no        sí, tres casos
```

La lección que esta clase traslada al resto de la parte 04: el contrato de identidad de las partes 02 y 03 se reutilizó entero —sin secretos, federación acotada por sujeto, privilegio mínimo, prueba negativa— y aun así hubo cuatro hallazgos, porque la cuenta de servicio es un recurso además de una identidad y eso duplica las preguntas de permiso. Una auditoría que mire solo quién tiene qué rol es correcta y está incompleta: en Google Cloud hay que preguntar además a quién puede suplantar cada quien, y responderlo requiere una herramienta, no una hoja de cálculo.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-04-gcp-core-platform/050-iam-service-accounts-y-workload-identity-federation/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-gcp 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-gcp 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 persona con rol de solo lectura despliega en producción	Tiene permiso para suplantar o adjuntar una cuenta de servicio privilegiada	Audita con el analizador de políticas incluyendo suplantación, y limita serviceAccountTokenCreator y serviceAccountUser con concesiones caducas.
Una máquina virtual puede modificar casi todo el proyecto	Usa la cuenta de servicio por defecto, que recibe roles/editor automáticamente	Desactiva las concesiones automáticas por política y crea una cuenta dedicada por carga con roles mínimos.
La cuenta tiene el rol correcto y la llamada sigue fallando en una máquina antigua	Los ámbitos de acceso de Compute Engine intersecan con IAM y no incluyen esa API	Fija el ámbito en cloud-platform y controla exclusivamente con roles de IAM.
Una credencial filtrada sigue siendo válida meses después	Es una clave de cuenta de servicio, que no caduca	Sustituye por identidad adjunta, suplantación o federación, y bloquea la creación de claves con una política de organización.
Un repositorio ajeno obtiene credenciales de la organización	El proveedor de identidad federada no tiene condición de atributo	Acota con --attribute-condition sobre repositorio y rama, y verifica con las tres pruebas negativas.
Aplicar las recomendaciones de privilegio rompe un proceso trimestral	El recomendador observa 90 días y un proceso trimestral puede no aparecer	Conserva los permisos de procesos periódicos conocidos, vigila los errores de permiso y restaura con una concesión condicionada a su ventana.

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. ¿Cuáles son las dos caras de una cuenta de servicio y qué pregunta se salta una auditoría que solo mira los roles de las personas?
2. ¿Por qué serviceAccountUser sobre una cuenta con Editor equivale a tener Editor?
3. ¿Qué se concede automáticamente al habilitar Compute Engine y cómo se impide?
4. Enumera las cuatro alternativas a una clave de cuenta de servicio y en qué caso corresponde cada una.
5. ¿En qué orden se evalúan las políticas de denegación, las concesiones y las políticas de organización?

Referencias

- Google Cloud (2025). Service accounts overview — identidad y recurso, suplantación y adjunción. <https://cloud.google.com/iam/docs/service-account-overview>
- Google Cloud (2025). Best practices for using service accounts — claves, alternativas y cuentas por defecto. <https://cloud.google.com/iam/docs/best-practices-service-accounts>
- Google Cloud (2025). Workload Identity Federation — grupos, proveedores, mapeo y condición de atributo. <https://cloud.google.com/iam/docs/workload-identity-federation>
- Google Cloud (2025). IAM deny policies — reglas de denegación y orden de evaluación. <https://cloud.google.com/iam/docs/deny-overview>
- Google Cloud (2025). Role recommendations — reducción de privilegios a partir del uso observado y sus límites. <https://cloud.google.com/policy-intelligence/docs/role-recommendations-overview>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Evaluación — 050 IAM, service accounts y Workload Identity Federation

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-gcp 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.

051 — VPC global, subredes regionales, firewall y Cloud NAT

Parte: 04 — Google Cloud: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: network · Estado: EXECUTABLECORE

Propósito

Diseñar la red de Google Cloud partiendo del hecho que reordena todo lo aprendido en las clases 027 y 039: la VPC es global. Una sola red cubre todas las regiones, las subredes son regionales y dos máquinas en continentes distintos se hablan por dirección interna sin ningún emparejamiento. Eso hace desaparecer el diseño de concentrador y radios para el caso interno, y desplaza el trabajo a otros sitios: quién puede usar qué subred, cómo se etiqueta el destino de una regla de firewall, y por qué una máquina sin dirección externa aquí sí está aislada de verdad.

Resultados de aprendizaje

Al finalizar podrás:

1. Explicar qué implica que la VPC sea global y qué diseño de las partes anteriores deja de hacer falta.
2. Elegir entre VPC compartida, emparejamiento y red por proyecto según quién debe gobernar el firewall.
3. Escribir reglas de firewall dirigidas por cuenta de servicio en vez de por etiqueta, y justificar por qué.
4. Habilitar el acceso privado a las API de Google y distinguir su fallo del fallo silencioso de otras plataformas.
5. Dimensionar la asignación de puertos de Cloud NAT antes de que un servicio locuaz la agote.

Conceptos centrales

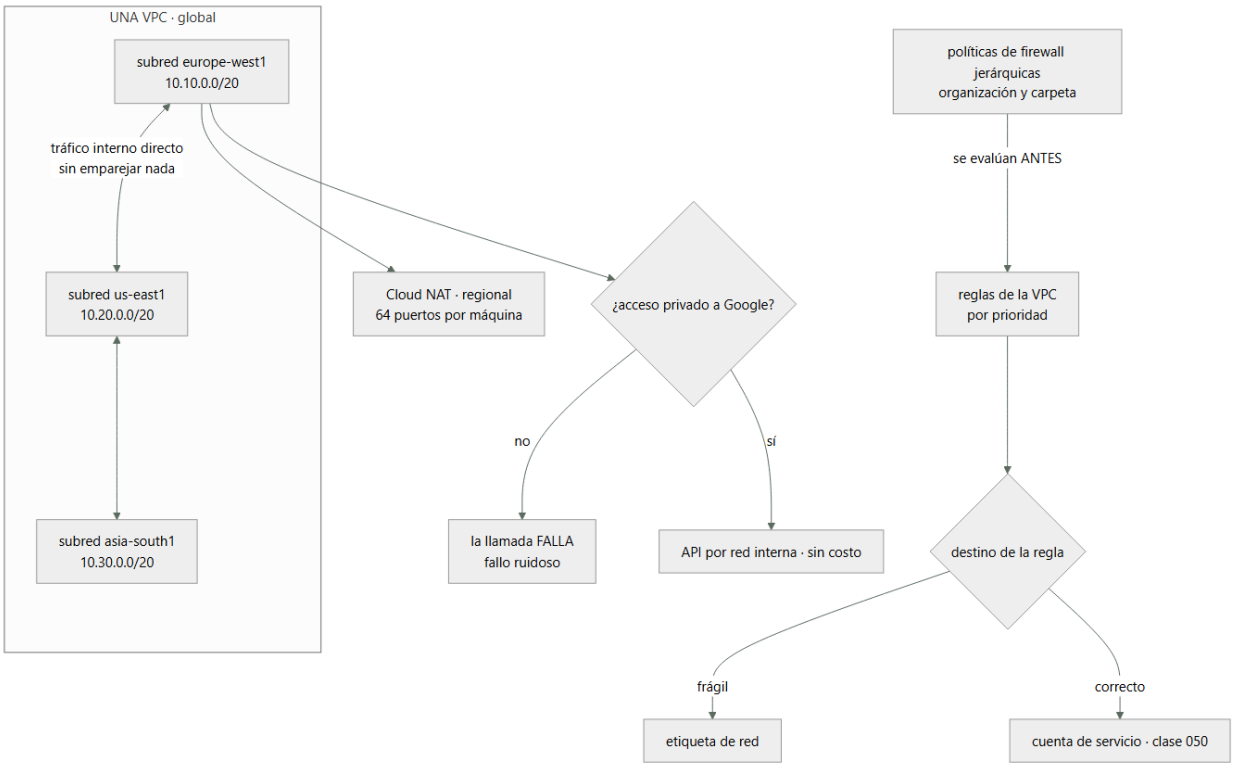
Concepto	Comprensión verificable
VPC global	Red que abarca todas las regiones. Las subredes son regionales y el enrutamiento entre ellas es interno y automático: no hay emparejamiento ni pasarela entre regiones de la misma VPC.
modo automático frente a personalizado	La red por defecto crea una subred en cada región con rangos predefinidos que coinciden con los de todo el mundo. Eso impide emparejar más adelante. La única opción defendible es el modo personalizado.
destino de regla de firewall	A qué máquinas aplica una regla: a todas, a las que tengan una etiqueta de red o a las que ejecuten con una cuenta de servicio. La etiqueta la puede poner cualquiera que edite la máquina; la cuenta de servicio, no.
VPC compartida	Un proyecto anfitrión posee la red y otros proyectos despliegan dentro de ella. No es un emparejamiento: es la misma red, con IAM decidiendo quién puede usar cada subred.
acceso privado a Google	Ajuste por subred que permite a una máquina sin dirección externa alcanzar las API de Google por la red interna. Sin él, la llamada no funciona — y eso es una buena noticia.
asignación de puertos de Cloud NAT	Puertos de traducción reservados por máquina, 64 por defecto. Es el mismo agotamiento de las clases 039 y 043, con un tercer mecanismo y la misma señal.

Modelo mental

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

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 VPC es global, y eso borra un diseño entero

En AWS una VPC vive en una región y sus subredes en zonas. En Azure, una red virtual vive en una región y conectar dos exige emparejarlas. En Google Cloud:

```
la VPC es GLOBAL: abarca todas las regiones
la subred es REGIONAL: no zonal
```

Dos consecuencias inmediatas. La primera: dos máquinas en regiones distintas de la misma VPC se hablan por dirección interna sin configurar nada. No hay emparejamiento, no hay pasarela, no hay tabla de rutas que tocar. La segunda: una subred cubre todas las zonas de su región, así que la alta disponibilidad zonal no obliga a repartir direcciones entre subredes.

El efecto práctico sobre lo construido en la clase 039 es que un diseño entero deja de hacer falta:

```
Azure    concentrador y radios, emparejamientos bidireccionales,
         allowForwardedTraffic, UDR por radio, tránsito facturado dos veces
Google   una VPC, subredes por región, y ya está
```

Eso no significa que no haya nada que diseñar: significa que el trabajo se desplaza. Lo que en Azure se resolvía con topología, aquí se resuelve con IAM sobre la red, y esa es la parte que sorprende a quien llega esperando dibujar cajas.

El segundo hecho estructural es la red por defecto. Todo proyecto nuevo recibe una VPC en modo automático: una subred en cada región, con rangos predefinidos dentro de 10.128.0.0/9. Son los mismos rangos para todos los clientes de Google Cloud, así que:

```
dos proyectos con red automática NO se pueden emparejar: sus rangos coinciden
tampoco se puede emparejar con la red de un socio que usara la automática
y descubrirlo es tarde: hay que recrear la red y mover todo lo que hay dentro
```

La medida se toma antes de crear el primer proyecto, y es una política de organización de la clase 049:

```
$ gcloud resource-manager org-policies enable-enforce \
    constraints/compute.skipDefaultNetworkCreation --organization $ORG_ID
```

```
$ gcloud compute networks create vpc-cloudshop --subnet-mode custom
$ gcloud compute networks subnets create snet-tienda-euw1 \
    --network vpc-cloudshop --region europe-west1 --range 10.10.0.0/20 \
    --enable-private-ip-google-access --enable-flow-logs
```

Y el reparto de direcciones se planifica con la misma disciplina de las clases 027 y 039, con dos particularidades: Google reserva cuatro direcciones por subred en lugar de cinco, y una subred se puede ampliar en caliente —cambiar su máscara a una mayor sin recrearla—, lo que no significa que se pueda improvisar, porque reducirla sigue sin ser posible.

```
vpc-cloudshop (global)
  snet-tienda-euw1      10.10.0.0/20    europe-west1
  snet-datos-euw1       10.10.16.0/20   europe-west1
  snet-tienda-use1      10.20.0.0/20    us-east1
  rango secundario para contenedores    10.60.0.0/14   (parte 06)
```

La última línea conviene reservarla ya: los clústeres de Kubernetes en Google Cloud consumen rangos secundarios de la subred para pods y servicios, y son grandes. Descubrirlo en la parte 06 con el espacio ya repartido obliga a rehacer lo de arriba.

2. El firewall es de la red, y el destino correcto es una identidad

Las reglas de firewall en Google Cloud no cuelgan de la subred ni de la interfaz: cuelgan de la VPC, y se aplican a un conjunto de máquinas definido por el propio destino de la regla. Son con estado, tienen prioridad —número menor gana— y se evalúan hasta la primera coincidencia.

```
reglas implícitas, que no se pueden borrar
  entrada  denegar todo         prioridad 65535
  salida   permitir todo        prioridad 65535
```

Comparado con las otras dos plataformas, esto ya es una diferencia notable: la entrada está denegada por defecto entre máquinas de la misma red, sin la regla AllowVnetInBound que en Azure dejaba hablar a todo con todo (clase 039). La red por defecto en modo automático sí trae una regla que permite el tráfico interno, y es otra razón para no usarla.

La decisión que de verdad importa es a quién aplica la regla, y hay tres formas:

Destino	Cómo se asigna	Quién puede cambiarlo
Todas las instancias de la red	—	—
Etiqueta de red	Cadena en la máquina	Cualquiera que pueda editar la máquina
Cuenta de servicio	La identidad con la que ejecuta	Solo quien tenga serviceAccountUser sobre ella

La fila del medio es la trampa. Una etiqueta de red es texto libre: quien tenga permiso para modificar una instancia puede añadirle permitir-ssh y quedar dentro del ámbito de la regla que abre el puerto 22. No hace falta tocar el firewall para abrir un puerto — basta con etiquetarse.

Dirigir por cuenta de servicio cierra ese camino y encadena con la clase 050: cambiar la identidad de una máquina exige permiso sobre esa cuenta de servicio, que es un permiso auditado y restringido.

```
$ gcloud compute firewall-rules create permitir-app-a-datos \
    --network vpc-cloudshop --direction INGRESS --priority 1000 --action ALLOW \
    --rules tcp:5432 \
    --source-service-accounts sa-tienda-web@cls-tienda-prod-euw1-01.iam.gserviceaccount.com \
    --target-service-accounts sa-datos@cls-datos-prod-euw1-01.iam.gserviceaccount.com
```

Esa regla dice exactamente lo que se quiere decir: el servicio web puede hablar con la base de datos por el 5432, sin mencionar direcciones ni depender de que nadie ponga la etiqueta correcta. Y sobrevive a que las máquinas cambien de dirección, de zona o de región.

Una limitación que hay que conocer: una regla no puede mezclar etiquetas y cuentas de servicio como origen y destino. La migración es por bloques, no gradual dentro de la misma regla.

Por encima de las reglas de la VPC están las políticas de firewall jerárquicas, definidas en la organización o en la carpeta y evaluadas antes:

```
orden de evaluación
  1. política jerárquica de la organización
```

```
  2. política jerárquica de la carpeta
  3. reglas de la VPC
cada nivel puede permitir, denegar o ceder al siguiente con goto_next
```

Es el mismo patrón de gobierno de las clases 046 y 049 aplicado a la red: una regla que ningún equipo puede saltarse, como denegar el 22 y el 3389 desde internet en toda la organización. Y con la misma advertencia de siempre: no cierra lo que ya está abierto por otra vía, solo impide que las reglas de abajo lo permitan.

Y el registro de reglas de firewall, que conviene activar en las que importan aunque tenga costo, porque responde la pregunta que aparece en todo incidente de conectividad —qué regla decidió— sin tener que deducirla:

```
$ gcloud compute firewall-rules update permitir-app-a-datos --enable-logging
```

3. VPC compartida: la red la gobierna quien debe, no quien despliega

Con una VPC global y proyectos baratos (clase 049) aparece una pregunta que en las otras plataformas no se planteaba así: si cada equipo tiene sus propios proyectos, ¿cada uno tiene su propia red?

Las tres respuestas posibles y cuándo vale cada una:

	Cómo funciona	Cuándo
Una red por proyecto	Cada proyecto con su VPC	Aislamiento fuerte, sin tráfico interno entre equipos
Emparejamiento de VPC	Redes distintas conectadas	Dos organizaciones, o un socio
VPC compartida	Un proyecto anfitrión posee la red; los demás despliegan en ella	El caso normal dentro de una organización

La VPC compartida no es un emparejamiento: es la misma red, usada desde varios proyectos. El proyecto anfitrión posee las subredes, las reglas de firewall y las rutas; los proyectos de servicio crean máquinas, balanceadores y clústeres dentro de esas subredes. Y el reparto se hace con IAM:

```
$ gcloud compute shared-vpc enable cls-red-prod
$ gcloud compute shared-vpc associated-projects add cls-tienda-prod-euw1-01 \
    --host-project cls-red-prod

# el equipo de tienda solo puede usar SU subred, no todas
$ gcloud compute networks subnets add-iam-policy-binding snet-tienda-euw1 \
    --region europe-west1 --project cls-red-prod \
    --member "group:equipo-tienda@cloudshop.example" \
    --role roles/compute.networkUser
```

Esa última concesión es la pieza elegante del modelo: el permiso de red se concede por subred, así que un equipo puede desplegar en la suya y no en la de datos, sin que nadie tenga que dibujar una topología para conseguirlo. Lo que en Azure se resolvía separando redes y emparejándolas, aquí se resuelve con una concesión.

La contrapartida hay que aceptarla porque define el modelo operativo: las reglas de firewall las gestiona el proyecto anfitrión. Un equipo que necesita abrir un puerto no puede hacerlo solo. Eso es exactamente lo que se quiere en una organización con requisitos de seguridad, y es un cuello de botella si el equipo de red tarda días en atender una petición. La respuesta razonable no es repartir el permiso, sino automatizar la petición: las reglas se declaran en código en el repositorio del proyecto anfitrión y se aprueban por revisión, que es el flujo de la clase 059.

El emparejamiento de VPC queda entonces para lo que de verdad lo necesita —otra organización, un proveedor— y trae las restricciones conocidas:

```
no es transitivo        A ↔ B ↔ C no da A ↔ C
rangos sin solapar      por eso importaba no usar la red automática
cuota de rutas          las rutas aprendidas cuentan contra un límite
firewall independiente  cada lado decide qué acepta
```

Y para el caso de muchas redes que sí necesitan hablarse entre sí, existe el centro de conectividad de red, que resuelve la transitividad con un concentrador gestionado — el mismo problema de la clase 039 con una pieza distinta. Conviene saber que existe y no llegar a él por costumbre: con una VPC global bien diseñada, la mayoría de organizaciones no lo necesitan.

4. Salir a internet y llegar a Google: dos caminos distintos

Aquí está la asimetría propia de esta plataforma, y va en la dirección contraria a la de Azure.

Una máquina sin dirección IP externa no alcanza internet. La ruta por defecto hacia la pasarela de internet existe en la tabla, y sin dirección externa y sin NAT el paquete no llega a ningún sitio. Comparado con lo aprendido en la clase 039:

```
Azure    existía una traducción implícita: la "subred privada" salía igual
Google   no hay traducción implícita: sin dirección externa y sin NAT, no sale
```

Es decir: el aislamiento es el estado por defecto, y es la buena noticia de esta clase. Lo que hay que declarar aquí es lo contrario que allí — no cómo cerrar la salida, sino cómo abrirla cuando hace falta.

Cloud NAT es esa salida, y es regional y sin instancias que gestionar:

```
$ gcloud compute routers create rt-euw1 --network vpc-cloudshop --region europe-west1
$ gcloud compute routers nats create nat-euw1 --router rt-euw1 --region europe-west1 \
    --nat-all-subnet-ip-ranges --auto-allocate-nat-external-ips \
    --enable-dynamic-port-allocation --min-ports-per-vm 64 --max-ports-per-vm 2048 \
    --enable-logging --log-filter ERRORS_ONLY
```

Dos parámetros de esa orden merecen atención. El primero es el precio, que tiene la misma forma que en las otras dos plataformas y por tanto la misma lección:

```
~0,044 USD/h por la pasarela  +  ~0,045 USD por GB procesado
```

El segundo es la asignación de puertos: por defecto, 64 por máquina. Un servicio que abre muchas conexiones cortas al mismo destino los agota, y el síntoma es el mismo de las clases 039 y 043 —tiempos de espera intermitentes hacia un destino concreto mientras el resto funciona—. Es la tercera vez que aparece este problema en el programa, con un tercer mecanismo:

```
AWS     puertos del NAT gateway, gestionados por el servicio
Azure   64.000 repartidos entre instancias, o NAT Gateway bajo demanda
Google  64 por máquina por defecto, ampliables y con asignación dinámica
```

La asignación dinámica es la respuesta correcta: reparte entre un mínimo y un máximo según la necesidad, en vez de reservar por adelantado. Y el registro con ERRORSONLY da la señal exacta —OUTOFRESOURCES— en vez de dejar deducirlo desde la aplicación.

Llegar a las API de Google es otro camino y no pasa por el NAT. Cloud Storage, BigQuery, Secret Manager y el resto viven en direcciones públicas, así que una máquina sin dirección externa no las alcanza — salvo que la subred tenga habilitado el acceso privado a Google:

```
$ gcloud compute networks subnets update snet-datos-euw1 --region europe-west1 \
    --enable-private-ip-google-access
```

Es un ajuste de subred, gratuito, y evita pagar el NAT por ese tráfico, que suele ser la mayor parte. La aritmética es la misma de la clase 027 y vuelve a salir a cuenta:

```
2,4 TB/mes hacia Cloud Storage por Cloud NAT   2.400 × 0,045 = 108 USD/mes
los mismos 2,4 TB con acceso privado a Google                   0 USD/mes
```

Y lo más valioso para quien viene de la clase 039: cuando falta, falla. Sin acceso privado a Google, la llamada agota su tiempo de espera y aparece en el registro. No hay un camino público alternativo por el que funcione mientras el diagrama afirma que es privado. Comparar los dos fallos es la lección portable:

```
Azure sin zona DNS privada    funciona por el camino público → fallo SILENCIOSO
Google sin acceso privado     no funciona                    → fallo RUIDOSO
```

Un fallo ruidoso se arregla en diez minutos. Uno silencioso dura cuatro meses.

Para el resto de casos —una API concreta con una dirección interna elegida, o un servicio de un tercero— está Private Service Connect, que crea un extremo con IP interna propia y su entrada de DNS. Es el análogo del punto de conexión privado de Azure y del endpoint de interfaz de AWS, y trae de vuelta la parte de DNS: si el nombre no resuelve al extremo, el tráfico busca el camino de siempre. La comprobación es la misma que se hizo en la clase 039, y hay que hacerla igual.

5. Diagnóstico y evidencia de una red que no se ve

Con la topología reducida a una VPC, el diagnóstico se apoya menos en el diagrama y más en tres herramientas que conviene conocer antes del incidente.

Pruebas de conectividad, que simulan un paquete sin enviarlo y dicen qué lo permitiría o lo bloquearía:

```
$ gcloud network-management connectivity-tests create prueba-app-datos \
    --source-instance web-01 --destination-instance db-01 \
    --destination-port 5432 --protocol TCP
$ gcloud network-management connectivity-tests describe prueba-app-datos \
    --format="value(reachabilityDetails.result)"
REACHABLE
```

Es la respuesta a «¿está abierto?» sin tener que abrir una sesión en ninguna máquina, y funciona igual antes de desplegar que durante una caída. Que además se puedan guardar y reejecutar la convierte en el equivalente de red del guion de pruebas negativas de la clase 046:

```
desde la subred de tienda a la base de datos por 5432   REACHABLE     ✓
desde la subred de tienda a la base de datos por 22     UNREACHABLE   ✓
desde internet a cualquier máquina por 22               UNREACHABLE   ✓
```

Registros de flujo de VPC, activados por subred, con muestreo configurable. Responden a quién habló con quién y cuánto, que es la pregunta de la investigación posterior. Su costo depende del muestreo, así que la decisión es la misma de la clase 045: se activan al 100 % donde hay datos sensibles y con muestreo bajo donde el volumen es alto.

Espejo de paquetes, para cuando hace falta el contenido y no el resumen. Es caro y es la única forma de responder ciertas preguntas; conviene tenerlo previsto y apagado.

Y el orden de diagnóstico, que con tres plataformas ya se puede escribir como método y no como receta de proveedor:

```
1. ¿existe camino?        prueba de conectividad · ruta y firewall
2. ¿resuelve el nombre?   la respuesta debe ser una dirección interna
3. ¿tiene permiso?        IAM, y si no, política de organización (049, 050)
4. ¿lo permite el destino? firewall del otro lado, o su propia configuración
```

Los cuatro pasos son idénticos en AWS, Azure y Google Cloud. Lo único que cambia es el nombre del comando y el texto del error — y esa es exactamente la afirmación que la clase 048 dejó por comprobar.

Ejemplo trabajado

CloudShop lleva a Google Cloud la red que diseñó en la clase 039. El primer día descubre que la mitad del diseño sobra, y las tres semanas siguientes descubre dónde estaba el trabajo de verdad.

Lo que desapareció el primer día.

El diseño de Azure tenía un concentrador, tres radios, seis emparejamientos y una tabla de rutas por radio. Al trasladarlo:

```
redes virtuales / VPC                          4                1
emparejamientos                                6                0
tablas de rutas definidas por el usuario       3                0
costo mensual de tránsito entre radios    50,40 USD         0 USD
latencia entre servicios de regiones distintas  vía concentrador  directa
```

Los 50,40 USD al mes de la clase 039 —de los cuales 32,40 eran el precio de inspeccionar— desaparecen para el tráfico interno. Lo que no desaparece es la necesidad de inspeccionar: se traslada a las reglas de firewall y a las políticas jerárquicas, que no cuestan por GB.

Incidente 1 — no se puede emparejar con la red del socio logístico.

```
$ gcloud compute networks peerings create socio --network default \
    --peer-project socio-logistico --peer-network default
ERROR: IP range 10.128.0.0/9 overlaps with peer network range
```

Ambas partes usaban la red por defecto en modo automático, con los mismos rangos. No hay ajuste que lo resuelva: hay que recrear la red y mover lo que hay dentro.

```
modo de la red                      automático      personalizado
rangos                           10.128.0.0/9      10.10.0.0/16 planificado
redes por defecto en proyectos nuevos  se crean   bloqueadas por política
máquinas que hubo que recrear             9              —
tiempo de la migración                 2 días            —
```

Dos días de trabajo por una decisión que costaba un minuto antes de empezar.

Incidente 2 — un puerto 22 abierto sin tocar el firewall.

Una revisión encuentra una máquina de producción accesible por SSH desde una subred de desarrollo.

```
$ gcloud compute instances describe api-03 --zone europe-west1-b \
    --format="value(tags.items)"
permitir-ssh-dev
```

Nadie modificó ninguna regla: alguien añadió una etiqueta a la máquina y quedó dentro del ámbito de una regla existente. El permiso necesario para hacerlo era el de editar instancias, que tenían nueve personas.

```
reglas dirigidas por etiqueta de red           14                0
reglas dirigidas por cuenta de servicio         0               14
quién puede cambiar el ámbito de una regla   9 personas   solo con permiso
                                                          sobre la cuenta (050)
política jerárquica que niega 22 y 3389
  desde internet                              ninguna       en la organización
```

Y la prueba negativa correspondiente:

```
prueba de conectividad: internet → api-03 : 22    UNREACHABLE   ✓
prueba de conectividad: dev → api-03 : 22         UNREACHABLE   ✓
prueba de conectividad: tienda → datos : 5432     REACHABLE     ✓
```

Incidente 3 — el procesador no puede leer de Cloud Storage, y se arregla en once minutos.

Al quitar las direcciones externas de las máquinas de proceso, el trabajo nocturno falla:

```
google.api_core.exceptions.RetryError: Deadline of 60.0s exceeded
  while calling storage.googleapis.com
```

```
$ gcloud compute networks subnets describe snet-datos-euw1 --region europe-west1 \
    --format="value(privateIpGoogleAccess)"
False
$ gcloud compute networks subnets update snet-datos-euw1 --region europe-west1 \
    --enable-private-ip-google-access
```

Once minutos desde el fallo hasta la corrección, porque el fallo fue ruidoso. El equipo anota la comparación con lo ocurrido en Azure, donde el mismo error conceptual —el camino privado no configurado— tardó cuatro meses en detectarse porque todo seguía funcionando por el camino público.

Y el efecto en la factura:

```
tráfico hacia API de Google por NAT     2,4 TB/mes         0
costo de ese tráfico                    108 USD/mes        0
tráfico restante por NAT                0,8 TB/mes      0,8 TB/mes
costo total de Cloud NAT                176 USD/mes      68 USD/mes
```

Incidente 4 — tiempos de espera hacia la pasarela de pago en los picos.

El mismo síntoma de la clase 043, con otro mecanismo debajo:

```
$ gcloud logging read 'resource.type="nat_gateway" AND jsonPayload.allocation_status="DROPPED"' \
    --limit 5 --format="value(jsonPayload.reason)"
OUT_OF_RESOURCES
```

Sesenta y cuatro puertos por máquina, y un cliente HTTP que abría conexión por petición.

```
asignación de puertos                 64 fijos     64-2048 dinámica
cliente HTTP                       uno por petición  uno por proceso
descartes por falta de recursos      1.412/hora           0
fallos hacia la pasarela               2,7 %            0,0 %
```

Dos correcciones a la vez, y la segunda es la que de verdad importa: el mismo defecto de código ha producido el mismo incidente en tres plataformas distintas, con tres mecanismos de traducción diferentes. Es la definición operativa de un problema portable.

Incidente 5 — cuatro redes en la sombra.

Con proyectos baratos, tres equipos habían creado sus propias VPC para no depender del equipo de red:

```
$ gcloud asset search-all-resources --scope organizations/$ORG_ID \
    --asset-types compute.googleapis.com/Network --format="value(name)" | wc -l
5
```

Cinco redes donde debía haber una, sin conectividad entre ellas y con reglas de firewall que nadie revisaba. Se migra a VPC compartida:

```
VPC en la organización                       5              1
```

```
quién gestiona las reglas de firewall      cada equipo    proyecto anfitrión,
                                                          por revisión de código
permiso de red                             completo por   por SUBRED, con
                                            proyecto      roles/compute.networkUser
tiempo medio de una petición de regla         —          4 h (revisión + despliegue)
reglas de firewall sin revisar                31               0
```

La fila del tiempo medio es la que se negoció explícitamente: cuatro horas es aceptable y cuatro días no lo sería. Sin ese compromiso, los equipos vuelven a crear redes propias.

Resumen de la red:

```
redes                                            4                 1
emparejamientos                                  6                 0
costo de tránsito interno                  50,40 USD/mes         0
costo de salida (NAT)                      68,85 USD/mes     68 USD/mes
reglas dirigidas por identidad                   —              14 de 14
fallo del camino privado                    silencioso         ruidoso
controles con prueba de conectividad guardada    0                 6
```

La lección que esta clase traslada al resto de la parte 04: la VPC global elimina un diseño entero y no elimina el trabajo — lo mueve de la topología a la identidad. Y la comparación entre el fallo silencioso de Azure y el ruidoso de Google Cloud es la más valiosa de la clase: entre dos plataformas que fallan, es mejor la que falla ruidosamente, y esa propiedad casi nunca aparece en una tabla comparativa de servicios.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-04-gcp-core-platform/051-vpc-global-subredes-regionales-firewall-y-cloud-nat/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 vpc-gcp 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 vpc-gcp 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
No se puede emparejar con la red de otra organización	Ambas usan la red por defecto en modo automático, con rangos idénticos	Bloquea la creación de la red por defecto con una política de organización y crea siempre VPC en modo personalizado con rangos planificados.
Un puerto queda abierto en producción sin que nadie modifique el firewall	La regla se dirige por etiqueta de red y cualquiera que edite la máquina puede ponérsela	Dirige las reglas por cuenta de servicio y añade una política de firewall jerárquica que niegue los puertos de administración desde internet.
Una máquina sin dirección externa no alcanza Cloud Storage	La subred no tiene habilitado el acceso privado a Google	Habilítalo en la subred: es gratuito y además evita pagar Cloud NAT por ese tráfico.
Tiempos de espera intermitentes hacia un destino externo concreto	Se agotan los 64 puertos por máquina que Cloud NAT asigna por defecto	Activa la asignación dinámica de puertos, revisa el registro con ERRORSONLY y reutiliza el cliente HTTP en la aplicación.
Aparecen varias VPC creadas por equipos distintos sin conectividad entre ellas	Los proyectos son baratos y crear una red propia es más rápido que pedir una regla	VPC compartida con permiso por subred, y un compromiso de tiempo de respuesta para las peticiones de firewall declaradas en código.
El espacio de direcciones no da para desplegar un clúster de Kubernetes	No se reservaron rangos secundarios para pods y servicios, que son grandes	Reserva los rangos secundarios al planificar la subred, antes de necesitarlos.

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 implica que la VPC sea global y qué parte del diseño de la clase 039 deja de ser necesaria?
2. ¿Por qué dirigir una regla de firewall por cuenta de servicio es más seguro que por etiqueta de red?
3. ¿Qué diferencia hay entre VPC compartida y emparejamiento, y qué gobierna cada una?
4. Una máquina sin IP externa no alcanza una API de Google. ¿Qué falta y por qué este fallo es preferible al equivalente de Azure?
5. ¿Cuál es la tercera versión del agotamiento de puertos de traducción que aparece en el programa y cómo se corrige aquí?

Referencias

- Google Cloud (2025). VPC network overview — alcance global, subredes regionales y rutas. <https://cloud.google.com/vpc/docs/vpc>
- Google Cloud (2025). VPC firewall rules — prioridad, reglas implícitas y destinos por etiqueta o cuenta de servicio. <https://cloud.google.com/firewall/docs/firewalls>
- Google Cloud (2025). Shared VPC overview — proyecto anfitrión, proyectos de servicio y permiso por subred. <https://cloud.google.com/vpc/docs/shared-vpc>
- Google Cloud (2025). Private Google Access — acceso a las API desde máquinas sin dirección externa. <https://cloud.google.com/vpc/docs/private-google-access>
- Google Cloud (2025). Cloud NAT overview — asignación de puertos, asignación dinámica y registro. <https://cloud.google.com/nat/docs/overview>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Evaluación — 051 VPC global, subredes regionales, firewall y Cloud NAT

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 vpc-gcp 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.

052 — Compute Engine, managed instance groups y load balancing

Parte: 04 — Google Cloud: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: compute · Estado: EXECUTABLECORE

Propósito

Dimensionar cómputo y reparto de carga en Google Cloud, donde tres cosas funcionan al revés de lo aprendido: el descuento por uso sostenido se aplica solo, sin comprar nada; la máquina se puede definir con el número exacto de vCPU y de memoria que hace falta; y el balanceador global tiene una sola dirección IP para todo el planeta, así que la conmutación entre regiones no depende del DNS ni de su TTL. A cambio, reaparece por tercera vez el problema de los dos límites de disco y aparece una forma nueva de tumbar un servicio: la reparación automática.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir tipo de máquina, incluido uno personalizado, a partir del uso medido en vez del catálogo.
2. Calcular el costo real considerando el descuento automático por uso sostenido antes de comprometer nada.
3. Separar la comprobación de estado del balanceador de la de reparación automática, y explicar qué rompe mezclarlas.
4. Configurar un balanceador global con modo de equilibrio por capacidad y medir su conmutación entre regiones.
5. Decidir entre disco zonal y disco regional a partir del RTO exigido ante la caída de una zona.

Conceptos centrales

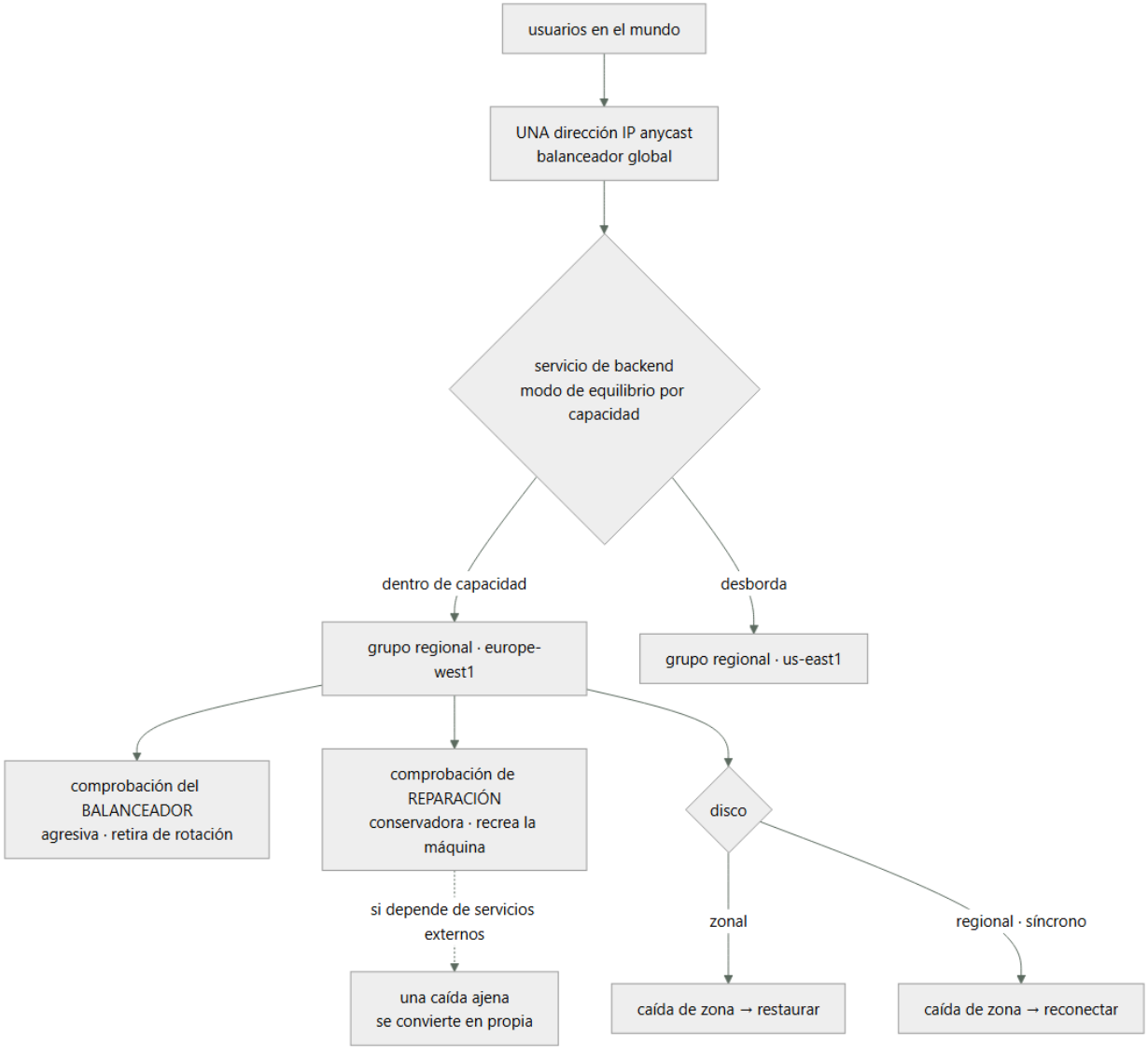
Concepto	Comprensión verificable
tipo de máquina personalizado	Definición con el número exacto de vCPU y de memoria, dentro de unas proporciones permitidas. Elimina el redondeo al escalón siguiente, que en las otras plataformas es obligatorio.
descuento por uso sostenido	Reducción automática y sin compromiso que crece con el porcentaje del mes que la máquina está encendida, hasta cerca del 30 %. No hay que comprar nada ni pedir nada.
migración en vivo	Traslado transparente de una máquina en ejecución durante el mantenimiento del anfitrión. Hace innecesario el concepto de dominio de actualización de la clase 040 — salvo en excedente y algunos tipos, que sí terminan.
plantilla de instancia	Definición inmutable de cómo es una instancia. Cambiar algo obliga a crear otra plantilla y ejecutar una actualización progresiva, lo que hace que cada cambio quede versionado.
reparación automática	Sustitución de instancias que fallan su comprobación de estado. Con una comprobación que depende de servicios externos, convierte una caída ajena en una caída propia.
modo de equilibrio por capacidad	Declaración de cuántas peticiones por segundo aguanta cada instancia. El balanceador global desborda a la región siguiente al superarla: así conmuta sin tocar el DNS.
disco persistente regional	Disco replicado de forma síncrona entre dos zonas. Permite reconectar el volumen en la otra zona en minutos, sin restaurar nada.

Modelo mental

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

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 máquina a medida y el descuento que no hay que pedir

La nomenclatura es la más simple de las tres plataformas y no esconde letras con significado:

```
n2 - standard - 8
■    ■          ■■■ vCPU
■    ■■■■■■■■■■■■■■■ perfil de memoria: standard (4 GB/vCPU),
■                   highmem (8), highcpu (2)
■■■■■■■■■■■■■■■■■■■■■ familia

E2   propósito general económico; en tamaños pequeños comparte núcleo
N2 / N2D   propósito general, Intel y AMD
C3 / C4    cómputo, la generación con mejor rendimiento por núcleo
M3         memoria extrema
A3 / G2    aceleradores
T2D        rendimiento por precio para cargas escalables
```

Y una capacidad que ninguna de las otras dos ofrece: el tipo personalizado. Si el servicio usa 6 vCPU y 50 GB, no hay que comprar el escalón de 16 vCPU y 64 GB para llegar a la memoria:

```
$ gcloud compute instances create app-01 --zone europe-west1-b \
    --custom-cpu 8 --custom-memory 52GB --custom-vm-type n2
```

Las proporciones tienen límites —hay un mínimo y un máximo de memoria por vCPU—, y dentro de ellos el ajuste es exacto. En una flota mediana, dejar de pagar el redondeo es de los ahorros más grandes y menos laboriosos que existen: no cambia la arquitectura, solo el tamaño.

El segundo hecho cambia por completo la conversación sobre compromisos. En AWS y en Azure, el precio bajo exige comprar por adelantado. Aquí hay un descuento automático por uso sostenido:

```
porcentaje del mes encendida        descuento aproximado
25 %                                  ~10 %
50 %                                  ~20 %
100 %                                 ~30 %
```

No se solicita, no se compra y no se puede olvidar. Una máquina encendida todo el mes ya está pagando cerca de un 30 % menos que su precio de lista, sin que nadie haya hecho nada. La consecuencia práctica al comparar costos entre proveedores es que el precio de lista de Google Cloud no es el precio que se paga, y compararlo con el precio de lista de otro proveedor sobreestima el gasto.

Encima de eso están los compromisos de uso, en dos formas:

```
por recurso   familia y región concretas, 1 o 3 años
              descuento mayor, flexibilidad menor
por gasto     un importe por hora comprometido, aplicable a varias familias
              descuento algo menor, mucha más flexibilidad
```

El orden correcto de decisión es el mismo que la clase 028 estableció y aquí tiene un escalón más: medir, ajustar el tamaño, dejar que el descuento automático haga su parte, y solo entonces comprometer lo que quede estable. Comprometerse antes de ajustar el tamaño es comprometerse con el desperdicio.

Y sobre el excedente, la tercera medición del mismo dato del programa:

```
AWS     ~120 s de aviso
Azure    ~30 s
Google   ~30 s
```

Treinta segundos, otra vez, dan para abortar y no para drenar. El aviso llega por el servicio de metadatos y por una señal del sistema operativo, así que se puede atender desde un script de apagado:

```
$ curl -s -H "Metadata-Flavor: Google" \
    "http://metadata.google.internal/computeMetadata/v1/instance/preempted"
TRUE
```

Y un detalle que evita una sorpresa: las máquinas de excedente no migran en vivo, terminan. La migración en vivo —que en el resto de casos hace que un mantenimiento del anfitrión no reinicie nada, y por eso aquí no existe el concepto de dominio de actualización de la clase 040— no aplica al excedente ni a algunos tipos con acelerador.

2. Discos: los dos límites, por tercera vez, y uno que sí replica

El disco persistente está conectado por red, no al anfitrión, y su rendimiento depende de dos cosas a la vez:

```
del TAMAÑO del disco       pd-ssd: ~30 IOPS por GB
del NÚMERO DE vCPU         cada tamaño de máquina tiene su tope
```

Es el mismo patrón de la clase 040 con otra aritmética, y es la tercera vez que aparece en el programa:

```
AWS     el volumen y la instancia tienen topes separados
Azure   el disco y la máquina tienen topes separados
Google  el disco escala por GB y la máquina por vCPU
```

La consecuencia es idéntica: el caudal real es el mínimo de los dos, y ampliar el disco no mueve la cifra si el techo era el otro.

```
pd-ssd de 1.024 GB            ~30.000 IOPS de lectura
n2-standard-2 (2 vCPU)        ~15.000 IOPS de lectura
→ real: 15.000
cambiar a n2-standard-8       ~25.000 IOPS  → real: 25.000
```

Los tipos disponibles y para qué sirve cada uno:

```
pd-standard    disco duro; archivos y arranques que no importan
pd-balanced    valor por defecto razonable para casi todo
pd-ssd         latencia baja y IOPS por GB
pd-extreme     IOPS aprovisionadas de forma independiente
Hyperdisk      familia moderna: tamaño, IOPS y caudal se fijan por separado
SSD local      efímero, muy rápido, SE PIERDE al detener la máquina
```

La última línea repite la lección de la clase 040: es la versión de Google Cloud del disco temporal, con la misma trampa. Sirve para caché, archivos temporales e índices reconstruibles, y no para datos.

Y aquí está la capacidad que no tiene equivalente directo en las otras dos plataformas: el disco persistente regional, replicado de forma síncrona entre dos zonas de la misma región.

```
$ gcloud compute disks create datos-01 --size 500GB --type pd-balanced \
    --region europe-west1 --replica-zones europe-west1-b,europe-west1-c
```

Lo que compra es un RTO distinto ante la caída de una zona:

```
disco zonal     la zona cae → el disco no está → restaurar desde instantánea
                RTO: el tiempo de restaurar y arrancar
disco regional  la zona cae → conectar el disco a una máquina de la otra zona
                RTO: minutos, sin restaurar nada
```

El precio es aproximadamente el doble por GB y la escritura paga la latencia de confirmar en dos zonas, así que no es la opción por defecto: es la opción para el volumen cuya pérdida de disponibilidad define el RTO del servicio —típicamente el de una base de datos que no está en un servicio gestionado.

Un apunte sobre instantáneas que ahorra trabajo: son incrementales y globales, así que una instantánea tomada en Europa se restaura en Estados Unidos sin copiarla explícitamente. Eso convierte la recuperación entre regiones en una operación sencilla, y hace que la política de instantáneas programadas sea una de las cosas más rentables de configurar el primer día.

3. Grupos gestionados: la plantilla inmutable y la reparación que hace daño

Un grupo de instancias gestionado se define con una plantilla inmutable. No se edita: para cambiar algo se crea otra y se ejecuta una actualización progresiva. Eso obliga a que cada cambio quede versionado, que es exactamente lo que se quiere:

```
$ gcloud compute instance-templates create tpl-tienda-v8 \
    --machine-type n2-standard-4 --image-family cls-tienda --image-project cls-imagenes \
    --service-account sa-tienda-web@cls-tienda-prod-euw1-01.iam.gserviceaccount.com \
    --scopes cloud-platform --no-address

$ gcloud compute instance-groups managed rolling-action start-update mig-tienda \
    --region europe-west1 --version template=tpl-tienda-v8 \
    --max-surge 2 --max-unavailable 0
```

--max-unavailable 0 es la elección por defecto correcta: se crean instancias nuevas antes de retirar las viejas, así que la capacidad nunca baja durante el despliegue. Y el canario está integrado, sin infraestructura adicional:

```
$ gcloud compute instance-groups managed rolling-action start-update mig-tienda \
    --region europe-west1 \
    --version template=tpl-tienda-v7 \
    --canary-version template=tpl-tienda-v8,target-size=10%
```

El grupo debe ser regional, no zonal: reparte automáticamente entre las zonas de la región y sobrevive a la caída de una. Un grupo zonal es un punto único de fallo con nombre de grupo.

Y ahora la parte que produce el incidente más caro de esta clase. Un grupo gestionado usa comprobaciones de estado para dos cosas distintas, y la tentación es usar la misma:

```
comprobación del BALANCEADOR      decide si la instancia recibe tráfico
                                   debe ser exigente: si no puede servir,
                                   que salga de rotación
comprobación de REPARACIÓN         decide si la instancia se DESTRUYE
                                   y se crea otra
```

Si la comprobación de reparación consulta las dependencias —base de datos, caché, otro servicio—, entonces una caída ajena hace que todas las instancias fallen a la vez, y el grupo las recrea todas a la vez. El resultado es que un fallo externo de cuarenta segundos se convierte en una caída propia de varios minutos: mientras las instancias nuevas arrancan, no hay ninguna sirviendo, y las nuevas fallan la misma comprobación en cuanto arrancan.

La regla que lo evita:

```
reparación → comprueba si el PROCESO está vivo, y nada más
             /healthz, sin dependencias, con retardo inicial generoso
balanceador → comprueba si la instancia PUEDE SERVIR
             /readyz, con dependencias, umbral corto
```

Es la distinción entre vivacidad y disponibilidad que la clase 029 introdujo, y aquí tiene una consecuencia destructiva si se ignora, porque el mecanismo no retira de rotación: borra la máquina.

```
$ gcloud compute health-checks create http hc-reparacion \
    --request-path /healthz --check-interval 30s --timeout 10s \
    --unhealthy-threshold 5 --healthy-threshold 2
$ gcloud compute instance-groups managed update mig-tienda --region europe-west1 \
    --health-check hc-reparacion --initial-delay 300
```

El retardo inicial de 300 segundos es la otra mitad: sin él, una instancia que tarda cuatro minutos en calentarse se destruye antes de llegar a servir, y el grupo entra en un ciclo de creación y destrucción que no converge nunca.

Y sobre el autoescalado, los cuatro parámetros de la clase 029 siguen valiendo. Lo que añade Google Cloud es un control de reducción que evita el otro extremo:

```
$ gcloud compute instance-groups managed set-autoscaling mig-tienda \
    --region europe-west1 --min-num-replicas 3 --max-num-replicas 30 \
    --target-cpu-utilization 0.6 --cool-down-period 90 \
    --scale-in-control max-scaled-in-replicas=3,time-window=600
```

Esa última línea limita cuántas instancias pueden retirarse en una ventana. Es la respuesta al comportamiento que la clase 029 describía: escalar hacia arriba es barato y escalar hacia abajo demasiado rápido deja al servicio sin margen justo cuando vuelve la carga.

4. Una sola dirección IP para el mundo

El balanceador externo global de aplicación es la pieza donde Google Cloud se separa más de las otras dos plataformas, y conviene entender por qué funciona como funciona.

```
una única dirección IP anycast, anunciada desde todo el borde de Google
el usuario conecta al borde más cercano
el borde decide a qué región enviar
```

Comparado con lo aprendido en la clase 040:

	Ámbito	Conmutación entre regiones
Traffic Manager (Azure)	DNS	La caché del cliente: minutos
Front Door (Azure)	Anycast	Segundos
Balanceador global (Google)	Anycast	Segundos, y por capacidad

La última columna esconde el mecanismo interesante. El reparto entre regiones no es una lista de prioridades: es capacidad declarada.

```
$ gcloud compute backend-services add-backend bs-tienda --global \
    --instance-group mig-tienda --instance-group-region europe-west1 \
    --balancing-mode RATE --max-rate-per-instance 120 --capacity-scaler 1.0
$ gcloud compute backend-services add-backend bs-tienda --global \
    --instance-group mig-tienda-use --instance-group-region us-east1 \
    --balancing-mode RATE --max-rate-per-instance 120 --capacity-scaler 1.0
```

Con eso, el balanceador envía a la región más cercana mientras quepa, y desborda a la siguiente cuando se supera la capacidad declarada o cuando las comprobaciones de estado la marcan como no disponible. No hay DNS de por medio, así que el TTL no interviene y la cola larga de clientes que seguían yendo a la región caída —los nueve minutos de la clase 040— no existe.

Y el capacity-scaler es una palanca operativa que conviene conocer: ponerlo a 0 drena una región sin borrar nada, lo que convierte un mantenimiento regional en una operación de una línea y reversible.

La familia completa, para elegir sin confundirse:

```
global externo de aplicación    HTTP(S), anycast, multirregión, con Cloud Armor
regional externo de aplicación  HTTP(S) en una región; requisitos de residencia
```

```
interno de aplicación          HTTP(S) dentro de la VPC
pasarela de red externo        capa 4, conserva la IP de origen
interno de paso                capa 4 interno, para dispositivos y bases de datos
```

Y dos piezas que completan el borde:

Cloud Armor aplica reglas de WAF y protección contra denegación de servicio en el borde, antes de llegar a la región. La disciplina de la clase 035 se traslada entera: primero medir en modo vista previa, luego bloquear. Una regla desplegada directamente en modo bloqueo contra tráfico real es la forma habitual de cortar a clientes legítimos.

```
$ gcloud compute security-policies rules create 1000 --security-policy pol-tienda \
    --expression "evaluatePreconfiguredExpr('sqli-v33-stable')" --action deny-403 \
    --preview
```

Cloud CDN se activa como una propiedad del servicio de backend, no como un producto aparte. Un solo interruptor pone el contenido estático en el borde, y su efecto sobre el costo de salida es inmediato — la partida que la clase 025 identificó como la que más sorprende.

Y la comprobación de estado, otra vez: la del balanceador es la exigente, y debe ejercitar las dependencias. Un / que devuelve 200 con la base de datos caída mantiene en rotación una región entera que no puede servir, y el desbordamiento por capacidad no se activa porque, desde fuera, esa región parece sana.

5. Elegir el tamaño con datos, no con la talla anterior

Con el descuento automático, el tipo personalizado y el recomendador, Google Cloud permite cerrar el bucle de dimensionamiento con datos en las tres decisiones. Merece un método corto porque es lo que se lleva a la clase 060.

```
$ gcloud recommender recommendations list --project cls-tienda-prod-euw1-01 \
    --location europe-west1-b --recommender google.compute.instance.MachineTypeRecommender \
    --format "table(content.overview.resourceName, content.overview.recommendedMachineType)"
```

Las recomendaciones de tamaño se basan en el uso de las últimas semanas, y tienen los mismos dos límites que las de permisos de la clase 050: no ven lo que ocurre una vez al trimestre, y miden lo usado y no lo necesario. La forma responsable de aplicarlas es la misma —empezar por lo predecible, observar, y tener a mano la vuelta atrás— con una ventaja: cambiar el tipo de máquina de un grupo gestionado es crear otra plantilla y hacer una actualización progresiva, que ya está ensayada.

El orden completo, que evita el error de comprometer desperdicio:

```
1. medir uso real de CPU y memoria durante al menos dos semanas,
   incluyendo el pico estacional si lo hay
2. ajustar el tamaño, usando tipo personalizado si el escalón no encaja
3. dejar que el descuento por uso sostenido se aplique solo
4. comprometer solo la parte que llevará un año encendida
5. mover a excedente lo que tolere 30 segundos de aviso
```

Y una comparación que conviene tener escrita para cuando alguien proponga migrar por precio:

```
máquina encendida todo el mes               100 %          ~70 %  (uso sostenido)
+ tamaño ajustado con tipo personalizado                   ~55 %
+ compromiso de un año sobre la base                       ~40 %
+ excedente para la capacidad de pico                      ~30 %
```

Cada línea es acumulativa y ninguna cambia la arquitectura. Es el mismo argumento de la clase 028 —la elección de capacidad es una decisión medible— con una plataforma que regala el primer escalón y permite el segundo con precisión.

Y el cierre que conecta con lo que viene: todo esto es cómputo gestionado por ti. Las clases 055 y 060 plantearán la pregunta que hace obsoleta buena parte de este ajuste — si el servicio puede correr en una plataforma que escala a cero, el trabajo de dimensionar deja de existir. Conviene medir el cómputo bien antes de decidir si merece la pena seguir teniéndolo.

Ejemplo trabajado

CloudShop traslada su capa de cómputo a Google Cloud. La primera medición trae una sorpresa agradable y las cuatro siguientes son problemas — uno de ellos, el mismo que ya había aparecido dos veces en el programa.

Sorpresa inicial — el descuento ya estaba aplicado.

La estimación previa se hizo con precios de lista y salió más cara que Azure. La primera factura real:

```
6 × n2-standard-4, mes completo        1.128 USD    790 USD
                                                    (−30 % por uso sostenido)
```

Nadie compró nada. El equipo corrige el método de estimación: comparar precios de lista entre proveedores con modelos de descuento distintos produce conclusiones falsas, y hay que comparar el precio efectivo de una carga concreta.

Incidente 1 — el disco de 1 TB que rinde la mitad.

La base de datos autogestionada de informes no pasa de 15.000 IOPS con un disco que debería dar 30.000.

```
$ gcloud compute disks describe datos-informes --zone europe-west1-b \
    --format="value(sizeGb,type)"
1024   pd-ssd
$ gcloud compute instances describe informes-01 --zone europe-west1-b \
    --format="value(machineType.basename())"
n2-standard-2
```

Es la tercera vez que el programa encuentra este problema: en AWS con el volumen y la instancia, en Azure con el disco y el tamaño de máquina, y aquí con los GB del disco y las vCPU de la máquina.

```
máquina                   n2-standard-2 (15.000)  n2-standard-8 (25.000)
disco                     pd-ssd 1.024 GB         pd-ssd 1.024 GB
IOPS reales                     15.000                  25.000
duración del informe nocturno   94 min                  41 min
```

Incidente 2 — 16 vCPU para usar 6.

El servicio de catálogo estaba en n2-standard-16 porque necesitaba memoria, no CPU:

```
uso medido    CPU 34 % de 16 vCPU     memoria 51 GB de 64
```

```
tipo                            n2-standard-16    n2-custom-8-53248
vCPU                                  16                 8
memoria                              64 GB              52 GB
costo mensual por instancia         ~395 USD          ~232 USD
instancias                             4                  4
costo mensual del servicio         1.580 USD          928 USD
```

Seiscientos cincuenta dólares al mes por no redondear al escalón siguiente. La arquitectura no cambió.

Incidente 3 — una caída de 40 segundos que duró nueve minutos.

La base de datos gestionada conmutó entre zonas —40 segundos—. El servicio estuvo caído nueve minutos.

```
$ gcloud compute operations list --filter="operationType:compute.instances.insert" \
    --format="value(insertTime,targetLink.basename())" | head
10:14:22  tienda-a7f2
10:14:22  tienda-c91e
10:14:23  tienda-b03d
10:14:23  tienda-4ae8
```

Las cuatro instancias se recrearon a la vez. La comprobación de reparación automática consultaba /readyz, que verifica la base de datos: al conmutar esta, las cuatro instancias fallaron a la vez y el grupo las destruyó a la vez. Las nuevas tardaron cuatro minutos en arrancar y volvieron a fallar la misma comprobación.

```
comprobación de reparación         /readyz (dependencias)  /healthz (proceso vivo)
umbral de fallo                    2 fallos                5 fallos
intervalo                          10 s                    30 s
retardo inicial                    0 s                     300 s
comprobación del balanceador       /readyz                 /readyz  ← se queda
duración del episodio equivalente   9 min                   41 s
```

La comprobación del balanceador no cambió: estaba bien. Lo que estaba mal era usar la misma para decidir si destruir una máquina.

Incidente 4 — el simulacro de región, comparado con la clase 040.

Se drena europe-west1 poniendo su escalador de capacidad a cero:

```
$ gcloud compute backend-services update-backend bs-tienda --global \
    --instance-group mig-tienda --instance-group-region europe-west1 \
    --capacity-scaler 0.0
```

```
mecanismo                        Traffic Manager / Front Door  anycast + capacidad
conmutación observada             9 min / 35 s                    2,4 s
```

```
errores vistos por el cliente     algunos / ninguno tras reintento  ninguno
direcciones IP implicadas              2                              1
reversión                         cambiar destinos             capacity-scaler 1.0
```

Dos coma cuatro segundos y una sola dirección IP. Y la operación inversa es la misma línea con otro valor, lo que la convierte en un procedimiento de mantenimiento y no en un plan de contingencia.

Incidente 5 — una zona cae y el disco estaba en ella.

El simulacro de zona deja fuera la máquina que aloja el servicio de búsqueda, con su índice en un disco zonal:

```
recuperación                  restaurar instantánea  conectar en la otra zona
RTO medido                        47 min                 4 min
pérdida de datos                  hasta 6 h              ninguna
costo mensual del volumen         42 USD                 84 USD
```

Cuarenta y dos dólares al mes por bajar el RTO de 47 a 4 minutos, solo para el volumen cuya disponibilidad define el RTO del servicio. El resto de discos se quedan zonales con instantáneas programadas.

Resumen de la capa de cómputo:

```
tipo de las máquinas de catálogo         n2-standard-16   n2-custom-8-53248
IOPS reales del volumen de informes           15.000          25.000
duración del informe nocturno                 94 min          41 min
episodio por conmutación de base de datos      9 min           41 s
conmutación entre regiones                       —             2,4 s
RTO ante caída de zona del índice             47 min           4 min
costo mensual de cómputo                    2.370 USD      1.718 USD
```

La lección que esta clase traslada al resto de la parte 04: dos de los cinco problemas eran conocidos —los dos límites de disco y la comprobación de estado que confunde vivacidad con disponibilidad— y aparecieron igual, porque son propiedades del problema y no del proveedor. El único mecanismo genuinamente nuevo, la reparación automática, hizo más daño que ninguno: es la primera pieza del programa que responde a un fallo destruyendo infraestructura sana, y por eso su comprobación tiene que ser la más conservadora de todo el sistema.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-04-gcp-core-platform/052-compute-engine-managed-instance-groups-y-load-balancing/
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-gcp 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-gcp 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 amplía el disco y las IOPS no cambian	El tope efectivo lo pone el número de vCPU de la máquina, no el tamaño del disco	Consulta ambos límites: el caudal real es el mínimo, y se corrige subiendo de tipo de máquina.
Un fallo externo de segundos se convierte en una caída propia de minutos	La comprobación de reparación automática consulta dependencias, así que todas las instancias fallan y se destruyen a la vez	Reparación con /healthz sin dependencias, umbral alto y retardo inicial; balanceador con /readyz.
Un grupo gestionado crea y destruye instancias sin converger	El retardo inicial es menor que el tiempo de arranque del servicio	Fija --initial-delay por encima del arranque real medido, con margen.
La comparación de costos con otro proveedor sale muy desfavorable	Se compararon precios de lista y el descuento por uso sostenido se aplica solo	Compara el precio efectivo de una carga concreta, con descuentos y tamaños ajustados en ambos lados.
Se paga una máquina grande para llegar a la memoria necesaria	Se eligió un tipo predefinido y el escalón siguiente duplica también la CPU	Usa un tipo personalizado con las vCPU y la memoria medidas, dentro de las proporciones permitidas.
Tras la caída de una zona hay que restaurar para recuperar un volumen	El disco era zonal y su disponibilidad definía el RTO del servicio	Usa disco persistente regional solo para ese volumen y deja el resto zonal con instantáneas programadas.

■ 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 descuento se aplica sin comprar nada y cómo cambia la forma correcta de comparar costos entre proveedores?
2. ¿Qué dos factores determinan las IOPS de un disco persistente, y en qué se parece a lo visto en las clases 028 y 040?
3. ¿Por qué la comprobación de reparación automática debe ser distinta de la del balanceador, y qué ocurre si son la misma?
4. ¿Cómo conmuta entre regiones un balanceador global sin depender del DNS, y qué papel tiene la capacidad declarada?
5. ¿En qué caso concreto compensa un disco persistente regional y cómo se justifica su costo?

Referencias

- Google Cloud (2025). Machine families resource and comparison guide — familias, tipos personalizados y proporciones. <https://cloud.google.com/compute/docs/machine-resource>
- Google Cloud (2025). Sustained use discounts — descuento automático y su interacción con los compromisos. <https://cloud.google.com/compute/docs/sustained-use-discounts>

• Google Cloud (2025). Block storage performance — IOPS por GB y límites por número de vCPU. <https://cloud.google.com/compute/docs/disks/performance>
• Google Cloud (2025). Autohealing instances in MIGs — comprobación de reparación, retardo inicial y diferencias con la del balanceador. <https://cloud.google.com/compute/docs/instance-groups/autohealing-instances-in-migs>
• Google Cloud (2025). External Application Load Balancer overview — anycast, modos de equilibrio y escalador de capacidad. <https://cloud.google.com/load-balancing/docs/https>
• Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 04 en PDF · Recorrido de Google Cloud en PDF · Manual integral

Anterior	Índice	Siguiente
← 051 · VPC global, subredes regionales, firewall y Cloud NAT	Parte 04 · Programa	053 · Cloud Storage, clases, lifecycle y replicación →

Evaluación — 052 Compute Engine, managed instance groups y load balancing

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-gcp 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.

053 — Cloud Storage, clases, lifecycle y replicación

Parte: 04 — Google Cloud: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: storage · Estado: EXECUTABLECORE

Propósito

Operar el almacenamiento de objetos de Google Cloud, donde dos de las trampas que costaron caro en las clases 030 y 041 sencillamente no existen —la región del par replicado se elige, y el nivel más frío se lee al instante— y aparece una dimensión que ninguna de las dos plataformas anteriores obligaba a considerar: el número de objetos cuesta dinero además del volumen, y una regla de ciclo de vida mal escrita puede gastar más en operaciones de lo que ahorra en almacenamiento.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir ubicación —región, birregión o multirregión— a partir de residencia de datos y RPO exigido.
2. Calcular el ahorro de un cambio de clase incluyendo el costo por operación y el número de objetos.
3. Configurar acceso uniforme y prevención de acceso público, con su prueba negativa.
4. Emitir URL firmadas sin ninguna clave, mediante suplantación y firma delegada.
5. Verificar la recuperación de objetos y de buckets borrados en vez de suponerla.

Conceptos centrales

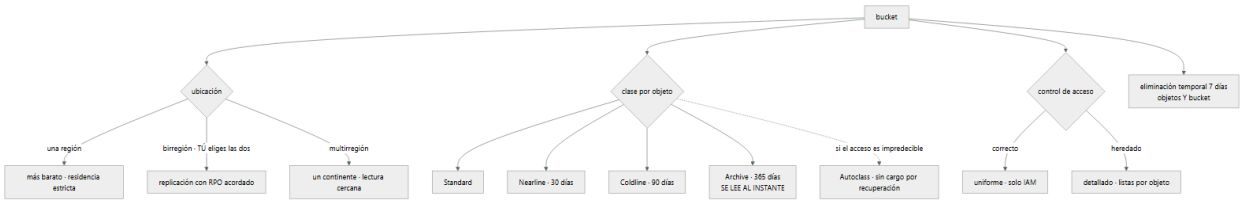
Concepto	Comprensión verificable
ubicación del bucket	Región, birregión —dos regiones que eliges tú— o multirregión. Es lo que en la clase 041 no se podía elegir, y aquí sí.
clase de almacenamiento	Propiedad de cada objeto. Las cuatro clases tienen acceso en milisegundos: no hay rehidratación. Lo que sube al enfriar no es el tiempo, es el precio de leer y de operar.
operación de clase A y de clase B	Escrituras y listados frente a lecturas. Su precio sube al enfriar la clase, hasta diez veces, así que mover millones de objetos pequeños puede costar más que guardarlos.
acceso uniforme a nivel de bucket	Modo en el que solo IAM decide el acceso. El modo detallado mantiene además listas de control por objeto, invisibles en la política del bucket.
Autoclass	Cambio de clase automático por objeto según su acceso real, sin cargos por recuperación, a cambio de una tarifa de gestión. Es la respuesta cuando el patrón de acceso no se puede predecir.
eliminación temporal	Retención por defecto de los objetos borrados y del bucket, durante siete días. Cubre de un golpe el hueco que en la clase 041 exigía dos interruptores distintos.

Modelo mental

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

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 ubicación se elige, y eso resuelve un problema de la clase 041

Un bucket tiene un nombre global —como el identificador de proyecto de la clase 049, único entre todos los clientes— y una ubicación que se fija al crearlo y no cambia:

	Qué replica	Cuándo
Región	Dentro de una región, entre zonas	Residencia estricta y menor costo
Birregión	Entre dos regiones que eliges	Recuperación regional con control del destino
Multirregión	Dentro de un continente	Lectura desde muchos sitios, sin elegir dónde

La fila del medio merece detenerse, porque responde a un problema concreto que la clase 041 dejó sin solución elegante. Allí, la redundancia geográfica de Azure replicaba a una región emparejada asignada por Microsoft, y cuando ese destino caía fuera de la jurisdicción aprobada había que renunciar a la redundancia geográfica y montar una replicación explícita.

Aquí la birregión se declara:

```
$ gcloud storage buckets create gs://cls-facturas \
    --placement europe-west1,europe-west4 \
    --default-storage-class STANDARD --uniform-bucket-level-access \
    --enable-autoclass
```

Dos regiones elegidas, ambas dentro del espacio aprobado, con una sola URL y una sola API. Y si el RPO importa, la replicación turbo lo convierte en un compromiso medible en vez de en una aspiración:

```
replicación birregional por defecto   asíncrona, sin RPO comprometido
replicación turbo                     objetivo de 15 minutos, con acuerdo
```

Esa segunda línea es el contraste directo con lastSyncTime de la clase 041: allí el RPO era un número que había que medir y que nadie garantizaba; aquí hay un compromiso, a cambio de un precio.

Un detalle de ingeniería que conviene entender antes de elegir multirregión por comodidad: la replicación no es síncrona, así que una escritura confirmada en una región puede no estar en la otra durante un intervalo. La consistencia de una lectura tras escritura sí está garantizada para el objeto —Google Cloud Storage es fuertemente consistente en lectura tras escritura y en listados— y lo que no está garantizado es que la copia de la otra región esté al día en ese instante. Diseñar suponiendo lo contrario produce el mismo tipo de sorpresa que la conmutación de la clase 041.

Y hay una asimetría de costos que decide más de lo que parece: multirregión cuesta más por GB almacenado y, según la operación, más por operación. Para datos que se leen desde una sola región, pagar la multirregión es pagar disponibilidad que nadie usa. El criterio es el de siempre: la ubicación se elige por el requisito de residencia y de recuperación, no por el que suena mejor.

2. Enfriar cuesta operaciones, no tiempo

Aquí está la diferencia que más cambia el diseño respecto a las dos plataformas anteriores. Las cuatro clases se leen en milisegundos:

Clase	Almacenamiento aprox.	Permanencia mínima	Recuperación por GB	¿Hay que rehidratar?
Standard	0,020 USD/GB/mes	—	—	—
Nearline	0,010	30 días	0,01	No

Clase	Almacenamiento aprox.	Permanencia mínima	Recuperación por GB	¿Hay que rehidratar?
Coldline	0,004	90 días	0,02	No
Archive	0,0012	365 días	0,05	No

Comparado con lo aprendido:

```
Azure archivo   hasta 15 h de rehidratación → incompatible con casi cualquier RTO
AWS Glacier     minutos a horas según el nivel
Google Archive  milisegundos, con cargo por lectura
```

Eso elimina la frontera que la clase 041 estableció —«archivo es para lo que hay que guardar, no para lo que hay que poder leer»— y la sustituye por otra: archivo es para lo que casi nunca se lee, porque leerlo cuesta.

Y aparece la dimensión nueva, la que ninguna de las dos plataformas anteriores obligaba a mirar: el precio por operación sube al enfriar la clase.

```
Standard                    ~0,005 USD por 1.000
Nearline                    ~0,010
Coldline                    ~0,020
Archive                     ~0,050            ← diez veces la de Standard
```

Una regla de ciclo de vida que mueve millones de objetos pequeños a Archive ejecuta una operación de clase A por objeto:

```
4,2 millones de objetos a Archive
  4.200.000 / 1.000 × 0,05 = 210 USD, de una vez

y si esos objetos suman 300 GB:
  ahorro mensual = 300 × (0,020 − 0,0012) = 5,64 USD/mes
  → la operación tarda 37 MESES en amortizarse
```

Y eso sin contar que cada lectura posterior cuesta 0,05 USD por GB. La operación destruye valor, y en las plataformas anteriores el mismo razonamiento no aparecía porque el precio por operación no cambiaba con la clase.

La corrección es una condición de tamaño en la regla, para que el ciclo de vida ignore lo pequeño:

```
{
  "lifecycle": {"rule": [
    {"action": {"type": "SetStorageClass", "storageClass": "NEARLINE"},
     "condition": {"age": 30, "matchesPrefix": ["facturas/"],
                   "matchesStorageClass": ["STANDARD"]}},
    {"action": {"type": "SetStorageClass", "storageClass": "ARCHIVE"},
     "condition": {"age": 365, "matchesStorageClass": ["NEARLINE"]}},
    {"action": {"type": "Delete"},
     "condition": {"daysSinceNoncurrentTime": 90, "isLive": false}}
  ]}
}
```

La última regla es la que ya ha aparecido en las clases 030 y 041 y aquí es la cuarta: con versionado activo, borrar no libera nada. Las versiones no actuales siguen facturando hasta que una regla las retire, y la factura no baja tras una limpieza si nadie escribió esa condición.

Y cuando el patrón de acceso no se puede predecir —que es el caso honesto en la mayoría de conjuntos de datos—, está Autoclass:

```
mueve cada objeto de clase según SU acceso real
no cobra recuperación al leer un objeto frío
no cobra la operación de transición
cobra una tarifa de gestión por objeto y mes
```

Eso elimina de un golpe todo el cálculo de punto de equilibrio de las clases 030 y 041. La decisión pasa a ser una comparación simple: la tarifa de gestión frente al ahorro esperado. Para un bucket con acceso irregular, casi siempre gana Autoclass; para uno con un patrón claro y estable, una regla explícita sale más barata.

3. Acceso: uniforme, sin claves y sin URL eternas

Google Cloud Storage tiene dos modelos de control de acceso y uno de ellos es un problema heredado:

```
acceso uniforme a nivel de bucket   solo IAM decide. Auditable.
acceso detallado                    IAM + listas de control POR OBJETO
```

El segundo es la razón por la que un bucket puede estar exponiendo un objeto concreto mientras su política de IAM parece impecable: la lista de control vive en el objeto, no en el bucket, y revisar la política no la muestra. Encontrar todos los objetos con una lista permisiva exige recorrerlos uno a uno.

```
$ gcloud storage buckets update gs://cls-facturas --uniform-bucket-level-access
```

Una vez activado, las listas por objeto dejan de tener efecto. Y por encima, la política de organización de la clase 049 impide que nadie vuelva a abrirlo:

```
$ gcloud resource-manager org-policies enable-enforce \
    constraints/storage.publicAccessPrevention --organization $ORG_ID
$ gcloud storage buckets add-iam-policy-binding gs://cls-facturas \
    --member allUsers --role roles/storage.objectViewer
ERROR: 412 Request violates constraint storage.publicAccessPrevention     ✓
```

Es la misma prueba negativa de la clase 046, ejecutada por segunda vez con otro vocabulario y el mismo valor: el mensaje no habla de permisos, así que añadir roles no habría cambiado nada.

Y sobre el acceso temporal a un objeto concreto, la URL firmada tiene el mismo problema estructural que la SAS de la clase 041:

```
una URL firmada NO se puede revocar
solo caduca
```

La diferencia está en con qué se firma. La vía cómoda es una clave HMAC o una clave de cuenta de servicio, y las dos son exactamente lo que la clase 050 acaba de eliminar. La vía correcta no crea ninguna clave: la firma la produce la propia API de IAM, previa suplantación.

```
$ gcloud storage sign-url gs://cls-facturas/2026-07.pdf \
    --duration 15m \
    --impersonate-service-account firmador@cls-tienda-prod-euw1-01.iam.gserviceaccount.com
```

Eso exige que quien firma tenga roles/iam.serviceAccountTokenCreator sobre la cuenta firmadora, lo que a su vez es auditable y caduca. Y la duración corta es la única protección real: si no se puede revocar, que dure quince minutos y no siete días.

Tres piezas más completan el control, y cada una responde a un requisito distinto:

```
política de retención + bloqueo   WORM: ni el propietario puede borrar antes
                                  el bloqueo es IRREVERSIBLE, como en la clase 041
retención por objeto              plazo distinto para objetos distintos
clave gestionada por el cliente   control y revocación, con la responsabilidad
                                  descrita en la clase 046
```

Y el que quien pague sea el lector —el bucket con pago por solicitante— resuelve un caso concreto que aparece al compartir datos: publicar un conjunto grande sin pagar la salida de quien lo descargue. Es una decisión de costo con nombre, y conviene conocerla antes de que alguien proponga copiar el conjunto a otro sitio para evitar el gasto.

4. Borrar y recuperar: el hueco de la clase 041 aquí está cubierto

La clase 041 dejó el incidente más caro de la parte 03: la eliminación temporal de blob estaba activa, la de contenedor no, y un contenedor borrado se llevó 412 GiB sin recuperación posible. Eran dos interruptores independientes y solo uno estaba puesto.

Aquí ese hueco no existe de la misma forma. La eliminación temporal de Cloud Storage está activada por defecto con siete días de retención y cubre las dos cosas a la vez:

```
$ gcloud storage buckets describe gs://cls-facturas \
    --format="value(soft_delete_policy.retentionDurationSeconds)"
604800
```

```
objetos borrados      recuperables durante la retención
BUCKET borrado        recuperable durante la retención, con su contenido
```

La restauración de un bucket entero:

```
$ gcloud storage buckets list --soft-deleted --format="value(name,generation)"
cls-pruebas   1753992014882
$ gcloud storage buckets restore gs://cls-pruebas --generation 1753992014882
```

Eso no elimina la obligación de probarlo. Es exactamente el mismo argumento de la clase 041: la única forma honesta de saber si una recuperación funciona es borrar algo a propósito y recuperarlo, y hacerlo periódicamente, porque la retención se puede reducir y alguien puede hacerlo.

Los mecanismos completos y qué cubre cada uno, que siguen siendo varios y no uno:

```
eliminación temporal   borrado accidental de objetos y del bucket, 7 días
versionado             sobrescrituras: conserva la versión anterior
retención + bloqueo    borrado deliberado, incluso por el propietario
instantáneas / copias  corrupción lógica propagada por la aplicación
```

La cuarta fila es la que ninguna de las tres primeras cubre y la que más se olvida: si la aplicación escribe datos corruptos, el versionado guarda la versión buena y la eliminación temporal no interviene, pero nadie avisa, y cuando se descubre puede haber pasado la ventana. La defensa es la de la clase 030: una copia con ciclo de vida propio, en otro proyecto, con permisos distintos.

Y una decisión de gobierno que conviene tomar pronto: el proyecto que contiene las copias no debe ser administrable por quien administra el original. Es la diferencia entre una copia de seguridad y una segunda carpeta.

La comparación final con las dos plataformas anteriores, que es lo que se lleva a la clase 060:

```
protección por defecto           ninguna       ninguna     eliminación temporal
interruptores necesarios            3             5              2
región del par replicado         se elige     asignada        se elige
lectura del nivel más frío       minutos-horas  hasta 15 h    milisegundos
costo por operación al enfriar    estable      estable       hasta ×10
```

La última fila es la única en la que Google Cloud añade una trampa que las otras no tenían. Las cuatro primeras van en la otra dirección, y confirman la parte de la hipótesis de la clase 048 que decía que las excepciones serían otras: no se repite ninguna de las de Azure, y aparece una nueva que hay que aprender.

5. Lo que cuesta de verdad: salida, operaciones y objetos pequeños

La factura de almacenamiento tiene cuatro partidas y la primera casi nunca es la mayor:

```
almacenamiento    GB por mes, según clase y ubicación
operaciones       clase A y clase B, según clase
recuperación      GB leídos desde clases frías
salida de red     GB que salen de Google Cloud
```

La salida de red es la que sorprende, igual que en las dos plataformas anteriores, y tiene aquí una respuesta directa: casi todo el tráfico interno no sale. Un servicio en Compute Engine o Cloud Run leyendo de un bucket de la misma región no genera salida, y el acceso privado a Google de la clase 051 asegura además que no pase por Cloud NAT. Las dos decisiones juntas eliminan la partida entera para el tráfico interno.

Para el tráfico hacia usuarios, la palanca es Cloud CDN delante del bucket, que la clase 052 dejó a un interruptor de distancia. El cálculo es el de siempre: cuánto del contenido es cacheable y qué proporción de peticiones deja de llegar al origen.

Y la partida de operaciones merece una regla propia, porque es donde esta plataforma se comporta distinto:

```
un objeto de 2 KB en Archive
  almacenamiento    0,0000024 USD/mes
  una lectura       0,0000004 USD de operación + 0,0000001 de recuperación
  → el objeto cuesta más leerlo que guardarlo un año
```

Eso lleva a un principio de diseño que no aparecía en las clases 030 ni 041: agrupar objetos pequeños antes de enfriarlos. Un millón de registros de 2 KB en un solo archivo comprimido cuesta una operación en lugar de un millón, y el ahorro es de dos órdenes de magnitud. La composición de objetos y los formatos columnares existen para eso, y la decisión se toma al escribir, no al archivar.

El método completo para decidir una regla de ciclo de vida, que es el entregable de esta clase:

```
1. ¿cuántos objetos y de qué tamaño medio?
   si el tamaño medio es de pocos KB, agrupar antes de nada
2. ¿qué proporción se relee al mes?
   por encima del punto de equilibrio, enfriar cuesta dinero
3. ¿se puede predecir el acceso?
   si no, Autoclass y se acabó el cálculo
4. ¿cuánto cuesta la transición?
   objetos × precio de operación de la clase destino, de una vez
5. ¿en cuántos meses se amortiza?
   si la respuesta es más de doce, no se hace
```

El paso 5 es el que falta en casi todas las políticas de ciclo de vida que existen, y es el único que convierte la decisión en algo defendible.

Ejemplo trabajado

CloudShop lleva su almacenamiento a Google Cloud con la línea base de la clase 041 en la mano. Dos de los cinco problemas de aquella clase no se reproducen, uno se reproduce igual, y aparece uno nuevo que cuesta 210 dólares en una tarde.

Lo que no se reprodujo.

```
la región del par replicado          en Azure se asignaba; aquí se eligió
                                     europe-west1 + europe-west4, ambas
                                     dentro de la jurisdicción aprobada
borrado de un contenedor             en Azure se perdieron 412 GiB por tener
                                     un interruptor de dos; aquí la eliminación
                                     temporal viene activa y cubre las dos cosas
```

Ambas se verificaron en vez de darse por buenas:

```
$ gcloud storage buckets delete gs://cls-prueba-recuperacion
$ gcloud storage buckets list --soft-deleted --format="value(name,generation)"
cls-prueba-recuperacion   1753992014882
$ gcloud storage buckets restore gs://cls-prueba-recuperacion --generation 1753992014882
$ gcloud storage ls gs://cls-prueba-recuperacion | wc -l
50                                                                          ✓
```

Incidente 1 — la regla de ciclo de vida que destruyó valor.

Se aplicó la política de la clase 041 traducida: a los 365 días, todo a Archive. El bucket de eventos contenía 4,2 millones de objetos de 71 KB de media.

```
$ gcloud storage ls -l gs://cls-eventos/** | tail -1
TOTAL: 4203118 objects, 298471029248 bytes (278 GiB)
```

```
costo de la transición
  4.203.118 / 1.000 × 0,05 USD = 210,16 USD, cobrados de una vez
ahorro mensual
  278 GiB × (0,020 − 0,0012) = 5,22 USD/mes
amortización                                  40 meses
```

Y el golpe real llegó al mes siguiente: un análisis releyó el 12 % del conjunto.

```
recuperación   33 GiB × 0,05 = 1,65 USD
operaciones    504.000 lecturas / 1.000 × 0,05 = 25,20 USD
total del análisis, que en Standard habría costado ~2,50 USD:  26,85 USD
```

La corrección tiene dos partes, y la segunda es la que dura:

```
regla de ciclo de vida         todo a Archive a 365 d   solo objetos > 1 MB
objetos pequeños                    4,1 M              agrupados al escribir:
                                                       un archivo diario
objetos tras agrupar                4,2 M               31.400
clase del conjunto agrupado         Archive             Coldline
costo de la siguiente transición   210 USD              1,57 USD
costo del mismo análisis            26,85 USD           2,90 USD
```

Incidente 2 — un objeto público con la política del bucket impecable.

Una revisión externa encuentra una factura accesible sin autenticar. La política del bucket no concede nada a allUsers.

```
$ gcloud storage buckets describe gs://cls-facturas \
    --format="value(uniform_bucket_level_access.enabled)"
False
$ gcloud storage objects describe gs://cls-facturas/2025-11.pdf --format="value(acl)"
[{'entity': 'allUsers', 'role': 'READER'}]
```

Una lista de control en el objeto, puesta dos años atrás por una herramienta de migración. La política del bucket nunca la mostró.

```
modelo de acceso                        detallado       uniforme
objetos con lista de control permisiva        3              0
prevención de acceso público            no aplicada    política de organización
prueba negativa                             no          sí, ejecutada
```

Incidente 3 — la factura que no baja. Cuarta vez.

```
datos visibles     412 GiB
facturado        1.108 GiB
```

Versionado activo y ninguna regla sobre versiones no actuales. Es el mismo problema del marcador de borrado de la clase 030 y de las versiones de la clase 041, con un tercer nombre.

```
facturado                   1.108 GiB     430 GiB
costo mensual del bucket     22,16 USD     8,60 USD
```

La nota que el equipo añade a su lista de comprobación de plataforma nueva: si hay versionado, tiene que haber una regla que retire versiones, y hay que verificar que el facturado coincide con lo visible.

Incidente 4 — una URL firmada de siete días y la clave con la que se firmaba.

```
$ gcloud storage hmac list --project cls-tienda-prod-euw1-01 --format="value(accessId,state)"
GOOG1E…   ACTIVE
```

La aplicación móvil pedía al backend una URL firmada de siete días, y el backend firmaba con una clave HMAC guardada en su configuración. Dos problemas encadenados: la clave es una credencial de larga duración —justo lo que la clase 050 eliminó— y la URL no se puede revocar.

```
firma                          clave HMAC en config   API de IAM, sin clave
duración de la URL                  7 días              15 minutos
revocación posible                    no          irrelevante: caduca sola
claves HMAC activas                    1                   0
```

Resumen del almacenamiento:

```
objetos en el bucket de eventos              4,2 M          31.400
costo de una transición de clase            210 USD        1,57 USD
costo de un análisis del 12 %               26,85 USD      2,90 USD
facturado frente a visible                 1.108/412 GiB   430/412 GiB
objetos públicos                               3               0
claves de firma de larga duración              1               0
duración de las URL firmadas                 7 días         15 minutos
recuperación de bucket probada                 no         sí, mensual
costo mensual de almacenamiento             64,80 USD      28,40 USD
```

La lección que esta clase traslada al resto de la parte 04: dos trampas de la clase 041 no se repitieron y una tercera —la factura que no baja con versionado— apareció por cuarta vez, lo que la confirma como propiedad del problema y no del proveedor. Y la trampa nueva tiene una forma que conviene reconocer para el futuro: una plataforma que abarata el almacenamiento y encarece la operación cambia el punto de equilibrio de sitio, y una política copiada de otro proveedor puede costar más de lo que ahorra sin que nada falle.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-04-gcp-core-platform/053-cloud-storage-clases-lifecycle-y-replicacion/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 bucket-gobernado-gcp 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 bucket-gobernado-gcp 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 regla de ciclo de vida dispara la factura en vez de reducirla	Mueve millones de objetos pequeños a una clase cuyo precio por operación es hasta diez veces mayor	Añade condición de tamaño mínimo, agrupa los objetos pequeños al escribirlos y calcula en cuántos meses se amortiza la transición.
Un objeto es público y la política del bucket no concede nada a nadie	El bucket está en modo de acceso detallado y el permiso está en una lista de control del objeto	Activa el acceso uniforme a nivel de bucket y la prevención de acceso público, y verifica con la prueba negativa.
Se libera espacio y el facturado no baja	Con versionado activo, borrar conserva la versión no actual	Añade una regla sobre daysSinceNoncurrentTime y comprueba que el volumen facturado converge con el visible.
Una URL firmada filtrada sigue siendo válida durante días	No se puede revocar y se emitió con una duración larga, firmada con una clave de larga duración	Firma mediante suplantación con la API de IAM, sin claves, y emite duraciones de minutos.
Leer un conjunto archivado cuesta más que haberlo dejado en Standard	El costo de recuperación y el de operación por objeto superan el ahorro de almacenamiento	Calcula la proporción releída al mes; si no se puede predecir, usa Autoclass, que no cobra recuperación.
Se elige multirregión por comodidad y el costo por GB sube	La ubicación se decidió por preferencia y no por requisito de residencia o recuperación	Elige región para lectura local, birregión con destino propio para recuperación y multirregión solo cuando la lectura es realmente global.

■ 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 problema concreto de la clase 041 resuelve poder elegir las dos regiones de una birregión?
2. ¿Por qué aquí no hay rehidratación y en qué cambia eso el criterio para usar la clase más fría?
3. Calcula si compensa mover 3 millones de objetos de 40 KB a Archive, y explica qué dato falta si no puedes.
4. Un objeto es público con la política del bucket vacía. ¿Dónde está el permiso y cómo se elimina esa clase entera de problemas?
5. ¿Cómo se emite una URL firmada sin que exista ninguna clave, y por qué la duración corta es la única protección real?

Referencias

- Google Cloud (2025). Bucket locations — región, birregión con destino elegido, multirregión y replicación turbo. <https://cloud.google.com/storage/docs/locations>
- Google Cloud (2025). Storage classes — permanencia mínima, recuperación y precios por operación. <https://cloud.google.com/storage/docs/storage-classes>
- Google Cloud (2025). Object Lifecycle Management — condiciones, versiones no actuales y transiciones. <https://cloud.google.com/storage/docs/lifecycle>
- Google Cloud (2025). Uniform bucket-level access — IAM frente a listas de control por objeto. <https://cloud.google.com/storage/docs/uniform-bucket-level-access>
- Google Cloud (2025). Soft delete — retención de objetos y de buckets, y restauración. <https://cloud.google.com/storage/docs/soft-delete>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 04 en PDF · Recorrido de Google Cloud en PDF · Manual integral

Anterior	Índice	Siguiente
← 052 · Compute Engine, managed instance groups y load balancing	Parte 04 · Programa	054 · Cloud SQL, Spanner, Firestore y Memorystore →

Evaluación — 053 Cloud Storage, clases, lifecycle y replicación

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 bucket-gobernado-gcp 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.

054 — Cloud SQL, Spanner, Firestore y Memorystore

Parte: 04 — Google Cloud: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: data · Estado: EXECUTABLECORE

Propósito

Elegir motor de datos en Google Cloud con el criterio de la clase 031 —los patrones de acceso primero— y con tres precisiones que solo aparecen aquí: llegar a Cloud SQL por red privada exige reservar un rango de direcciones que nadie planificó, Spanner cuesta lo que cuesta y solo se justifica por lo que únicamente él da, y en Firestore el precio de una consulta es el número de documentos que devuelve, así que un panel mal escrito factura más que la base de datos entera.

Resultados de aprendizaje

Al finalizar podrás:

1. Conectar a Cloud SQL por red privada sabiendo qué rango hay que reservar y por qué el proxy autenticado es la vía correcta.
2. Decidir entre Cloud SQL, Spanner y Firestore con una justificación de costo y de patrón de acceso, no de catálogo.
3. Diseñar una clave primaria que no concentre las escrituras, en el motor que sea.
4. Calcular el costo de una consulta de Firestore a partir del tamaño del resultado.
5. Anticipar las decisiones irreversibles de esta clase, empezando por el almacenamiento que crece y no baja.

Conceptos centrales

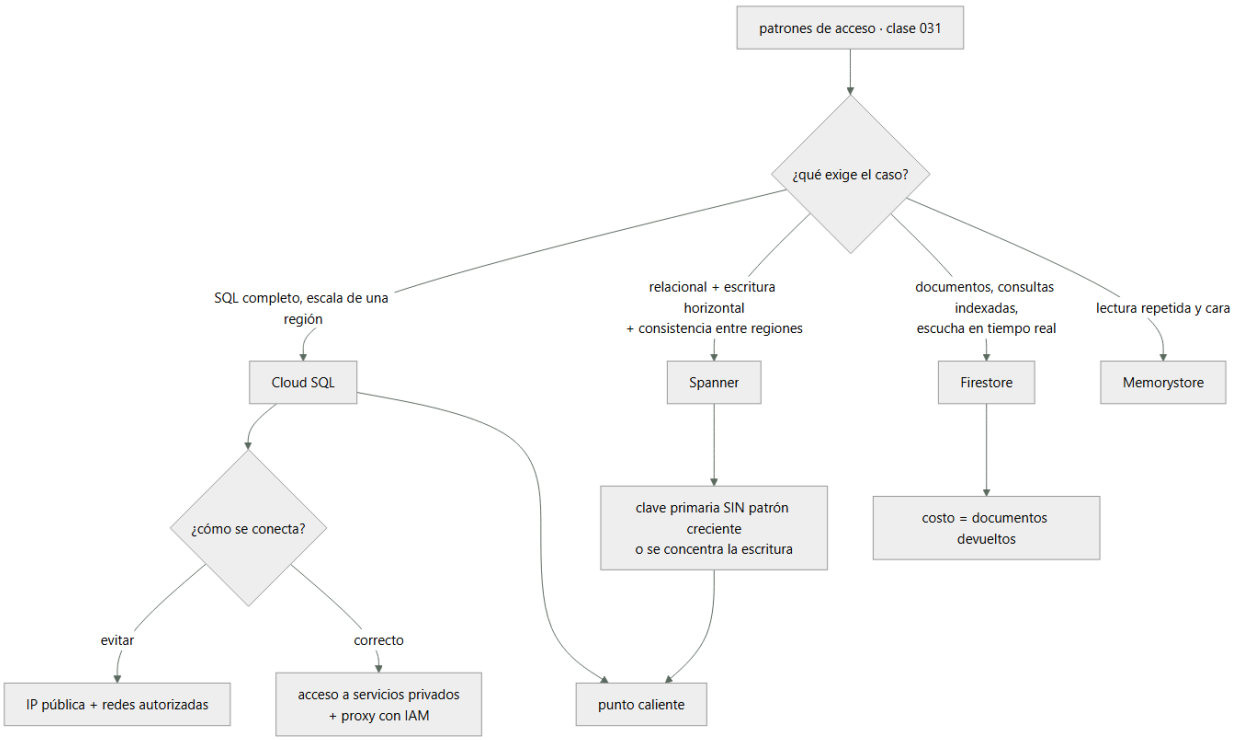
Concepto	Comprensión verificable
acceso a servicios privados	Emparejamiento con una red gestionada por Google que permite alcanzar Cloud SQL por dirección interna. Consume un rango reservado de tu espacio, que hay que planificar en la clase 051.
proxy de autenticación de Cloud SQL	Conector que autentica con IAM y cifra el tránsito sin gestionar certificados ni listas de direcciones autorizadas. Es la vía recomendada y la que evita exponer una IP pública.
punto caliente	Concentración de escrituras en una porción del espacio de claves. Una clave primaria creciente —una marca de tiempo, un contador— lo garantiza, y limita el rendimiento por mucho que se añada capacidad.
consistencia externa	Garantía de Spanner: las transacciones se ven en un orden global coherente incluso entre regiones. Es lo único que no se puede construir encima de otro motor, y es lo que se está pagando.
lectura de documento	Unidad de facturación de Firestore. Una consulta que devuelve mil documentos cuesta mil lecturas: el precio es el tamaño del resultado, no la complejidad.
crecimiento automático de almacenamiento	Cloud SQL amplía el disco solo cuando hace falta y no lo reduce nunca. Un crecimiento accidental fija un costo mensual hasta que se migre la instancia.

Modelo mental

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

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. Llegar a Cloud SQL sin exponerlo: el rango que nadie reservó

Una instancia de Cloud SQL no vive en tu VPC. Vive en una red gestionada por Google, y eso deja tres formas de llegar a ella, con consecuencias muy distintas:

```
IP pública + redes autorizadas   el tráfico sale de tu red y vuelve
                                 la lista de direcciones hay que mantenerla
                                 y una IP dinámica la rompe
acceso a servicios privados      emparejamiento con la red de Google
                                 dirección interna, tráfico que no sale
Private Service Connect          extremo con IP interna elegida por ti
                                 el modelo de la clase 051
```

La primera es la que sale por defecto y la que hay que evitar. La segunda es la habitual, y tiene un requisito que se descubre a mitad del despliegue: consume un rango de tu espacio de direcciones, que hay que reservar explícitamente.

```
$ gcloud compute addresses create rango-servicios-google \
    --global --purpose VPC_PEERING --prefix-length 20 \
    --network vpc-cloudshop --addresses 10.90.0.0
$ gcloud services vpc-peerings connect --service servicenetworking.googleapis.com \
    --ranges rango-servicios-google --network vpc-cloudshop
```

Un /20 para empezar, y conviene que sea holgado: el rango se comparte entre todos los servicios gestionados que usen este mecanismo —Cloud SQL, Memorystore, algunas versiones de otros productos— y ampliarlo después obliga a recrear el emparejamiento. Es exactamente el tipo de reserva que la clase 051 pedía hacer al planificar la subred, junto con los rangos secundarios de Kubernetes.

Y sobre cómo se autentica la aplicación, la vía correcta elimina dos gestiones a la vez:

```
$ ./cloud-sql-proxy --private-ip --auto-iam-authn \
    cls-datos-prod-euw1-01:europe-west1:pg-pedidos

--private-ip      usa la dirección interna, no la pública
--auto-iam-authn  autentica con la identidad de la carga (clase 050)
                  → no hay contraseña de base de datos que guardar ni rotar
```

La segunda línea cierra el círculo que empezó en la clase 026 y siguió en la 038: el motor de datos deja de tener una credencial propia. El usuario de base de datos se corresponde con una cuenta de servicio, y quitarle el acceso es quitarle un rol.

Dos comportamientos de la alta disponibilidad que hay que anticipar:

La conmutación produce errores transitorios. Una instancia con alta disponibilidad mantiene una réplica en espera en otra zona, y la conmutación tarda alrededor de un minuto. Durante ese minuto, las conexiones fallan. Es la cuarta vez que este hecho aparece en el programa —la clase 048 lo descubrió midiendo 1.104 peticiones perdidas con las alarmas en verde— y merece enunciarse como propiedad general: en cualquier base de datos gestionada, la conmutación es parte del funcionamiento normal y el cliente tiene que reintentar. Ninguna cantidad de infraestructura sustituye ese reintento.

El mantenimiento reinicia. A diferencia de Compute Engine, que migra en vivo (clase 052), los servicios gestionados de datos aplican mantenimiento con un reinicio dentro de una ventana declarada. Elegir esa ventana y declararla como parte del contrato del servicio es trabajo de diseño, no un trámite:

```
$ gcloud sql instances patch pg-pedidos \
    --maintenance-window-day SUN --maintenance-window-hour 3 \
    --maintenance-release-channel production
```

Y una puerta de un solo sentido con precio mensual: el almacenamiento crece automáticamente y no se reduce nunca. Una tabla de registro descontrolada que lleve el disco de 500 GB a 4 TB deja ese costo fijado hasta que alguien migre los datos a una instancia nueva. Conviene poner un límite al crecimiento automático y una alerta sobre el uso, en vez de confiar en que nada crezca.

2. Spanner: qué se está comprando y qué exige a cambio

Spanner es un motor relacional que escala horizontalmente la escritura y mantiene consistencia externa entre regiones. Es la única pieza del programa que ofrece las tres cosas a la vez, y la conversación honesta empieza por el precio:

```
~0,90 USD por nodo y hora en configuración regional
→ ~657 USD por nodo y mes, más almacenamiento
una configuración de producción razonable son 3 nodos: ~1.970 USD/mes
```

Comparado con una instancia de Cloud SQL equivalente en capacidad para una carga de una región:

```
Cloud SQL 4 vCPU / 16 GB con alta disponibilidad   ~320 USD/mes
Spanner, 3 nodos                                  ~1.970 USD/mes
```

Seis veces. Eso no descalifica a Spanner: sitúa la pregunta. ¿Qué hace este caso que solo Spanner resuelve?

```
sí lo justifica
  escritura que no cabe en una instancia vertical, con esquema relacional
  consistencia fuerte entre regiones, sin ventana de replicación
  crecimiento sin ventanas de mantenimiento ni fragmentación manual

no lo justifica
  "queremos que escale" sin una cifra que lo respalde
  volumen que cabe en una instancia grande de Cloud SQL
  necesidad de extensiones o funciones específicas de PostgreSQL
```

Y si se elige, exige una disciplina de esquema que no es opcional. La clave primaria decide el rendimiento, y una clave creciente lo destruye:

```
-- concentra TODAS las escrituras en la última división
CREATE TABLE pedidos (
  creado TIMESTAMP NOT NULL,
  pedido_id STRING(36) NOT NULL,
) PRIMARY KEY (creado, pedido_id);

-- reparte: el identificador aleatorio distribuye por el espacio de claves
CREATE TABLE pedidos (
  pedido_id STRING(36) NOT NULL,   -- UUID v4
  creado TIMESTAMP NOT NULL,
) PRIMARY KEY (pedido_id);
```

El mecanismo del fallo es el mismo que la clase 042 describió para Cosmos DB y la 031 para DynamoDB, con una diferencia importante: aquí no hay un límite duro que produzca un error. La escritura no falla; simplemente no pasa de un techo, y añadir nodos no lo mueve porque el trabajo sigue cayendo en la misma división. El síntoma es una gráfica

de rendimiento plana con la CPU de un nodo alta y la de los demás baja.

Es la tercera aparición del mismo problema en el programa, con tres formas de manifestarse:

```
DynamoDB    partición caliente: limitación
Cosmos DB   partición lógica llena: error 403, la escritura FALLA
Spanner     división caliente: techo de rendimiento, sin ningún error
```

La versión de Spanner es la más difícil de diagnosticar precisamente porque no falla nada.

Y la herramienta de localidad propia del motor, las tablas intercaladas, guarda las filas hijas físicamente junto a la padre:

```
CREATE TABLE lineas_pedido (
  pedido_id STRING(36) NOT NULL,
  linea_id INT64 NOT NULL,
) PRIMARY KEY (pedido_id, linea_id),
  INTERLEAVE IN PARENT pedidos ON DELETE CASCADE;
```

Leer un pedido con todas sus líneas pasa a ser una sola lectura local en vez de una unión distribuida. Es la misma idea que la clave de partición compartida de la clase 031, expresada en el esquema.

3. Firestore: el precio es el tamaño del resultado

Firestore es una base de datos de documentos con dos propiedades que definen cómo se diseña encima de ella.

Toda consulta necesita un índice. Los de un solo campo se crean solos; los compuestos se declaran. Y una consulta sin índice no se ejecuta lentamente: falla, con un mensaje que incluye el enlace para crear el que falta.

```
FAILED_PRECONDITION: The query requires an index.
```

Eso es una buena noticia por la misma razón que lo era el fallo ruidoso de la clase 051: no existe la consulta que funciona en desarrollo con mil documentos y tumba producción con diez millones, porque no llega a ejecutarse sin índice. El coste de esta propiedad es que el conjunto de consultas posibles se declara por adelantado, así que un patrón de acceso nuevo exige un despliegue de índices.

El precio es el número de documentos. No hay unidades de solicitud ni horas de instancia:

```
lecturas   ~0,03 USD por 100.000 documentos
escrituras ~0,09 USD por 100.000
borrados   ~0,01 USD por 100.000
```

La consecuencia práctica cambia el diseño de la aplicación más que cualquier ajuste de infraestructura:

```
una consulta que devuelve 40.000 documentos cuesta 40.000 lecturas
aunque la aplicación solo muestre un total
```

Un panel que calcula «pedidos de hoy» leyendo todos los pedidos de hoy paga por cada uno, cada vez que alguien abre la página. La aritmética se hace grande deprisa:

```
40.000 documentos por carga × 12.000 cargas al día × 30 días
= 14.400 millones de lecturas al mes
× 0,03 / 100.000 = 4.320 USD/mes por un contador
```

Las dos correcciones son de modelo de datos, no de configuración:

```
1. consultas de agregación (COUNT, SUM) — se facturan por índice leído,
   no por documento devuelto
2. contadores materializados: un documento que la escritura actualiza
   y el panel lee de uno en uno
```

La segunda trae de vuelta el problema de los puntos calientes en su forma documental: un documento admite del orden de una escritura por segundo de forma sostenida. Un contador global actualizado por cada pedido se convierte en el cuello de botella de todo el sistema. La solución conocida es repartirlo:

```
contador repartido en N documentos, elegido al azar al escribir
lectura = suma de los N
con N = 10, se admiten ~10 escrituras por segundo
```

Y los modos conviene aclararlos porque el nombre confunde: Firestore tiene modo nativo y modo Datastore, la elección se hace al crear la base de datos y no se cambia. El nativo es el que trae escuchas en tiempo real y bibliotecas para cliente móvil; el modo Datastore existe por compatibilidad. Para todo lo nuevo, nativo.

Sobre Memorystore basta con situarlo, porque las lecciones de la clase 042 se aplican sin cambios: nivel básico sin réplica y sin acuerdo de servicio —no para producción—, política de expulsión que debe considerar todas las claves, TTL siempre, y una sola conexión reutilizada por proceso. Lo único propio es que se conecta por el mismo mecanismo de acceso a servicios privados que Cloud SQL, así que consume del mismo rango reservado.

4. Elegir con una tabla y defenderlo con un número

La discusión sobre qué motor usar se repite en cada equipo hasta que alguien la escribe. Escrita, cabe aquí:

Pregunta	Cloud SQL	Spanner	Firestore
¿SQL completo y extensiones?	Sí	Subconjunto	No
¿Transacciones entre entidades?	Sí	Sí, entre regiones	Sí, acotadas
¿Escritura que escala horizontalmente?	No	Sí	Sí
¿Consultas no previstas?	Sí	Sí	No: hacen falta índices
¿Escucha en tiempo real?	No	No	Sí
Se paga por	Instancia y hora	Nodo y hora	Documento
Costo de partida	320 USD/mes	1.970 USD/mes	Casi cero

La última fila engaña si se lee sola. Firestore empieza casi gratis y su costo crece con el uso de la aplicación, no con el tamaño de los datos: una aplicación con pocos datos y muchas lecturas puede costar más que una instancia de Cloud SQL con terabytes. Al revés, Cloud SQL cobra la instancia esté ociosa o no.

La forma defendible de decidir es estimar el costo de la carga real en los candidatos, con las cifras que se conocen:

```
para CloudShop, catálogo con 4M de productos y 30M de lecturas al mes:
  Cloud SQL 4 vCPU con alta disponibilidad          ~320 USD/mes
  Firestore: 30M lecturas × 0,03/100.000              ~9 USD/mes
             + escrituras y almacenamiento             ~40 USD/mes
  → Firestore, con diferencia

para pedidos, con transacciones entre pedido, stock y pago:
  Firestore obliga a rediseñar la transacción
  Cloud SQL lo resuelve sin cambiar el modelo
  → Cloud SQL
```

Ese ejercicio de dos párrafos evita la discusión de dos semanas, y deja escrito por qué. Lo que hay que exigir a cualquier propuesta es exactamente eso: una cifra por candidato para la carga real, no una preferencia.

Y las tres decisiones de esta clase que no se deshacen editando una configuración, que conviene señalar antes de tomarlas:

1. la clave primaria de una tabla de Spanner
   cambiarla es crear otra tabla y migrar
2. el modo de Firestore, nativo o Datastore
   se elige al crear la base de datos
3. el almacenamiento de Cloud SQL, que crece y no baja
   revertirlo es volcar y restaurar en una instancia nueva

Las tres tienen la misma forma que las de la clase 042 y la misma respuesta: se toman con la evidencia delante, se documentan como decisión con su alternativa descartada, y se protegen con un límite o una alerta que avise antes de que sea tarde.

5. Diagnosticar por el límite que no es la CPU

La clase 042 cerró con una afirmación que aquí se confirma con otros tres motores: el recurso que se agota primero casi nunca es la CPU, y el panel por defecto muestra la CPU.

Lo que hay que mirar en cada uno:

```
Cloud SQL
  conexiones activas frente al máximo    un agotamiento de conexiones
                                          parece lentitud del motor
  retraso de la réplica de lectura        informes leyendo datos viejos
  IOPS del disco                          ligadas al TAMAÑO del disco,
```

```
                                        igual que en la clase 052
  espera de bloqueos                    una transacción larga bloquea a todas

Spanner
  utilización de CPU POR DIVISIÓN       la media engaña; el punto caliente
                                        es una división al 100 %
  latencia por operación                y su percentil, no la media

Firestore
  lecturas por operación                el indicador de costo y de diseño
  contención en documentos              escrituras al mismo documento
```

La fila de las IOPS de Cloud SQL merece subrayarse porque es la cuarta aparición del mismo patrón: el rendimiento del disco depende del tamaño del disco, así que una instancia con un disco pequeño tiene un techo de E/S bajo aunque le sobre CPU. Ampliar el disco lo sube — y como el almacenamiento no baja, esa ampliación es permanente.

El orden de diagnóstico que sirve para los tres, y que ya es el mismo de las partes 02 y 03:

```
1. ¿qué límite está al 100 %?   conexiones, E/S, bloqueos, una división
2. ¿desde cuándo?               percentil de latencia en serie temporal
3. ¿qué cambió en esa ventana?  despliegue, esquema, índice, volumen
4. ¿qué consulta concreta?      Query Insights o el registro de consultas lentas
```

Y una recomendación de operación que evita la mitad de los incidentes de esta clase: medir el costo de las consultas en el mismo sitio donde se mide su latencia. En Firestore eso significa registrar cuántos documentos devolvió cada consulta; en Cloud SQL, cuántas filas examinó frente a cuántas devolvió. Las dos cifras detectan el mismo problema —una consulta que lee mucho para devolver poco— y lo detectan el día del despliegue en lugar de en la factura del mes siguiente. Es el mismo principio de la cabecera de costo por consulta de la clase 042: lo que tiene número se discute; lo que no, se opina.

Ejemplo trabajado

CloudShop reparte su capa de datos en Google Cloud. La decisión inicial es razonable, el despliegue falla por una razón de red, y después aparecen cuatro problemas de los cuales dos ya se habían visto en otras plataformas.

Decisión inicial, documentada con cifras:

```
pedidos      Cloud SQL PostgreSQL, alta disponibilidad   transacciones entre entidades
catálogo     Firestore modo nativo                       30M lecturas/mes, sin uniones
sesiones     Memorystore Redis estándar                   caché con TTL
inventario   pendiente de decidir                         se propone Spanner
```

Incidente 1 — la aplicación no puede conectar, y la solución rápida saca el tráfico de la VPC.

```
$ gcloud sql instances describe pg-pedidos --format="value(ipAddresses[].type)"
PRIMARY
```

Solo IP pública. Para conectar en privado hacía falta acceso a servicios privados, y este exige un rango reservado que no estaba en el plan de direcciones de la clase 051. La solución de urgencia fue autorizar el rango de salida del NAT, lo que significaba que el tráfico salía de la red y volvía.

```
camino del tráfico              sale por NAT y vuelve   dirección interna
rango reservado para servicios       ninguno            10.90.0.0/20
autenticación                  contraseña en secreto    IAM con el proxy
contraseñas de base de datos            2                    0
IP pública de la instancia            activa              desactivada
```

Y la prueba negativa:

```
$ psql "host=34.78.x.x dbname=pedidos" -c "select 1"
psql: error: connection to server ... failed: Connection timed out          ✓
```

Incidente 2 — la conmutación de alta disponibilidad, por cuarta vez en el programa.

```
simulacro de conmutación
  duración                      62 s
  errores devueltos al cliente  847
  pedidos no registrados        31
```

El mismo hallazgo de la clase 048, con otro motor y otros códigos de error. El equipo ya tenía la corrección escrita del capstone anterior y la aplicó directamente:

```
reintento de errores transitorios       no            sí
```

```
tiempo de espera de conexión        30 s (largo)   5 s + reintento
errores en la conmutación repetida     847             0
```

Es la primera vez en el programa que un problema conocido se corrige antes de causar daño en la plataforma nueva. Ese es el valor concreto del contrato portable de la clase 048.

Incidente 3 — el disco que creció a 4 TB y no baja.

Una tabla de auditoría sin política de retención creció durante siete semanas.

```
$ gcloud sql instances describe pg-pedidos \
    --format="value(settings.dataDiskSizeGb,settings.storageAutoResize)"
4096   True
$ psql -c "SELECT pg_size_pretty(pg_total_relation_size('auditoria'))"
3312 GB
```

Se purgó la tabla y el disco siguió en 4 TB, porque el crecimiento automático no se revierte.

```
disco                               4.096 GB      500 GB (instancia nueva)
costo mensual del almacenamiento     ~697 USD         ~85 USD
límite de crecimiento automático     ninguno          800 GB
alerta de uso de disco               ninguna          > 70 %
retención de la tabla de auditoría   ninguna     90 días, exportada a
                                                 Cloud Storage (clase 053)
migración necesaria                     —        volcado y restauración, 4 h
```

Seiscientos doce dólares al mes y cuatro horas de migración por un límite que se configura en un minuto.

Incidente 4 — Spanner sí, pero no como se propuso.

La propuesta era «inventario en Spanner porque escala». La evaluación con cifras:

```
volumen de inventario                  18 GB
escrituras en el pico                1.400 por segundo
requisito real          stock coherente entre tres regiones,
                        sin vender dos veces la última unidad
```

El volumen cabía en Cloud SQL de sobra; lo que no cabía era el requisito de consistencia entre regiones sin ventana de replicación. Eso sí es lo que solo Spanner da, y quedó escrito así en la decisión.

Pero la primera implementación no llegaba a 1.200 escrituras por segundo con tres nodos:

```
PRIMARY KEY (actualizado, sku)   -- marca de tiempo primero
```

```
utilización media de CPU        38 %
utilización de la división caliente   99 %
```

Ninguna escritura falló: simplemente había un techo. Es la tercera versión del mismo problema del programa y la más difícil de ver, porque no produce ningún error.

```
clave primaria               (actualizado, sku)    (sku)  — reparte por hash
tablas relacionadas             tablas aparte      intercaladas en el padre
escrituras por segundo             1.180              6.400
nodos                                3                  3
lectura de un SKU con su historial  2 consultas    1 lectura local
```

El rendimiento se multiplicó por cinco sin añadir capacidad. Es la misma lección de la clase 042 con otro motor: el precio de un modelo de datos equivocado no se paga con infraestructura.

Incidente 5 — un panel que costaba más que la base de datos.

```
factura de Firestore, mes 2
  lecturas   14.380 millones     4.314 USD
  escrituras     41 millones        37 USD
  almacenamiento                    12 USD
```

El panel de operaciones cargaba «pedidos de hoy» leyendo todos los documentos del día para contarlos, en cada carga de página.

```
cálculo del total                 leer 40.000 docs   consulta de agregación
contador de pedidos                     —            documento repartido en 10
lecturas mensuales               14.380 millones     62 millones
costo mensual de Firestore          4.363 USD          31 USD
```

Y se añadió la medida que evita la repetición: el número de documentos devueltos se registra en cada consulta, junto a su latencia, y aparece en el mismo panel.

Resumen de la capa de datos:

```
camino a Cloud SQL                          público          privado
contraseñas de base de datos                   2               0
errores en conmutación de alta disponibilidad 847              0
almacenamiento de Cloud SQL                  4.096 GB        500 GB
escrituras por segundo en Spanner             1.180           6.400
lecturas mensuales de Firestore           14.380 M            62 M
costo mensual de la capa de datos           7.190 USD       2.470 USD
```

La lección que esta clase traslada al resto de la parte 04: de los cinco problemas, dos eran conocidos —la conmutación que exige reintento y la clave que concentra escrituras— y uno de ellos se corrigió antes de causar daño porque estaba escrito en el contrato de la clase 048. Los otros tres eran propios de la plataforma, y los tres tenían la misma forma: un valor por defecto cómodo que fija un costo o un límite difícil de revertir. La IP pública, el crecimiento automático de disco y el precio por documento no fallan; simplemente cobran.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-04-gcp-core-platform/054-cloud-sql-spanner-firestore-y-memorystore/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-gcp 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-gcp 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 no puede conectar a Cloud SQL por dirección interna	El acceso a servicios privados exige un rango reservado del propio espacio de direcciones	Reserva un rango holgado al planificar la red y conecta con el proxy autenticado por IAM, sin IP pública.
Se purga una tabla enorme y el costo de almacenamiento no baja	El crecimiento automático de Cloud SQL amplía el disco y nunca lo reduce	Pon un límite al crecimiento y una alerta de uso; revertir exige volcar y restaurar en una instancia nueva.
Spanner no supera un techo de escrituras pese a tener nodos libres	La clave primaria es creciente y concentra las escrituras en una división	Usa una clave que reparta —identificador aleatorio o secuencia invertida— e intercala las tablas hijas.
La factura de Firestore es mayor que la de la base de datos relacional	El precio es el número de documentos devueltos y un panel los leía todos para contar	Usa consultas de agregación o contadores materializados repartidos, y registra los documentos devueltos junto a la latencia.
Una consulta nueva falla en producción con FAILEDPRECONDITION	Firestore exige un índice para cada consulta y el compuesto no estaba declarado	Declara los índices como parte del despliegue; el fallo es ruidoso y es preferible a una consulta que degrada en silencio.
Se pierden peticiones durante un mantenimiento planificado	Los servicios gestionados de datos reinician en su ventana, a diferencia de la migración en vivo de Compute Engine	Declara la ventana, reintenta los errores transitorios y prueba la conmutación como parte del simulacro.

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 hay que reservar para llegar a Cloud SQL por red privada, y en qué clase debía haberse planificado?
2. ¿Qué exige un caso para justificar el sobrecoste de Spanner frente a Cloud SQL?
3. ¿Cómo se manifiesta un punto caliente en Spanner y por qué es más difícil de diagnosticar que en Cosmos DB?
4. Calcula el costo mensual de un panel que lee 25.000 documentos y se carga 8.000 veces al día.
5. ¿Cuál es la cuarta aparición en el programa del problema de los errores transitorios, y qué lo corrige?

Referencias

- Google Cloud (2025). Configure private IP for Cloud SQL — acceso a servicios privados y rango reservado. <https://cloud.google.com/sql/docs/postgres/configure-private-ip>
- Google Cloud (2025). About the Cloud SQL Auth Proxy — autenticación con IAM y conexión cifrada. <https://cloud.google.com/sql/docs/postgres/sql-proxy>
- Google Cloud (2025). Schema design best practices in Spanner — claves primarias, puntos calientes y tablas intercaladas. <https://cloud.google.com/spanner/docs/schema-design>
- Google Cloud (2025). Understand Firestore billing — lecturas, escrituras y costo de las consultas. <https://cloud.google.com/firestore/docs/billing-example>
- Google Cloud (2025). Distributed counters in Firestore — límite de escritura por documento y contadores repartidos. <https://cloud.google.com/firestore/docs/solutions/counters>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 04 en PDF · Recorrido de Google Cloud en PDF · Manual integral

Evaluación — 054 Cloud SQL, Spanner, Firestore y Memorystore

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-gcp 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.

055 — Cloud Run, Cloud Functions y API Gateway

Parte: 04 — Google Cloud: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: serverless · Estado: EXECUTABLECORE

Propósito

Ejecutar aplicaciones sin gestionar servidores en Google Cloud, donde la pieza central se comporta de una forma que rompe el hábito adquirido en las clases 032 y 043: una instancia de Cloud Run atiende muchas peticiones a la vez. Esa sola propiedad cambia el costo por un factor cercano a diez, cambia lo que el código puede suponer sobre su propio aislamiento, y hace que el ajuste por defecto —la CPU solo se asigna durante la petición— produzca el fallo más desconcertante de la clase: trabajo que se planifica y nunca se ejecuta.

Resultados de aprendizaje

Al finalizar podrás:

1. Explicar el efecto de la concurrencia por instancia sobre el costo y sobre los requisitos del código.
2. Diagnosticar por qué desaparecen registros, métricas o tareas en segundo plano según la asignación de CPU.
3. Desplegar revisiones con reparto de tráfico y una URL propia para probar antes de dirigir usuarios.
4. Autenticar llamadas entre servicios con testigos de identidad en vez de dejarlos públicos.
5. Elegir entre salida directa a la VPC y conector, y entre servicio, trabajo y pasarela de API.

Conceptos centrales

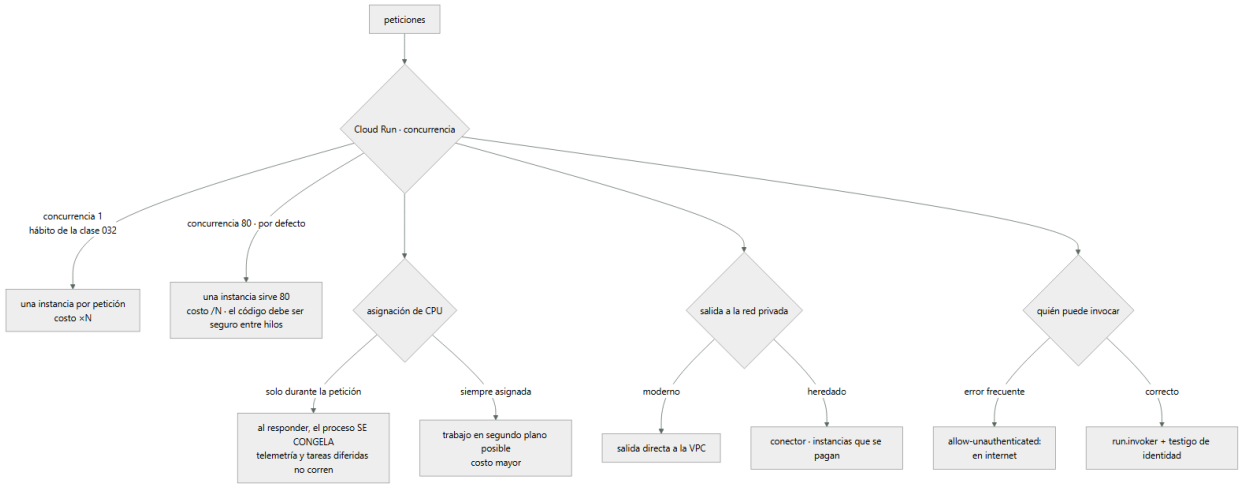
Concepto	Comprensión verificable
concurrencia por instancia	Número de peticiones que una misma instancia atiende a la vez, 80 por defecto. Es lo que separa el modelo de Cloud Run del de una función por petición, y lo que divide el costo.
asignación de CPU	Por defecto solo durante la petición. Fuera de ella el proceso está congelado: lo que se programó para «después de responder» no se ejecuta hasta la petición siguiente, si la hay.
instancias mínimas	Instancias mantenidas vivas para evitar el arranque en frío. Convierten la latencia del arranque en un costo fijo, con el mismo criterio de la clase 032.
revisión con etiqueta	Versión inmutable con URL propia. Permite probar una revisión con tráfico real dirigido a mano antes de asignarle un porcentaje del tráfico general.
testigo de identidad	Credencial que un servicio presenta a otro para demostrar quién es. Con roles/run.invoker, sustituye a dejar el servicio abierto a internet.
trabajo de Cloud Run	Ejecución por lotes con tareas paralelas y reintentos, sin servidor HTTP. Es la respuesta al proceso largo que en la clase 043 chocaba con un tope de diez minutos.

Modelo mental

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

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 instancia atiende ochenta peticiones, y eso cambia la aritmética

La clase 032 estableció que en serverless la concurrencia es el recurso que se agota, con un modelo donde cada petición ocupa un entorno de ejecución. La clase 043 mantuvo esa forma. Cloud Run funciona al revés:

```
modelo de función por petición   1 petición  → 1 entorno ocupado
Cloud Run                         N peticiones → 1 instancia, hasta la concurrencia
```

La consecuencia inmediata es de costo, y es grande. Con 10 millones de peticiones al mes de 200 ms, 1 vCPU y 512 MiB:

```
concurrencia 1
  10.000.000 × 0,2 s = 2.000.000 vCPU-s × 0,000024 =  48,00 USD
  memoria                                            +  2,50
  peticiones  10 M × 0,40/millón                     +  4,00
                                                     ██████████
                                                       54,50 USD/mes

concurrencia 80 (con empaquetado imperfecto y algo de tiempo ocioso)
                                                       ~8,90 USD/mes
```

Seis veces menos por un parámetro. Y el error que produce la cifra alta es exactamente el que comete quien llega de las clases 032 o 043: poner la concurrencia a 1 «para que cada petición esté aislada», que es como funcionaba allí.

Pero el aislamiento que se pierde es real y hay que entenderlo, porque la concurrencia impone dos requisitos al código:

1. tiene que ser seguro entre peticiones simultáneas
   una variable global mutable la comparten 80 peticiones a la vez
2. su consumo de memoria y de conexiones se multiplica por la concurrencia
   80 peticiones × una conexión a la base de datos cada una = 80 conexiones
   por instancia

La segunda es la que produce el incidente sutil: un servicio con 20 instancias y concurrencia 80 puede abrir 1.600 conexiones a una base de datos que admite 200. La corrección es un grupo de conexiones acotado por instancia, dimensionado como máximo de conexiones / instancias máximas, y no como «una por petición».

La primera produce algo peor: si un objeto global guarda estado de la petición en curso —un cliente con una cabecera de usuario, un contexto que se reasigna—, dos peticiones simultáneas se pisan y un usuario ve datos de otro. No hay error, no hay traza: hay una respuesta incorrecta. Es el tipo de fallo que aparece solo bajo carga y desaparece al reproducirlo.

La forma de elegir el valor no es dejar el que viene:

```
servicio con trabajo ligado a E/S (espera a la base de datos, a otra API)
  → concurrencia alta: la instancia está esperando, no calculando
servicio con trabajo intensivo de CPU (imágenes, cifrado, compresión)
  → concurrencia baja: las peticiones compiten por el mismo núcleo
servicio con memoria proporcional a la petición
  → concurrencia limitada por la memoria de la instancia
```

Y se comprueba midiendo, no razonando: se sube la concurrencia y se observa el percentil 95. Cuando empieza a subir, se ha pasado el punto donde las peticiones compiten.

2. La CPU solo durante la petición: el trabajo que nunca se ejecuta

Este es el fallo más desconcertante de la clase, porque no produce ningún error y desaparece cuando se investiga.

Por defecto, Cloud Run asigna CPU únicamente mientras se procesa una petición. En cuanto el servicio responde, el proceso se congela: sigue en memoria y no ejecuta nada.

```
recibe petición  → CPU asignada → procesa → responde → CPU RETIRADA
                                                       el proceso se congela
```

Todo lo que el código haga «después de responder» queda pendiente hasta la petición siguiente:

```
volcado de telemetría por lotes      los registros no aparecen, o aparecen tarde
                                     y atribuidos a otra petición
métricas acumuladas en memoria       se pierden si la instancia se retira
tareas diferidas tras responder      no se ejecutan
mantenimiento del grupo de conexiones no se ejecuta: conexiones muertas
reintentos programados               no se ejecutan
```

El síntoma que llega al equipo suele ser «faltan registros» o «la última petición de cada rato falla con conexión cerrada», y ninguno de los dos apunta a la causa.

Hay dos respuestas, y la elección tiene precio:

```
# opción A: hacer el trabajo ANTES de responder
#   correcto, y añade latencia a la petición

# opción B: CPU siempre asignada
$ gcloud run deploy svc-tienda --no-cpu-throttling --min-instances 1
```

La opción B cobra la CPU también fuera de la petición, a una tarifa menor, y es la única que permite trabajo real en segundo plano. Es obligatoria si el servicio mantiene una conexión persistente, consume de una suscripción o procesa en un hilo propio.

La regla de decisión, corta:

```
servicio HTTP puro, sin nada que hacer entre peticiones   por defecto
cualquier cosa que deba ocurrir sin una petición en curso  CPU siempre asignada
```

Y sobre el arranque en frío, el método de la clase 032 se aplica sin cambios —comparar con el percentil del SLO, no con la media— y las palancas aquí son tres:

```
$ gcloud run deploy svc-tienda \
    --min-instances 2 \            # instancias vivas: latencia → costo fijo
    --cpu-boost \                  # más CPU durante el arranque
    --execution-environment gen2
```

La imagen del contenedor también pesa: una imagen de 900 MB arranca claramente más despacio que una de 80 MB, y esa es una optimización que no cuesta dinero, solo un Dockerfile con varias etapas.

Y un tercer modo de ejecución que resuelve el problema con el que chocó la clase 043 —el proceso de catorce minutos contra un tope de diez—: los trabajos de Cloud Run.

```
$ gcloud run jobs create cierre-mensual --image $IMAGEN \
    --tasks 12 --parallelism 4 --max-retries 3 --task-timeout 3600
$ gcloud run jobs execute cierre-mensual --wait
```

No hay servidor HTTP, hay tareas con índice, paralelismo, reintentos y un tiempo límite de horas. Es la forma correcta de ejecutar un cierre, una migración o una reindexación, y evita la deformación de convertir un proceso por lotes en un servicio web que se llama a sí mismo.

3. Revisiones, reparto de tráfico y quién puede invocar

Cada despliegue crea una revisión inmutable, y el tráfico se dirige por porcentaje o por etiqueta:

```
$ gcloud run deploy svc-tienda --image $IMAGEN --tag rc8 --no-traffic
# → https://rc8---svc-tienda-xxxx.a.run.app  : URL propia, 0 % del tráfico

$ gcloud run services update-traffic svc-tienda --to-tags rc8=10
$ gcloud run services update-traffic svc-tienda --to-revisions svc-tienda-rc8=100
$ gcloud run services update-traffic svc-tienda --to-revisions svc-tienda-rc7=100  # vuelta atrás
```

La etiqueta con --no-traffic es la pieza que faltaba en los espacios de la clase 043: una URL real, con tráfico real dirigido a mano, sin que ningún usuario llegue a ella por accidente. El canario posterior por porcentaje es el mismo mecanismo de las revisiones de Container Apps, y con la misma condición: hace falta una señal por revisión para que repartir 90/10 signifique algo. La clase 057 monta esa parte.

La vuelta atrás es un cambio de porcentaje, así que dura segundos. Es la misma propiedad que hacía valiosos los espacios de App Service, sin el problema de los ajustes que viajan.

Y ahora la decisión de seguridad que más veces está mal en despliegues nuevos:

```
$ gcloud run deploy svc-pedidos --allow-unauthenticated
```

Esa opción publica el servicio en internet, sin autenticación, con una URL adivinable por su patrón. Es correcta para el frontal público y es un error para todo lo demás: un servicio interno desplegado así es una API abierta que ningún firewall protege, porque no está en la VPC.

El patrón correcto para llamadas entre servicios:

```
$ gcloud run deploy svc-pedidos --no-allow-unauthenticated \
    --service-account sa-pedidos@cls-tienda-prod-euw1-01.iam.gserviceaccount.com
$ gcloud run services add-iam-policy-binding svc-pedidos \
    --member "serviceAccount:sa-tienda@cls-tienda-prod-euw1-01.iam.gserviceaccount.com" \
    --role roles/run.invoker
```

Y el llamante presenta un testigo de identidad obtenido de su propia cuenta de servicio, sin ninguna credencial almacenada:

```
import google.auth.transport.requests, google.oauth2.id_token

destino = "https://svc-pedidos-xxxx.a.run.app"
testigo = google.oauth2.id_token.fetch_id_token(
    google.auth.transport.requests.Request(), destino)
respuesta = requests.post(f"{destino}/pedidos", json=datos,
                          headers={"Authorization": f"Bearer {testigo}"})
```

La prueba negativa correspondiente es de una línea y debería estar en el guion de verificación:

```
$ curl -s -o /dev/null -w '%{http_code}\n' https://svc-pedidos-xxxx.a.run.app/pedidos
403                                                                          ✓
```

Para la salida hacia la red privada —alcanzar Cloud SQL o Memorystore de la clase 054— hay dos mecanismos y uno está obsoleto:

```
salida directa a la VPC    la instancia obtiene una dirección de tu subred
                           sin infraestructura intermedia que pagar, escala mejor
conector de acceso serverless  máquinas gestionadas que hacen de puente
                           se pagan por hora y son un cuello de botella propio
```

```
$ gcloud run deploy svc-pedidos --network vpc-cloudshop \
    --subnet snet-serverless-euw1 --vpc-egress private-ranges-only
```

private-ranges-only envía por la VPC solo lo dirigido a rangos privados y deja salir el resto por el camino de siempre; all-traffic lo manda todo, que es lo que hace falta si la salida debe pasar por Cloud NAT para tener una dirección de origen estable. La subred necesita espacio suficiente: es otra reserva que la clase 051 debía haber previsto.

4. Funciones y pasarelas: menos productos de los que parece

Dos aclaraciones ahorran una discusión de arquitectura entera.

Las funciones de segunda generación son Cloud Run. Se ejecutan sobre la misma plataforma, con el mismo modelo de concurrencia, la misma asignación de CPU y las mismas revisiones. Lo que cambia es el empaquetado: en vez de una imagen de contenedor se entrega código fuente y la plataforma construye la imagen.

```
función         despliegas código; la plataforma construye
                disparadores integrados de eventos
servicio        despliegas una imagen; controlas el entorno completo
```

La consecuencia práctica es que la elección es de empaquetado, no de capacidades, y que todo lo aprendido sobre concurrencia y CPU se aplica igual a las dos. Y que migrar de una a otra es cambiar cómo se despliega, no rehacer la aplicación.

Los disparadores de eventos son la parte propia, y conectan con la clase 056:

```
$ gcloud functions deploy indexar-factura --gen2 --runtime python312 \
    --trigger-event-filters="type=google.cloud.storage.object.v1.finalized" \
    --trigger-event-filters="bucket=cls-facturas" \
    --run-service-account sa-indexador@cls-tienda-prod-euw1-01.iam.gserviceaccount.com
```

La pasarela de API hace falta menos veces de las que se cree. Hay tres productos y conviene situarlos antes de elegir:

	Qué aporta	Cuándo
Balanceador global + Cloud Armor	Enrutamiento, TLS, WAF, CDN, una IP global	La mayoría de los casos
API Gateway	Contrato OpenAPI, claves de API, autenticación por API	Exponer API a terceros con contrato
Apigee	Gestión completa: monetización, portal, cuotas por cliente	Programa de API como producto

La primera fila cubre lo que la mayoría de los equipos busca cuando dice «necesitamos una pasarela»: un punto de entrada, TLS, reglas de seguridad y reparto por ruta. Eso ya está montado desde la clase 052 y no añade un salto más en la ruta de la petición.

API Gateway aporta algo distinto y concreto: un contrato declarado que valida las peticiones y gestiona claves para consumidores externos. Si no hay terceros ni contrato que imponer, añade latencia y una pieza más que operar.

Y Apigee resuelve un problema de negocio —API como producto, con planes y facturación— cuyo precio solo se justifica si ese negocio existe.

La regla que evita el sobrediseño: añade una pasarela cuando puedas nombrar la función concreta que hace y que no hace el balanceador. Si no se puede nombrar, no hace falta.

5. Qué cuesta y qué se mide

El modelo de precio de Cloud Run tiene tres componentes y una decisión que los domina:

```
CPU        ~0,000024 USD por vCPU-segundo mientras hay asignación
memoria    ~0,0000025 USD por GiB-segundo
peticiones ~0,40 USD por millón
```

Y la decisión que los domina es la concurrencia, porque divide los dos primeros. Las palancas, en orden de efecto:

```
1. concurrencia            divide el costo por instancia-segundo
2. tamaño de la instancia  1 vCPU y 512 MiB suele bastar; medir antes de subir
3. instancias mínimas      cuestan cuando no hay tráfico: solo donde el SLO lo exige
4. asignación de CPU       «siempre» cuesta más: solo donde haga falta
5. tamaño de la imagen     no cuesta dinero directamente; acorta el arranque
```

Y una comparación honesta con lo construido en las clases 032 y 043, para el mismo servicio y el mismo volumen:

```
función por petición (clase 032)              ~54 USD          sí
plan de App Service con instancias fijas      ~124 USD         no
Cloud Run, concurrencia 80, sin mínimos         ~9 USD         sí
Cloud Run, concurrencia 80, 2 mínimas           ~38 USD        no
```

La última fila es la que suele elegirse en producción: elimina el arranque en frío y sigue costando menos de un tercio que un plan con instancias fijas. Y la tercera es la que hace que un entorno de pruebas o un servicio poco usado cueste prácticamente nada.

Lo que hay que medir, y que no es lo mismo que en las plataformas anteriores:

```
utilización de la concurrencia   peticiones simultáneas frente al máximo
                                 si nunca se acerca, la concurrencia sobra
                                 y hay instancias de más
instancias activas               la curva que explica la factura
tiempo de arranque de instancia  el que decide si hacen falta mínimas
peticiones con arranque en frío  proporción, no media
conexiones abiertas por instancia el número que hay que multiplicar
                                 por el máximo de instancias
```

La última merece una alerta propia, porque su fallo no ocurre en el servicio sino en la base de datos, y llega justo en el pico: el máximo de instancias multiplicado por las conexiones por instancia debe caber en el límite del motor. Es una cuenta de una línea que casi nunca se hace y que la clase 054 dejó preparada.

```
máximo de instancias (100) × conexiones por instancia (10) = 1.000
límite de la instancia de Cloud SQL                        =   200
→ el servicio puede tumbar su propia base de datos al escalar
```

La corrección es acotar las dos cifras a la vez —--max-instances y el tamaño del grupo de conexiones—, y no solo una.

Ejemplo trabajado

CloudShop despliega su capa de aplicación en Cloud Run. La traducción desde las clases 032 y 043 funciona a la primera y cuesta seis veces de más; los cuatro problemas siguientes son consecuencias del modelo que nadie había tenido que considerar antes.

Despliegue inicial, traducido literalmente:

```
svc-tienda      concurrencia 1  · "para que cada petición esté aislada"
svc-pedidos     concurrencia 1  · --allow-unauthenticated
svc-informes    concurrencia 1  · proceso de cierre partido en trozos de 9 min
conector de acceso serverless para llegar a Cloud SQL
```

Problema 1 — la factura del primer mes.

```
svc-tienda   (10,2 M peticiones)           54,80 USD          8,90 USD
svc-pedidos  (3,1 M)                       17,40 USD          3,20 USD
svc-informes (0,4 M)                        9,10 USD          2,10 USD
                                          ■■■■■■■■■■■        ■■■■■■■■■■
                                           81,30 USD         14,20 USD
```

El cambio fue un parámetro. Y la comprobación de que la concurrencia elegida es correcta se hizo midiendo, no razonando:

```
concurrencia   p95      instancias activas en el pico
     1        118 ms              210
    40        121 ms                7
    80        124 ms                4
   200        347 ms                2     ← aquí las peticiones ya compiten
```

Se fija en 80. El percentil apenas se mueve y las instancias caen de 210 a 4.

Problema 2 — faltan registros y las conexiones mueren.

Dos síntomas aparentemente sin relación:

```
los registros de una petición aparecen atribuidos a la SIGUIENTE
la primera petición tras un rato de silencio falla con "connection reset"
```

Ambos tienen la misma causa: la CPU solo se asigna durante la petición, así que el volcado de telemetría por lotes y el mantenimiento del grupo de conexiones quedaban congelados hasta la petición siguiente.

```
asignación de CPU                    solo en petición   siempre asignada
instancias mínimas                          0                 1
volcado de telemetría                    por lotes      síncrono antes de responder
registros mal atribuidos                   ~7 %               0
fallos de primera petición                1 de 12             0
costo adicional                              —             +11 USD/mes
```

Once dólares al mes por eliminar dos clases de fallo que consumieron una semana de investigación entre los dos.

Problema 3 — un cliente ve el pedido de otro.

Un cliente reporta ver, durante un segundo, datos que no eran suyos. Ocurre menos de una vez cada diez mil peticiones y solo en horas punta.

```
# el cliente HTTP era global y guardaba la cabecera de la petición en curso
cliente = ApiClient()

def manejar(peticion):
    cliente.headers["X-Usuario"] = peticion.usuario     # ← compartido entre 80
    return cliente.get("/perfil")
```

Con concurrencia 1 esto era correcto. Con concurrencia 80, ochenta peticiones comparten ese objeto y se pisan la cabecera.

```
estado por petición              en objeto global   pasado como argumento
pruebas de concurrencia                ninguna      carga con 200 simultáneas
                                                    y verificación de aislamiento

incidencias reportadas                   3                  0
```

La medida que evita la repetición no es la corrección: es la prueba de carga con verificación de aislamiento, que comprueba que cada respuesta corresponde a su petición. Sin ella, el mismo defecto vuelve con el siguiente objeto

global.

Problema 4 — la API interna estaba en internet.

```
$ curl -s -o /dev/null -w '%{http_code}\n' https://svc-pedidos-xxxx.a.run.app/pedidos
200
```

Desplegado con --allow-unauthenticated, sin ninguna red que lo protegiera, porque Cloud Run no está en la VPC.

```
servicios públicos                        3 de 3          1 de 3
autenticación entre servicios            ninguna     testigo de identidad
permiso de invocación                   allUsers      roles/run.invoker
                                                       a cuentas concretas
prueba negativa                            no        sí, 403 sin testigo
```

Problema 5 — el conector y el cierre mensual.

Dos correcciones menores con efecto medible:

```
salida a la red privada             conector (2 instancias)  salida directa
costo del conector                        ~74 USD/mes              0
cierre mensual                     7 llamadas HTTP encadenadas  trabajo de Cloud Run
                                    con estado en un bucket     12 tareas, 4 en paralelo
duración del cierre                        52 min                14 min
fallos parciales sin reintento           sí, a mano          reintento automático
```

El cierre mensual era el proceso que en la clase 043 había obligado a usar Durable Functions por el tope de diez minutos. Aquí el mecanismo correcto existe de fábrica y además paraleliza.

Resumen de la capa de aplicación:

```
concurrencia por instancia                    1              80
instancias activas en el pico                210               4
servicios expuestos a internet              3 de 3          1 de 3
registros mal atribuidos                     ~7 %              0
incidencias de datos cruzados                  3               0
duración del cierre mensual                 52 min          14 min
costo mensual de la capa de aplicación     155 USD          64 USD
```

La lección que esta clase traslada al resto de la parte 04: los tres primeros problemas tienen la misma raíz y no la tenía ninguna plataforma anterior — una instancia sirve a muchas peticiones a la vez, y el proceso se congela cuando no hay ninguna. Ese modelo divide el costo por seis y exige del código dos propiedades que antes se podían ignorar: aislamiento entre peticiones simultáneas y ningún trabajo que dependa de ejecutarse «después». No es una configuración: es un contrato con la aplicación, y el único de los tres proveedores que lo pide de forma explícita.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-04-gcp-core-platform/055-cloud-run-cloud-functions-y-api-gateway/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-serverless-gcp 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-serverless-gcp 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
El costo por petición es varias veces mayor de lo esperado	La concurrencia se puso a 1 por costumbre del modelo de una función por petición	Sube la concurrencia midiendo el percentil 95: si no empeora, la instancia estaba desaprovechada.
Faltan registros o aparecen atribuidos a la petición siguiente	La CPU solo se asigna durante la petición y el volcado por lotes queda congelado	Vuelca antes de responder o activa la CPU siempre asignada; el trabajo en segundo plano exige lo segundo.
Bajo carga, una respuesta contiene datos de otra petición	Un objeto global mutable se comparte entre las peticiones simultáneas de la instancia	Pasa el estado por argumento y añade una prueba de carga que verifique el aislamiento entre respuestas.
Un servicio interno responde desde internet	Se desplegó con --allow-unauthenticated y Cloud Run no está dentro de la VPC	Despliega sin acceso anónimo, concede roles/run.invoker a las cuentas llamantes y verifica que sin testigo devuelve 403.
La base de datos agota sus conexiones cuando el servicio escala	Conexiones por instancia multiplicadas por el máximo de instancias supera el límite del motor	Acota --max-instances y el tamaño del grupo de conexiones a la vez, con la cuenta hecha explícitamente.
Un proceso por lotes se implementa como una cadena de llamadas HTTP con estado externo	Se arrastró la limitación de tiempo de una plataforma anterior	Usa un trabajo de Cloud Run con tareas, paralelismo y reintentos; el tiempo límite es de 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. ¿Cómo cambia el costo al pasar la concurrencia de 1 a 80, y qué dos requisitos impone eso al código?
2. ¿Qué deja de ejecutarse con la asignación de CPU por defecto, y qué dos síntomas produce?
3. ¿Cómo pruebas una revisión con tráfico real sin que ningún usuario llegue a ella?
4. ¿Por qué un servicio interno con --allow-unauthenticated no está protegido por ninguna regla de red?
5. Con 100 instancias máximas y 10 conexiones por instancia, ¿qué comprobación hay que hacer contra la base de datos?

Referencias

- Google Cloud (2025). About instance concurrency in Cloud Run — concurrencia, efecto en costo y requisitos del código. <https://cloud.google.com/run/docs/about-concurrency>
- Google Cloud (2025). CPU allocation in Cloud Run — CPU durante la petición frente a siempre asignada. <https://cloud.google.com/run/docs/configuring/cpu-allocation>
- Google Cloud (2025). Rollbacks, gradual rollouts and traffic migration — revisiones, etiquetas y reparto de tráfico. <https://cloud.google.com/run/docs/rollouts-rollbacks-traffic-migration>
- Google Cloud (2025). Authenticating service-to-service — testigos de identidad y roles/run.invoker. <https://cloud.google.com/run/docs/authenticating/service-to-service>
- Google Cloud (2025). Cloud Run jobs overview — tareas, paralelismo, reintentos y tiempos límite. <https://cloud.google.com/run/docs/create-jobs>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 04 en PDF · Recorrido de Google Cloud en PDF · Manual integral

Anterior	Índice	Siguiente
← 054 · Cloud SQL, Spanner, Firestore y Memorystore	Parte 04 · Programa	056 · Pub/Sub, Cloud Tasks y Workflows →

Evaluación — 055 Cloud Run, Cloud Functions y API Gateway

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-serverless-gcp 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.

056 — Pub/Sub, Cloud Tasks y Workflows

Parte: 04 — Google Cloud: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: messaging · Estado: EXECUTABLECORE

Propósito

Construir la columna asíncrona de Google Cloud, donde la pregunta que ordenó la clase 044 —¿el mensaje se consume o se observa?— ya no elige entre productos: la responden las suscripciones de un mismo tema. A cambio aparecen tres decisiones que allí no existían: qué tipo de suscripción, quién controla el ritmo de entrega hacia un destino que no escala, y qué significa exactamente la entrega «exactamente una vez» que esta plataforma ofrece y las anteriores no.

Resultados de aprendizaje

Al finalizar podrás:

1. Modelar consumo y observación con temas y suscripciones en vez de con dos productos distintos.
2. Configurar una cola de mensajes fallidos que de verdad reciba mensajes, y demostrarlo.
3. Elegir entre Pub/Sub y Cloud Tasks a partir de quién debe controlar el ritmo de entrega.
4. Explicar qué garantiza la entrega exactamente una vez y qué sigue exigiendo idempotencia.
5. Reproducir un intervalo de mensajes ya procesados para reparar un daño posterior.

Conceptos centrales

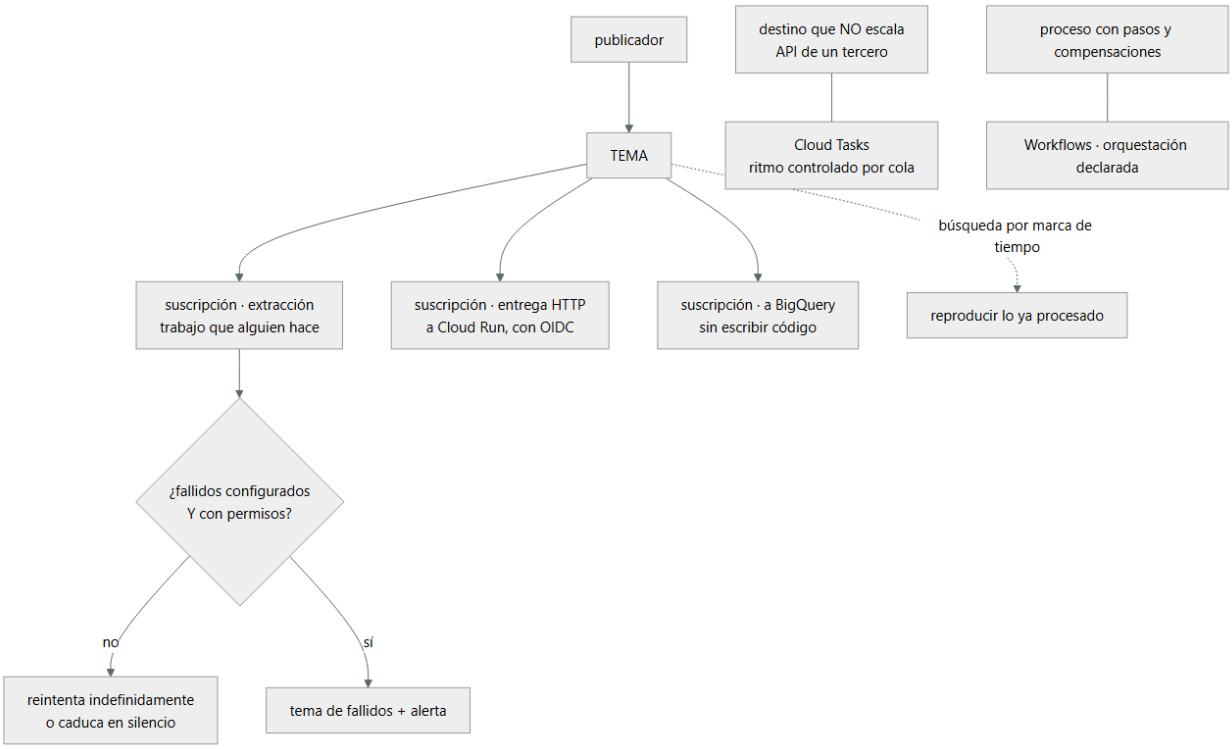
Concepto	Comprensión verificable
tema y suscripción	El tema recibe; cada suscripción tiene su propia copia y su propio estado de confirmación. Añadir un consumidor es añadir una suscripción, no otro producto.
suscripción de extracción o de entrega	La primera la consume el cliente; la segunda hace que Pub/Sub llame por HTTP a un extremo, autenticándose. La misma pieza cubre lo que en la clase 044 eran una cola y un enrutador.
plazo de confirmación	Tiempo reservado para procesar antes de reentregar. Las bibliotecas lo amplían solas si el hilo puede ejecutarse: es la tercera versión del mismo problema del programa.
entrega exactamente una vez	Garantía de que Pub/Sub no reentrega un mensaje ya confirmado dentro de una región. No garantiza que el efecto se aplique una sola vez: si el proceso muere tras aplicar y antes de confirmar, se repite.
límite de ritmo de la cola	Capacidad de Cloud Tasks para entregar como mucho N por segundo. Pub/Sub no tiene equivalente: entrega tan rápido como puede.
búsqueda por marca de tiempo	Reposicionar una suscripción en el pasado para volver a entregar lo ya procesado. Es lo que permite reparar un daño detectado tarde.

Modelo mental

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

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. Un producto, y la decisión se traslada a la suscripción

La clase 044 separaba tres servicios con una pregunta: si el mensaje se consume, Service Bus; si se observa, Event Hubs; si se enruta, Event Grid. Aquí hay un tema y la respuesta está en cuántas suscripciones tiene y de qué tipo:

```
tema "pedidos"
  ■■ suscripción "facturacion"    extracción · su propio estado de confirmación
  ■■ suscripción "almacen"        extracción · su propio estado
  ■■ suscripción "notificar"      entrega HTTP a Cloud Run
  ■■ suscripción "analitica"       escritura directa a BigQuery, sin código
```

Cada suscripción recibe una copia y avanza por su cuenta. Eso significa que «observar» —varios interesados leyendo lo mismo a su ritmo— y «consumir» —trabajo que alguien hace una vez— son la misma pieza usada de dos maneras, y que añadir un consumidor nuevo no obliga a elegir otro producto.

Los tipos de suscripción y cuándo usa cada uno:

Tipo	Quién inicia	Encaja con
Extracción	El consumidor	Trabajo por lotes, control de ritmo propio, consumidores largos
Entrega HTTP	Pub/Sub	Cloud Run y funciones: el consumidor escala con la carga
A BigQuery	Pub/Sub	Analítica sin escribir ni operar un consumidor
A Cloud Storage	Pub/Sub	Archivo en bruto, como la captura de la clase 044

Las dos últimas merecen conocerse porque eliminan código: una suscripción a BigQuery sustituye a un consumidor que solo insertaba filas, con su despliegue, su vigilancia y sus fallos.

Y hay una diferencia que hay que anticipar respecto de la clase 044: una suscripción creada hoy no recibe lo publicado ayer. Es el mismo comportamiento de las suscripciones de un tema de Service Bus, y aquí tiene un remedio que allí no existía —la reproducción, que veremos más abajo— siempre que el tema tenga retención configurada.

```
$ gcloud pubsub topics create pedidos --message-retention-duration 7d
$ gcloud pubsub subscriptions create facturacion --topic pedidos \
    --ack-deadline 60 --retain-acked-messages --message-retention-duration 7d
```

Esa retención tiene costo de almacenamiento y es lo que hace posible reparar un error detectado tres días después. Sin ella, un mensaje confirmado desaparece y la única fuente es el sistema que lo originó.

2. La cola de fallidos que no recibe nada

Este es el fallo de configuración más frecuente de Pub/Sub y es completamente silencioso.

A diferencia de Service Bus, donde la subcola de fallidos existe siempre (clase 044), aquí hay que configurarla:

```
$ gcloud pubsub topics create pedidos-fallidos
$ gcloud pubsub subscriptions update facturacion \
    --dead-letter-topic pedidos-fallidos --max-delivery-attempts 5
```

Y ahí termina la mayoría de las configuraciones, que es donde empieza el problema: Pub/Sub necesita permisos propios para publicar en el tema de fallidos y para confirmar en la suscripción de origen. Sin ellos, no hay error visible; el mensaje simplemente se reintenta indefinidamente hasta caducar.

```
$ AGENTE="service-${NUM_PROYECTO}@gcp-sa-pubsub.iam.gserviceaccount.com"
$ gcloud pubsub topics add-iam-policy-binding pedidos-fallidos \
    --member "serviceAccount:$AGENTE" --role roles/pubsub.publisher
$ gcloud pubsub subscriptions add-iam-policy-binding facturacion \
    --member "serviceAccount:$AGENTE" --role roles/pubsub.subscriber
```

Y la prueba negativa, que es lo único que demuestra que funciona:

```
$ gcloud pubsub topics publish pedidos --message '{"roto": true}'
# el consumidor rechaza el mensaje cinco veces
$ gcloud pubsub subscriptions pull fallidos-sub --limit 1 --auto-ack \
    --format="value(message.data)" | base64 -d
{"roto": true}                                                    ✓
```

Sin esa comprobación, un equipo puede pasar meses creyendo que tiene una red de seguridad que no existe. Y el resto de la disciplina de la clase 044 se aplica igual, empezando por la alerta:

```
mensajes en el tema de fallidos > 0 durante N minutos  →  aviso
edad del mensaje más antiguo sin confirmar > umbral    →  aviso
```

La segunda es la que detecta el sistema que dejó de trabajar sin caerse, que es el fallo silencioso que este programa ha encontrado ya en tres plataformas.

El plazo de confirmación repite el patrón del bloqueo de Service Bus, con los mismos números que importan:

```
por defecto           10 s
máximo por mensaje   600 s
las bibliotecas lo amplían solas mientras el mensaje se procesa
  → SI el hilo puede ejecutarse
```

Y con suscripción de entrega HTTP hay una precisión propia: el plazo de confirmación es el tiempo que Pub/Sub espera la respuesta HTTP. Un manejador en Cloud Run que tarda 90 segundos con un plazo de 60 recibe una reentrega mientras sigue trabajando — el mismo escenario que en la clase 044 cobró dos pedidos duplicados. La corrección es la misma que allí y no ha cambiado en tres plataformas: el manejador registra la intención y devuelve; el trabajo largo ocurre fuera.

Y el control de flujo del cliente, que en las plataformas anteriores no era tan visible:

```
from google.cloud import pubsub_v1

flujo = pubsub_v1.types.FlowControl(max_messages=100, max_bytes=50 * 1024 * 1024)
suscriptor.subscribe(ruta, callback=manejar, flow_control=flujo)
```

Sin acotarlo, un consumidor que se reincorpora tras una caída puede traerse decenas de miles de mensajes a memoria y morir por falta de ella — repitiendo el ciclo indefinidamente. Es un fallo que solo aparece después de un incidente, es decir, en el peor momento.

3. «Exactamente una vez» dicho con precisión

Pub/Sub ofrece entrega exactamente una vez en suscripciones de extracción dentro de una región. Es una capacidad real y conviene decir con exactitud qué cubre, porque la frase invita a concluir de más.

```
$ gcloud pubsub subscriptions create facturacion --topic pedidos \
    --enable-exactly-once-delivery --ack-deadline 60

lo que garantiza
  un mensaje ya confirmado NO se vuelve a entregar
```

```
  no hay reentregas por vencimiento de plazo mal calculado
  la confirmación es transaccional: o vale, o falla y lo sabes

lo que NO garantiza
  que el EFECTO se aplique una sola vez
```

La secuencia que lo demuestra, y que ninguna garantía de transporte puede evitar:

```
1. el consumidor recibe el mensaje
2. aplica el efecto: cobra el pedido
3. el proceso muere antes de confirmar
4. el mensaje NO estaba confirmado → se entrega otra vez
5. otro consumidor vuelve a cobrar
```

Es el mismo razonamiento de las clases 033 y 044, y el hueco está en el paso 3, que está fuera del alcance del sistema de mensajería. La conclusión del programa se mantiene entera y ahora con tres plataformas detrás:

> La entrega exactamente una vez se compra; el efecto exactamente una vez se construye en el manejador.

```
def manejar(mensaje):
    clave = mensaje.attributes["idempotency-key"]
    if repositorio.ya_procesado(clave):
        mensaje.ack(); return
    with repositorio.transaccion():
        aplicar_efecto(mensaje)
        repositorio.marcar_procesado(clave)   # en la MISMA transacción
    mensaje.ack()
```

Lo que sí cambia la garantía es el volumen de trabajo del manejador: sin ella, la idempotencia se ejercita constantemente; con ella, solo en el caso raro del fallo entre aplicar y confirmar. Eso reduce la contención sobre la tabla de claves procesadas, que en volumen alto es una diferencia medible.

Y tiene un coste que hay que conocer: la entrega exactamente una vez reduce el rendimiento máximo de la suscripción y solo funciona en extracción. Para un flujo de telemetría de alto volumen donde un duplicado es irrelevante, activarla es pagar por una garantía que nadie necesita.

Sobre el orden, la tercera aparición del mismo compromiso del programa:

```
$ gcloud pubsub topics publish pedidos --message "$CUERPO" --ordering-key "$PEDIDO_ID"

el orden se garantiza por CLAVE de ordenación, dentro de una región
y la clave de ordenación es también la unidad de SERIALIZACIÓN
```

Exactamente igual que la sesión de Service Bus (clase 044) y que la clave de partición de Cosmos DB y Spanner (clases 042 y 054). Una clave demasiado gruesa —el país, la tienda— convierte el consumo en un solo hilo. La regla es la de siempre: se pide orden donde el negocio lo exige, con la clave más fina que lo satisfaga.

4. Cloud Tasks: cuando el destino no puede con el ritmo

Pub/Sub entrega tan rápido como puede. Es lo correcto cuando el consumidor escala, y es un problema cuando el destino no escala: una API de un tercero con un límite de peticiones, un sistema heredado, un servicio con cuota.

Cloud Tasks existe para eso, y su diferencia es la cola con ritmo:

```
$ gcloud tasks queues create envio-facturas \
    --max-dispatches-per-second 10 --max-concurrent-dispatches 5 \
    --max-attempts 8 --min-backoff 10s --max-backoff 600s

Pub/Sub      abanico de eventos a N interesados, a la velocidad del consumidor
Cloud Tasks  envío controlado a UN destino, al ritmo que ese destino aguanta
```

Y dos capacidades más que Pub/Sub no tiene y que resuelven casos concretos:

Programar una tarea para un instante. No «dentro de un rato» sino a las 09:00 del martes:

```
tarea = {"http_request": {"http_method": "POST", "url": destino, "body": cuerpo},
         "schedule_time": momento,          # instante exacto
         "name": f"{cola}/tasks/recordatorio-{pedido_id}"}
```

Deduplicar por nombre. Dar nombre a la tarea impide crear otra igual durante aproximadamente una hora. Es útil para el caso «si el usuario pulsa cinco veces, se envía un correo», y tiene el mismo límite que la detección de duplicados de la clase 044: es una ventana, no una garantía, y la idempotencia sigue haciendo falta.

La regla de decisión, que evita la discusión:

```
¿varios interesados en el mismo hecho?              Pub/Sub
¿un destino concreto que hay que llamar?            Cloud Tasks
¿hay que controlar el ritmo o programar el momento? Cloud Tasks
¿hace falta reproducir el histórico?                Pub/Sub
```

Y Workflows cubre el tercer caso, el que la clase 032 planteó como orquestación frente a coreografía: un proceso con pasos, condiciones, compensaciones y espera.

```
main:
  params: [pedido]
  steps:
    - reservar_stock:
        call: http.post
        args: {url: "${ALMACEN}/reservar", body: "${pedido}"}
        result: reserva
        retry: {predicate: "${http.default_retry_predicate}", max_retries: 5,
                backoff: {initial_delay: 1, max_delay: 60, multiplier: 2}}
    - cobrar:
        try:
          call: http.post
          args: {url: "${PAGOS}/cobrar", body: "${pedido}"}
        except:
          as: e
          steps:
            - compensar:
                call: http.post
                args: {url: "${ALMACEN}/liberar", body: "${reserva}"}
            - propagar:
                raise: "${e}"
    - confirmar:
        return: "${pedido.id}"
```

Lo que aporta frente a encadenar mensajes es que la compensación está escrita en un sitio y se puede leer. En una coreografía por eventos, la respuesta a «qué pasa si el cobro falla después de reservar» está repartida entre tres servicios y nadie la tiene entera.

Su precio es por paso ejecutado —del orden de céntimos por millar—, así que es barato para procesos de negocio y caro para bucles con miles de iteraciones. Y su límite es el mismo de toda orquestación: introduce un componente que conoce el proceso completo, lo que hay que asumir a cambio de poder leerlo.

5. Reproducir: la capacidad que repara un daño detectado tarde

Esta es la propiedad que en la clase 044 no existía y que conviene tener presente antes de necesitarla.

Una suscripción puede reposicionarse en el pasado:

```
# volver a entregar todo lo publicado desde un instante
$ gcloud pubsub subscriptions seek facturacion \
    --time 2026-07-28T02:00:00Z

# o saltar hacia adelante para descartar lo pendiente
$ gcloud pubsub subscriptions seek facturacion --time $(date -u +%Y-%m-%dT%H:%M:%SZ)
```

Y con instantáneas, se puede guardar el estado de confirmación de una suscripción antes de un despliegue arriesgado y volver a él:

```
$ gcloud pubsub snapshots create antes-de-v8 --subscription facturacion
# … despliegue, y si sale mal:
$ gcloud pubsub subscriptions seek facturacion --snapshot antes-de-v8
```

Eso convierte un fallo lógico en un incidente reparable. El caso típico: un cambio introduce un error de cálculo que corrompe datos durante dos días, y se detecta al tercero. Sin reproducción, la reparación es un guion a medida que lee del sistema de origen. Con reproducción, es corregir el consumidor y reposicionar la suscripción.

Con dos condiciones que hay que haber cumplido antes:

1. el tema y la suscripción deben tener retención suficiente
   la ventana de reparación es exactamente la retención configurada
2. el consumidor debe ser IDEMPOTENTE
   reproducir entrega mensajes que YA se procesaron

La segunda es la que convierte la capacidad en utilizable. Un consumidor no idempotente enfrentado a una reproducción duplica todos los efectos del intervalo, así que la herramienta de reparación se convierte en la causa de

un daño mayor. Es el mismo argumento de idempotencia por cuarta vez en el programa, ahora con una razón nueva: no solo por los duplicados que ocurren, sino por los que tú vas a provocar a propósito el día que necesites reparar algo.

Y el resto del ecosistema encaja alrededor. Eventarc unifica el enrutamiento de eventos de la plataforma —un blob nuevo en un bucket de la clase 053, un cambio de configuración auditado de la clase 049— hacia Cloud Run o hacia Workflows:

```
$ gcloud eventarc triggers create indexar-factura \
    --destination-run-service svc-indexador --destination-run-region europe-west1 \
    --event-filters="type=google.cloud.storage.object.v1.finalized" \
    --event-filters="bucket=cls-facturas" \
    --service-account sa-eventarc@cls-tienda-prod-euw1-01.iam.gserviceaccount.com
```

Por debajo eso crea un tema y una suscripción de entrega, así que todo lo de esta clase se aplica igual: plazo de confirmación, fallidos con permisos, idempotencia y alertas. Saberlo evita tratarlo como una caja distinta cuando algo va mal.

Ejemplo trabajado

CloudShop monta su columna asíncrona en Google Cloud con el contrato de la clase 044 en la mano. Dos problemas conocidos se evitan de entrada, uno reaparece con otra cara, y aparecen dos propios de la plataforma — uno de ellos completamente silencioso.

Lo que se evitó por venir escrito.

```
manejadores idempotentes desde el primer día     (044)
alerta sobre mensajes fallidos con umbral cero   (044)
el manejador registra la intención y devuelve;
  el trabajo largo ocurre fuera                  (044)
clave de ordenación fina: pedidoId, no pais      (042, 044, 054)
```

Cuatro decisiones tomadas antes de desplegar, cada una por un incidente pagado en otra plataforma.

Problema 1 — la cola de fallidos llevaba tres semanas sin recibir nada.

La alerta de mensajes fallidos nunca se disparó. El equipo lo interpretó como buena señal, hasta que una revisión encontró mensajes con más de 400 intentos de entrega:

```
$ gcloud pubsub subscriptions pull facturacion --limit 1 --format="value(deliveryAttempt)"
412
```

Cuatrocientos doce intentos con --max-delivery-attempts 5 configurado. La causa: el agente de servicio de Pub/Sub no tenía permiso para publicar en el tema de fallidos, así que el traslado fallaba y el mensaje volvía a la cola.

```
permisos del agente de servicio              ninguno      publisher + subscriber
mensajes con más de 5 intentos                1.841              0
mensajes en el tema de fallidos                  0               1.841 trasladados
prueba negativa de la cola de fallidos         no          sí, ejecutada
```

El equipo añade la comprobación a su lista de plataforma nueva, en la forma que ya tienen para otras: una cola de fallidos sin una prueba que demuestre que recibe es una cola de fallidos que no existe.

Problema 2 — el proveedor de envíos bloquea a CloudShop por exceso de peticiones.

El servicio de notificación de envíos usaba una suscripción de entrega HTTP hacia un servicio que llamaba a la API del transportista. En una campaña, Pub/Sub entregó todo lo acumulado de golpe:

```
peticiones al transportista en 60 s   4.312
límite contratado                       600 por minuto
resultado                             bloqueo de la cuenta durante 2 h
```

Pub/Sub no tiene control de ritmo: entrega tan rápido como el consumidor acepta, y el consumidor escalaba. Se cambia la pieza:

```
mecanismo                       Pub/Sub → Cloud Run     Pub/Sub → Cloud Run
                                → API del transportista  → Cloud Tasks
                                                         → API del transportista
ritmo hacia el transportista      sin control        10 por segundo, 5 simultáneas
reintentos                      del consumidor      de la cola, con retroceso
bloqueos del proveedor               1                    0
```

La lección se generaliza: el ritmo lo tiene que controlar quien conoce el límite del destino, y ese no es el sistema de mensajería.

Problema 3 — el plazo de confirmación, por tercera vez.

El consumidor de facturación tardaba entre 40 y 95 segundos, con un plazo de 60.

```
mensajes reentregados mientras seguían procesándose   214 en 24 h
facturas duplicadas                                     0
```

Cero facturas duplicadas, porque los manejadores eran idempotentes desde el primer día. El daño fue trabajo desperdiciado, no datos incorrectos — que es exactamente la diferencia que la idempotencia compra.

```
plazo de confirmación                60 s              600 s
trabajo dentro del manejador     generar la factura   registrar y devolver
generación de la factura             —              trabajo de Cloud Run (055)
reentregas mientras se procesa       214                 0
```

Problema 4 — dos días de comisiones mal calculadas.

Un cambio en el cálculo de comisiones introdujo un error que se detectó al tercer día: 61.000 pedidos con la comisión mal.

En la plataforma anterior esto habría exigido un guion de reparación leyendo del sistema de pedidos. Aquí:

```
$ gcloud pubsub snapshots create antes-de-reparar --subscription comisiones
$ gcloud pubsub subscriptions seek comisiones --time 2026-07-26T00:00:00Z
```

```
mensajes reentregados                61.412
duración del reproceso                38 min
efectos duplicados                       0    ← por la idempotencia
comisiones corregidas                61.412
reparación manual necesaria           ninguna
```

La fila de los efectos duplicados es la que hace que esto funcione. La idempotencia, que hasta ahora se justificaba por los duplicados que ocurren, aquí se justifica por los duplicados que se provocan a propósito. Sin ella, la herramienta de reparación habría multiplicado el daño por dos.

Y una simplificación que quitó código.

El consumidor de analítica solo insertaba filas en el almacén de datos:

```
consumidor de analítica       servicio propio en Cloud Run   suscripción a BigQuery
líneas de código                     310                        0
despliegues que mantener              1                         0
fallos de ese consumidor en 3 meses   4                         —
costo mensual                    12 USD                      2 USD
```

Resumen de la columna asíncrona:

```
productos usados                            1 (Pub/Sub)     3, por criterio
mensajes atascados sin llegar a fallidos      1.841             0
bloqueos del proveedor externo                    1             0
reentregas por plazo corto                      214             0
consumidores escritos a mano                      4             3
reparación de un daño de 2 días           no posible    38 min, sin duplicados
costo mensual de mensajería                 47 USD          29 USD
```

La lección que esta clase traslada al resto de la parte 04: el contrato de la clase 044 se reutilizó entero y evitó cuatro incidentes antes de ocurrir, y los dos problemas nuevos fueron de la misma familia que los de las plataformas anteriores — una red de seguridad que parecía existir y no existía, y un destino que no aguantaba el ritmo de quien le enviaba. Lo genuinamente nuevo fue la capacidad de reproducir, y su valor dependió por completo de una propiedad que ya estaba escrita en el contrato: sin manejadores idempotentes, reproducir habría sido peor que no poder hacerlo.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-04-gcp-core-platform/056-pub-sub-cloud-tasks-y-workflows/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-gcp 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-gcp 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 cola de mensajes fallidos nunca recibe nada y hay mensajes con cientos de intentos	El agente de servicio de Pub/Sub no tiene permiso para publicar en el tema de fallidos ni para confirmar en el origen	Concede publisher y subscriber al agente y demuéstralo publicando un mensaje que el consumidor rechace.
Un proveedor externo bloquea la cuenta por exceso de peticiones	Pub/Sub entrega tan rápido como el consumidor acepta y el destino tiene un límite	Interpón Cloud Tasks con max-dispatches-per-second acorde al límite del destino.
Mensajes reentregados mientras el manejador sigue trabajando	El plazo de confirmación es menor que el tiempo de proceso; en entrega HTTP, ese plazo es el tiempo de espera de la respuesta	Registra la intención y devuelve; ejecuta el trabajo largo en un trabajo aparte y amplía el plazo si hace falta.
Se activa la entrega exactamente una vez y sigue habiendo efectos duplicados	La garantía es de entrega, no de aplicación: un fallo entre aplicar y confirmar reentrega	Idempotencia en el manejador, marcando la clave en la misma transacción que el efecto.
Un consumidor muere por falta de memoria al reincorporarse tras una caída	Sin control de flujo, se trae a memoria todo el acumulado	Configura maxmessages y maxbytes en el cliente antes de que ocurra el primer incidente.
No se puede reparar un daño detectado tres días después	La retención del tema y de la suscripción era menor que ese plazo	Configura la retención según la ventana de reparación que quieras tener, y verifica que los consumidores son idempotentes antes de reproducir.

■ 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. ¿Cómo se resuelve con un solo producto la distinción entre consumir y observar de la clase 044?
2. ¿Qué tres cosas hacen falta para que una cola de mensajes fallidos reciba de verdad, y cómo lo demuestras?
3. ¿Qué garantiza exactamente la entrega exactamente una vez y qué secuencia sigue produciendo efectos duplicados?
4. ¿En qué caso concreto Cloud Tasks es la pieza correcta y Pub/Sub no puede sustituirlo?
5. ¿Qué dos condiciones hay que haber cumplido antes para poder reparar un daño reproduciendo mensajes?

Referencias

• Google Cloud (2025). Pub/Sub subscription types — extracción, entrega HTTP, BigQuery y Cloud Storage. <https://cloud.google.com/pubsub/docs/subscriber>

• Google Cloud (2025). Handling message failures — cola de fallidos, permisos del agente de servicio y reintentos. <https://cloud.google.com/pubsub/docs/handling-failures>

• Google Cloud (2025). Exactly-once delivery — alcance de la garantía y sus límites. <https://cloud.google.com/pubsub/docs/exactly-once-delivery>

• Google Cloud (2025). Replay and purge messages — búsqueda por marca de tiempo e instantáneas. <https://cloud.google.com/pubsub/docs/replay-overview>

• Google Cloud (2025). Cloud Tasks overview — control de ritmo, programación y deduplicación por nombre. <https://cloud.google.com/tasks/docs/dual-overview>

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

Descargar: Parte 04 en PDF · Recorrido de Google Cloud en PDF · Manual integral

Anterior	Índice	Siguiente
← 055 · Cloud Run, Cloud Functions y API Gateway	Parte 04 · Programa	057 · Cloud Logging, Monitoring, Trace y Audit Logs →

Evaluación — 056 Pub/Sub, Cloud Tasks y Workflows

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-gcp 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.

057 — Cloud Logging, Monitoring, Trace y Audit Logs

Parte: 04 — Google Cloud: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: observability · Estado: EXECUTABLECORE

Propósito

Construir la evidencia operativa de una plataforma de Google Cloud, donde el problema es el contrario al de la clase 045: aquí casi todo se registra desde el primer minuto, así que un incidente nunca se queda sin datos y la factura llega antes que la pregunta. Y donde existe la pieza que faltaba en las dos plataformas anteriores: alertar sobre el consumo del presupuesto de error en lugar de sobre umbrales de recursos, que es la diferencia entre 340 avisos al mes y once que importan.

Resultados de aprendizaje

Al finalizar podrás:

1. Distinguir qué registros están activos y son gratuitos de los que hay que habilitar y se pagan.
2. Reducir la factura de ingesta con filtros de exclusión y destinos, sin perder capacidad de investigación.
3. Convertir un patrón de registro en una métrica para alertar en un minuto en vez de en quince.
4. Definir un SLI y un SLO, y alertar por velocidad de consumo del presupuesto de error.
5. Reconstruir una petición completa entre servicios y localizar el salto que corta la traza.

Conceptos centrales

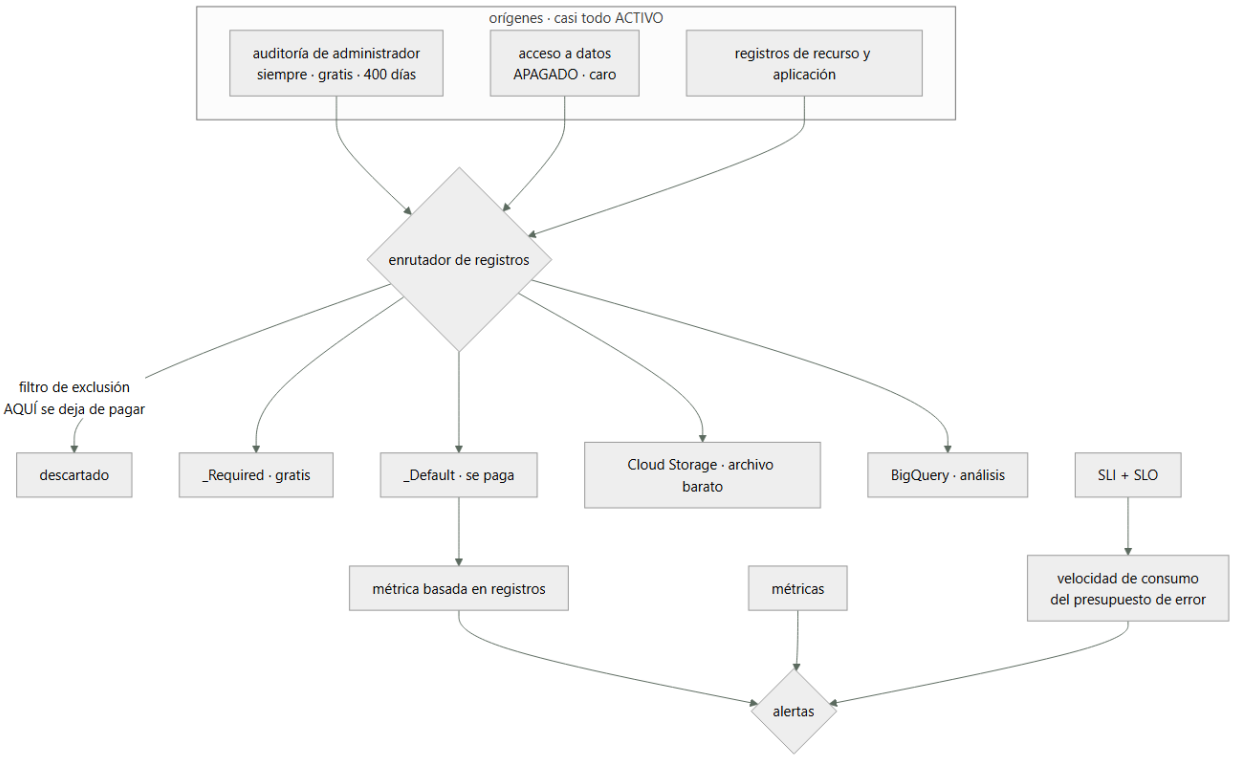
Concepto	Comprensión verificable
registro de auditoría de actividad de administrador	Quién creó, modificó o borró qué. Está siempre activo, es gratuito y se conserva 400 días sin configurar nada — justo lo que en la clase 045 exigía exportar.
registro de acceso a datos	Quién leyó qué dato. Está apagado por defecto —salvo excepciones— y es el más voluminoso: habilitarlo en todo es la forma más rápida de multiplicar la factura.
enrutador de registros	Punto por el que pasa todo antes de almacenarse. Es donde se excluye antes de pagar y donde se decide el destino: bucket de registros, almacenamiento, BigQuery o Pub/Sub.
métrica basada en registros	Contador o distribución extraído de un patrón de texto. Convierte una señal que solo existe en los registros en una métrica sobre la que se alerta en un minuto.
presupuesto de error	Fracción de peticiones que el SLO permite fallar en un periodo. Alertar sobre su velocidad de consumo sustituye a los umbrales de CPU y elimina la mayoría de los avisos inútiles.
perfilador continuo	Muestreo de CPU y memoria en producción con sobrecarga mínima. Responde a «dónde se va el tiempo» sin reproducir nada en un entorno de pruebas.

Modelo mental

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

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. Aquí el problema no es la falta de datos, es la factura

La clase 045 empezaba con una frase que aquí no vale: «nada se registra hasta que alguien lo enciende». En Google Cloud el reparto es el contrario, y conviene conocerlo con precisión porque decide dónde está el riesgo.

```
siempre activo y GRATUITO
  auditoría de actividad de administrador   quién creó, modificó o borró
  auditoría de eventos del sistema          acciones de la propia plataforma
  → se guardan 400 días en el bucket _Required, que no se puede modificar

activo y facturado
  registros de aplicación y de la mayoría de servicios
  registros de peticiones de los balanceadores y de Cloud Run
  auditoría de política denegada

APAGADO por defecto
  auditoría de ACCESO A DATOS: quién leyó qué
```

La primera línea resuelve de fábrica el problema que la clase 045 tuvo que resolver exportando a almacenamiento inmutable: quién borró un recurso hace cuatro meses tiene respuesta, sin haber configurado nada, y sin que nadie pueda borrar esa respuesta.

```
$ gcloud logging read 'protoPayload.methodName:"delete" AND
  timestamp>="2026-04-01T00:00:00Z"' --limit 20 \
  --format="table(timestamp, protoPayload.authenticationInfo.principalEmail,
                  protoPayload.methodName, protoPayload.resourceName)"
```

La tercera línea es la asimetría propia, y es una decisión con precio. Los registros de acceso a datos responden a «quién leyó la tabla de clientes», que es la pregunta de cualquier investigación sobre datos personales — y son, con diferencia, los más voluminosos, porque hay muchas más lecturas que cambios de configuración.

```
$ gcloud projects get-iam-policy $PROYECTO --format=json > politica.json
# habilitar SOLO donde importa, no en toda la organización
```

El criterio que evita la factura sin renunciar a la evidencia: habilitarlos por servicio y por proyecto, en los que contienen datos personales o financieros, y dejarlos apagados en el resto. Activarlos en la organización entera «por si acaso» es

la forma más rápida de multiplicar el gasto de telemetría por varias veces.

Y el modelo de precios, que cambia lo que es razonable guardar:

```
ingesta          ~0,50 USD por GiB
retención        30 días incluidos en _Default; más allá, por GiB y mes
_Required        gratis, 400 días, no configurable
métricas de plataforma  gratuitas
```

Compararlo con la clase 045 explica por qué el diseño cambia: allí la ingesta costaba unas cuatro veces más, así que la palanca era el plan por tabla. Aquí la ingesta es más barata y el volumen es mayor porque todo está encendido, de modo que la palanca principal es otra: descartar antes de ingerir.

2. El enrutador: excluir antes de pagar, archivar lo que solo se guarda

Todo registro pasa por el enrutador antes de almacenarse, y ahí hay dos operaciones distintas que se confunden:

```
destino (sink)     copia los registros que coinciden hacia un destino
                   → se paga la ingesta en el destino que corresponda
exclusión          DESCARTA antes de almacenar
                   → no se paga
```

La exclusión es la que baja la factura. Y se escribe con el mismo lenguaje de filtro que las consultas:

```
$ gcloud logging sinks update _Default \
    --add-exclusion=name=peticiones-correctas,\
filter='resource.type="cloud_run_revision" AND httpRequest.status<400'

$ gcloud logging sinks update _Default \
    --add-exclusion=name=flujos-vpc-muestreo,\
filter='resource.type="gce_subnetwork" AND log_id("compute.googleapis.com/vpc_flows")
        AND sample(insertId, 0.05)'
```

La segunda usa muestreo: conserva el 5 % de los registros de flujo de la clase 051 y descarta el resto. Para detectar patrones de tráfico, una muestra basta; para una investigación forense concreta, no — y ahí es donde entra el archivo barato:

```
$ gcloud logging sinks create archivo-crudo \
    storage.googleapis.com/projects/$P/buckets/cls-registros-crudos \
    --log-filter='resource.type="gce_subnetwork" OR resource.type="http_load_balancer"'
```

Ese destino escribe a Cloud Storage, donde el GB cuesta una fracción y las reglas de ciclo de vida de la clase 053 lo retiran solas. La combinación resuelve las dos necesidades sin pagar dos veces:

```
lo que se consulta a diario   en el bucket de registros, con retención corta
lo que se consulta una vez    en Cloud Storage, barato, con ciclo de vida
lo que no se consulta nunca   excluido: no se paga
```

La tercera línea exige una comprobación honesta que casi nadie hace: mirar qué se ha consultado de verdad en los últimos noventa días. Un tipo de registro que nadie ha buscado nunca es candidato a exclusión o a archivo, y esa revisión es la que convierte la reducción de costo en una decisión con datos en vez de en un recorte a ciegas.

Y para el análisis, Log Analytics convierte un bucket de registros en algo consultable con SQL, lo que evita mover los datos a otro sitio solo para poder agregarlos:

```
SELECT
  JSON_VALUE(resource.labels.service_name) AS servicio,
  COUNT(*) AS peticiones,
  COUNTIF(CAST(JSON_VALUE(http_request.status) AS INT64) >= 500) AS errores
FROM `cls-obs.global._Default._AllLogs`
WHERE timestamp > TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 1 HOUR)
GROUP BY servicio ORDER BY errores DESC
```

Y un límite del enrutador que produce una sorpresa: las exclusiones se aplican al bucket de destino, no al origen. Un registro excluido de Default sigue llegando a otros destinos que lo seleccionen. Es lo que se quiere —excluir del caro y archivar en el barato— y hay que tenerlo presente al calcular el ahorro.

3. Alertar por presupuesto de error, no por umbral de recurso

Esta es la pieza que las clases 034 y 045 no tenían y que cambia la operación más que ninguna otra de la parte.

El problema con las alertas de umbral es conocido: la CPU al 85 % puede ser perfectamente normal, y el 40 % puede ser catastrófico si la mitad de las peticiones fallan. Un panel lleno de avisos que nadie atiende no es vigilancia, es ruido

con guardia.

Google Cloud modela el objetivo directamente:

```
SLI   indicador: proporción de peticiones buenas sobre el total
SLO   objetivo: 99,9 % en 28 días
presupuesto de error   0,1 % de las peticiones
  con 10 M de peticiones al mes → 10.000 peticiones pueden fallar
```

Y la alerta se define sobre la velocidad a la que se consume ese presupuesto:

```
velocidad de consumo = tasa de error observada / tasa que agota justo el presupuesto

velocidad 1    a este ritmo, el presupuesto se agota exactamente al final del periodo
velocidad 14,4 el presupuesto de 28 días se agota en 2 días
               → si dura una hora, ya se ha gastado el 2 % del mes
```

La configuración estándar usa dos ventanas, y su lógica merece entenderse porque es lo que elimina el ruido:

```
ventana corta + velocidad alta   (5 min, ×14,4)    incidente agudo → avisar YA
ventana larga + velocidad baja   (6 h, ×1)         degradación lenta → avisar,
                                                     pero sin urgencia
y se exige que AMBAS ventanas coincidan
  → un pico de 30 segundos no dispara nada
  → una degradación sostenida sí
```

```
$ gcloud alpha monitoring policies create --policy-from-file - <<'YAML'
displayName: "Consumo rápido del presupuesto de error · tienda"
conditions:
  - displayName: "velocidad de consumo > 14,4 en 5 min y 1 h"
    conditionThreshold:
      filter: 'select_slo_burn_rate("projects/…/services/tienda/serviceLevelObjectives/disponibilidad-
999", "3600s")'
      comparison: COMPARISON_GT
      thresholdValue: 14.4
      duration: 300s
YAML
```

Lo que se gana no es solo menos ruido. Se gana que cada alerta corresponde a algo que el usuario nota, y que la urgencia está calibrada: una alerta de velocidad 14,4 significa que si nadie hace nada en dos días no queda presupuesto, lo que es una frase que un responsable de guardia puede entender a las tres de la madrugada. «CPU al 85 %» no lo es.

Y las alertas de umbral no desaparecen: se quedan para lo que sí es un límite duro, que es distinto de una degradación:

```
SLO / presupuesto de error   la experiencia del usuario
umbral                        cuotas, disco lleno, certificado que caduca,
                              conexiones al máximo, mensajes fallidos > 0
```

La segunda columna incluye la alerta que este programa ha justificado en tres plataformas: trabajo que se acumula. Un consumidor detenido no consume presupuesto de error de un servicio HTTP, así que un sistema que solo vigile el SLO de la API no lo detecta.

Y para lo que solo existe en los registros, la métrica basada en registros cierra el hueco que la clase 045 dejó abierto —allí, alertar sobre un registro costaba entre siete y veinte minutos—:

```
$ gcloud logging metrics create errores-cobro \
    --description "Excepciones del servicio de cobro" \
    --log-filter='resource.type="cloud_run_revision"
      AND resource.labels.service_name="svc-pagos"
      AND severity>=ERROR AND jsonPayload.tipo="CobroRechazado"'
```

Una vez extraída, es una métrica normal: se alerta sobre ella en aproximadamente un minuto, con el mismo mecanismo rápido que cualquier otra.

4. Trazas, errores y dónde se va el tiempo

Tres herramientas responden tres preguntas distintas, y las tres están integradas sin montar nada.

Cloud Trace reconstruye la petición completa entre servicios. La correlación viaja en la cabecera estándar del W3C —o en la propia de Google— y tiene la misma condición que en la clase 045: cada salto debe propagarla.

```
X-Cloud-Trace-Context: TRACE_ID/SPAN_ID;o=1
```

```
traceparent: 00-<trace-id>-<span-id>-01
```

Y el mismo síntoma cuando falta: la traza se corta justo en el servicio interno que se escribió a mano, que es normalmente el interesante. La comprobación es directa —buscar una traza y contar los tramos esperados— y conviene hacerla el día que se despliega un servicio nuevo, no durante un incidente.

En Cloud Run hay un detalle que ahorra trabajo: la plataforma ya emite un tramo por petición, así que el mapa básico existe sin instrumentar nada. Lo que hay que añadir es la propagación entre servicios propios y los tramos de las llamadas a bases de datos, que es donde suele estar el tiempo.

Error Reporting agrupa las excepciones por firma y cuenta ocurrencias, primera y última vez. Sustituye a buscar cadenas en los registros y responde a la pregunta que abre casi todo incidente de aplicación: qué ha empezado a fallar ahora que antes no fallaba. Funciona sin configuración si las excepciones se escriben con la severidad y el formato adecuados, lo que es un argumento concreto para registrar en estructura y no en texto libre:

```
import json, sys
print(json.dumps({
    "severity": "ERROR",
    "message": str(exc),
    "stack_trace": traceback.format_exc(),
    "logging.googleapis.com/trace": f"projects/{proyecto}/traces/{trace_id}",
}), file=sys.stderr)
```

Ese campo de traza es el que enlaza el error con su petición completa: desde la excepción se llega a la traza, y desde la traza a la dependencia que la causó. Es la cadena que la clase 034 pedía y que aquí cuesta una línea.

Cloud Profiler responde a una pregunta que ninguna de las plataformas anteriores contestaba en producción: dónde se va el tiempo de CPU y la memoria. Muestrea continuamente con una sobrecarga muy baja y presenta el resultado como gráfico de llamas.

```
import googlecloudprofiler
googlecloudprofiler.start(service="svc-tienda", service_version="v8")
```

Su valor es que encuentra cosas que nadie buscaría: un serializador que consume un tercio de la CPU, una expresión regular recompilada en cada petición, una biblioteca de fechas que domina el perfil. Son mejoras que no aparecen en ninguna traza porque están dentro de un tramo, y que suelen tener una corrección de pocas líneas.

Y el Servicio Gestionado para Prometheus merece mención porque decide el diseño de la parte 06: Google Cloud ingiere métricas en formato Prometheus de forma nativa, con consultas en PromQL. Eso significa que la instrumentación de una aplicación no es específica del proveedor: las mismas métricas y los mismos paneles valen aquí, en un clúster propio o en otra nube. Es exactamente el tipo de contrato portable que la clase 048 pedía identificar, y aparece en la capa donde más veces se pierde.

5. El método, ya idéntico en tres plataformas

Con tres proveedores recorridos, el procedimiento de investigación ya no es una receta por producto. Es el mismo, y solo cambia el nombre del comando:

```
1. ¿el usuario lo nota?        SLO y presupuesto de error, no CPU
2. ¿desde cuándo?              percentil de latencia y tasa de error en serie
3. ¿qué cambió en la ventana?  auditoría de administrador + despliegues
4. ¿qué traza lo demuestra?    una petición fallida, de extremo a extremo
5. ¿qué dependencia?           tramos de la traza, o errores agrupados
6. ¿quién y con qué permiso?   auditoría, y el sistema de identidad (050)
```

Y las consultas de cada paso se escriben una vez y se guardan, por la razón que la clase 045 ya dio: un incidente no es el momento de aprender a consultar.

```
# 3. qué cambió en las últimas 6 horas
$ gcloud logging read 'logName:"cloudaudit.googleapis.com%2Factivity"
  AND timestamp>="'$(date -u -d "6 hours ago" +%Y-%m-%dT%H:%M:%SZ)'"' \
  --format="table(timestamp, protoPayload.authenticationInfo.principalEmail,
                  protoPayload.methodName, resource.labels.service_name)"
```

Y la comparación de las tres plataformas en observabilidad, que es lo que se lleva a la clase 060:

```
registros de recurso              parcial       APAGADOS       activos
auditoría, retención gratuita     90 días       90 días        400 días
precio de ingesta                 medio         ~2,30 USD/GiB  ~0,50 USD/GiB
registros de acceso a datos       opcionales    opcionales     apagados, caros
alerta rápida sobre registros     métrica       ~9 min         métrica de registro
```

```
SLO y presupuesto de error        se construye  se construye   INTEGRADO
perfilado en producción           aparte        aparte         INTEGRADO
```

Dos filas van claramente a favor de esta plataforma y una en contra —los registros de acceso a datos apagados, que es la versión local del error de la clase 045—. Y la fila del SLO es la que más cambia la operación diaria, porque no es una herramienta más: es la que decide qué despierta a alguien.

Un cierre que conviene dejar escrito para el proyecto de la clase 060: la observabilidad no se juzga por cuántos paneles hay, sino por dos números que se pueden medir en un simulacro —cuánto tarda en detectarse un fallo y cuánto en localizar su causa—. Todo lo de esta clase existe para bajar esos dos números, y cualquier pieza que no lo haga es decoración que cuesta dinero al mes.

Ejemplo trabajado

CloudShop instrumenta su plataforma de Google Cloud. El incidente que abrió la clase 045 —una caída sin datos que analizar— no se reproduce: los registros estaban. Los cinco problemas de este mes son de otra naturaleza.

Lo que no se reprodujo.

```
en Azure    48 minutos de caída sin ningún registro del plano de datos,
            porque las configuraciones de diagnóstico estaban apagadas
aquí        los registros de la petición, del balanceador y de la auditoría
            estaban desde el primer minuto, sin configurar nada
tiempo hasta identificar la causa raíz    23 minutos
```

Problema 1 — la factura de telemetría, en el día doce.

```
SELECT log_name, ROUND(SUM(bytes)/POW(1024,3), 1) AS gib
FROM `cls-obs.global._Default._AllLogs` …
```

```
flujos de VPC (clase 051)                    31 GiB/día   50 %
peticiones de Cloud Run, todas               18 GiB/día   29 %
balanceador, incluidas las 2xx                9 GiB/día   15 %
resto                                         4 GiB/día    6 %
                                            ■■■■■■■■■■
                                              62 GiB/día  → ~930 USD/mes
```

El 94 % del gasto eran registros que nadie había consultado nunca. La comprobación se hizo mirando el historial de consultas, no por intuición.

```
flujos de VPC                                 31 GiB/día   1,6 GiB/día (muestreo 5 %)
  el volumen íntegro va a Cloud Storage con ciclo de vida a 30 días
peticiones de Cloud Run                       18 GiB/día   3,1 GiB/día (solo >= 400)
balanceador                                    9 GiB/día   1,4 GiB/día (solo >= 400)
resto                                          4 GiB/día     4 GiB/día
                                             ■■■■■■■■■■    ■■■■■■■■■■
ingesta                                       62 GiB/día  10,1 GiB/día
costo mensual de ingesta                      ~930 USD       ~152 USD
archivo en Cloud Storage                           —          ~19 USD
```

Capacidad de investigación perdida: ninguna. Lo excluido sigue existiendo, más barato y más lento de consultar.

Problema 2 — «¿quién leyó la tabla de clientes?» no tenía respuesta.

Una consulta de cumplimiento pregunta por accesos a datos personales de los tres meses anteriores.

```
$ gcloud logging read 'protoPayload.methodName="google.cloud.bigquery.v2.JobService.InsertJob"
  AND protoPayload.resourceName:"clientes"' --limit 5
# (vacío para los servicios distintos de BigQuery)
```

Los registros de acceso a datos estaban apagados, que es su valor por defecto. Se habilitan de forma selectiva, con la aritmética delante:

```
volumen añadido estimado                  +48 GiB/día        +2,3 GiB/día
costo añadido                             +720 USD/mes        +35 USD/mes
cobertura                                 todo                los 4 proyectos
                                                              con datos personales
```

Treinta y cinco dólares al mes por poder responder la pregunta que importa. Setecientos veinte por poder responder también las que nadie hace.

Problema 3 — 340 alertas al mes y ninguna correspondía a un usuario afectado.

```
alertas del mes anterior            340
```

```
de ellas, con impacto en usuarios     6
de ellas, atendidas                  31
de ellas, silenciadas                46
```

El resto se ignoraron. Se sustituyen los umbrales de recurso por objetivos de servicio:

```
SLI      peticiones con estado < 500 y latencia < 800 ms / total
SLO      99,9 % en 28 días
presupuesto de error   0,1 % → ~10.200 peticiones al mes
alertas  velocidad ×14,4 en ventanas de 5 min y 1 h   → aviso inmediato
         velocidad ×1 en ventanas de 6 h y 3 días     → aviso sin urgencia
```

```
alertas al mes                          340            11
con impacto real en usuarios              6            11
falsos positivos                        334             0
alertas silenciadas                      46             0
tiempo medio hasta la detección       19 min         3 min 40 s
```

Once alertas, once incidentes reales. Y se conservan seis alertas de umbral para lo que sí es un límite duro: cuota al 80 %, disco al 70 %, certificado a 30 días, conexiones al 85 %, mensajes fallidos mayor que cero y acumulación en cola — la última, por tercera vez en el programa.

Problema 4 — la traza se cortaba en el servicio de precios.

```
tramos esperados en una petición de pedido    6
tramos observados                             3
el corte, siempre en la llamada a svc-precios
```

El cliente HTTP de ese servicio se había escrito a mano y no reenviaba la cabecera de traza. Tercera vez que este fallo aparece en el programa, con la misma corrección.

```
propagación de la cabecera        4 de 6 saltos   6 de 6
tiempo hasta localizar una dependencia lenta   ~25 min   ~3 min
comprobación de continuidad de traza  ninguna   en el despliegue de cada servicio
```

Problema 5 — el 38 % de la CPU estaba en un sitio que nadie habría mirado.

Con el perfilador activo una semana:

```
serialización a JSON del catálogo completo    38 % del tiempo de CPU
  causa: se serializaba el objeto entero para devolver 4 campos
```

```
CPU en serialización                  38 %          6 %
p95 del listado de catálogo         214 ms        131 ms
instancias en el pico                    9             5
costo mensual de Cloud Run           31 USD        19 USD
```

Una corrección de once líneas, encontrada porque el perfilador estaba encendido y no porque nadie sospechara.

Resumen de la observabilidad:

```
ingesta diaria                              62 GiB/día    10,1 GiB/día
costo mensual de telemetría                  930 USD        206 USD
registros de acceso a datos                apagados     activos en 4 proyectos
alertas al mes                                340             11
falsos positivos                              334              0
tiempo medio hasta la detección              19 min         3 min 40 s
propagación de traza                       4 de 6 saltos   6 de 6
```

La lección que esta clase traslada al proyecto de la clase 060: el problema de la observabilidad cambia de forma según la plataforma —en la parte 03 era la ausencia de datos, aquí es su exceso— y no cambia el criterio. En ambos casos la pregunta correcta es la misma: ¿qué señal necesito para responder a un incidente, y qué estoy pagando que nunca he consultado? La respuesta se obtiene mirando el historial de consultas, y casi siempre sorprende.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-04-gcp-core-platform/057-cloud-logging-monitoring-trace-y-audit-logs/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-gcp 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-gcp 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 factura de telemetría es varias veces la prevista desde el primer mes	Casi todo se registra por defecto, incluidos los flujos de red y las peticiones correctas	Excluye en el enrutador antes de ingerir, muestrea lo voluminoso y archiva el volumen íntegro en Cloud Storage con ciclo de vida.
No se puede saber quién leyó un dato personal	Los registros de auditoría de acceso a datos están apagados por defecto	Habilítalos de forma selectiva en los proyectos y servicios con datos sensibles, no en toda la organización.
Cientos de alertas al mes y casi ninguna corresponde a impacto real	Se alerta sobre umbrales de recursos en vez de sobre la experiencia del usuario	Define SLI y SLO y alerta por velocidad de consumo del presupuesto de error, con dos ventanas que deban coincidir.
Una alerta basada en registros llega demasiado tarde	Se evalúa como consulta programada, con la latencia acumulada de la clase 045	Convierte el patrón en una métrica basada en registros y alerta sobre ella como sobre cualquier métrica.
La traza se corta siempre en el mismo servicio	Ese salto no propaga la cabecera de contexto de traza	Propágala en todos los clientes HTTP y comprueba el número de tramos al desplegar cada servicio.
Se optimiza el servicio y la latencia no mejora	El tiempo se va dentro de un tramo, donde ninguna traza lo muestra	Activa el perfilador continuo en producción: su sobrecarga es mínima y localiza el consumo 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. ¿Qué registros están activos y son gratuitos aquí, y qué problema concreto de la clase 045 resuelven?
2. ¿Cuál es la diferencia entre un destino y una exclusión en el enrutador, y cuál baja la factura?
3. Explica qué significa una velocidad de consumo de 14,4 y por qué se exigen dos ventanas coincidentes.
4. ¿Qué tipo de alerta sigue necesitando un umbral aunque exista el presupuesto de error?
5. ¿Qué pregunta responde el perfilador que ninguna traza puede responder, y por qué?

Referencias

• Google Cloud (2025). Cloud Audit Logs overview — tipos, cuáles están activos y cuáles se pagan. <https://cloud.google.com/logging/docs/audit>

• Google Cloud (2025). Routing and storage overview — enrutador, destinos, exclusiones y buckets Required y Default. <https://cloud.google.com/logging/docs/routing/overview>

• Google Cloud (2025). Log-based metrics — contadores y distribuciones extraídos de registros. <https://cloud.google.com/logging/docs/logs-based-metrics>

• Google Cloud (2025). SLO monitoring and burn rate alerts — SLI, SLO, presupuesto de error y ventanas múltiples. <https://cloud.google.com/stackdriver/docs/solutions/slo-monitoring>

• Google Cloud (2025). Cloud Profiler concepts — muestreo continuo en producción y su sobrecarga. <https://cloud.google.com/profiler/docs/concepts-profiling>

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

Descargar: Parte 04 en PDF · Recorrido de Google Cloud en PDF · Manual integral

Anterior	Índice	Siguiente
← 056 · Pub/Sub, Cloud Tasks y Workflows	Parte 04 · Programa	058 · Cloud KMS, Secret Manager y Security Command Center →

Evaluación — 057 Cloud Logging, Monitoring, Trace y Audit Logs

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-gcp 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.

058 — Cloud KMS, Secret Manager y Security Command Center

Parte: 04 — Google Cloud: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: security · Estado: EXECUTABLECORE

Propósito

Cerrar la seguridad de la plataforma de Google Cloud con las tres piezas que la sostienen, y con dos precisiones que producen incidentes reales: un secreto referenciado como variable de entorno se resuelve al arrancar la instancia, así que rotarlo no cambia nada hasta que se despliega otra vez; y una clave rotada no deja de usarse, porque lo ya cifrado sigue necesitando la versión antigua. La clase 046 dejó la disciplina —cada control con su prueba negativa—; aquí cambian los mecanismos y aparece una herramienta que ninguna de las dos plataformas anteriores tenía: la simulación de rutas de ataque.

Resultados de aprendizaje

Al finalizar podrás:

1. Estructurar claves en anillos y versiones sabiendo qué se puede destruir y qué no se puede borrar nunca.
2. Rotar una clave entendiendo qué queda cifrado con la versión anterior y cuándo se puede destruir.
3. Referenciar secretos desde un servicio sabiendo cuándo se resuelven y qué exige una rotación.
4. Priorizar hallazgos de seguridad por ruta de ataque explotable en vez de por puntuación.
5. Verificar cada control de la parte con una prueba negativa ejecutada.

Conceptos centrales

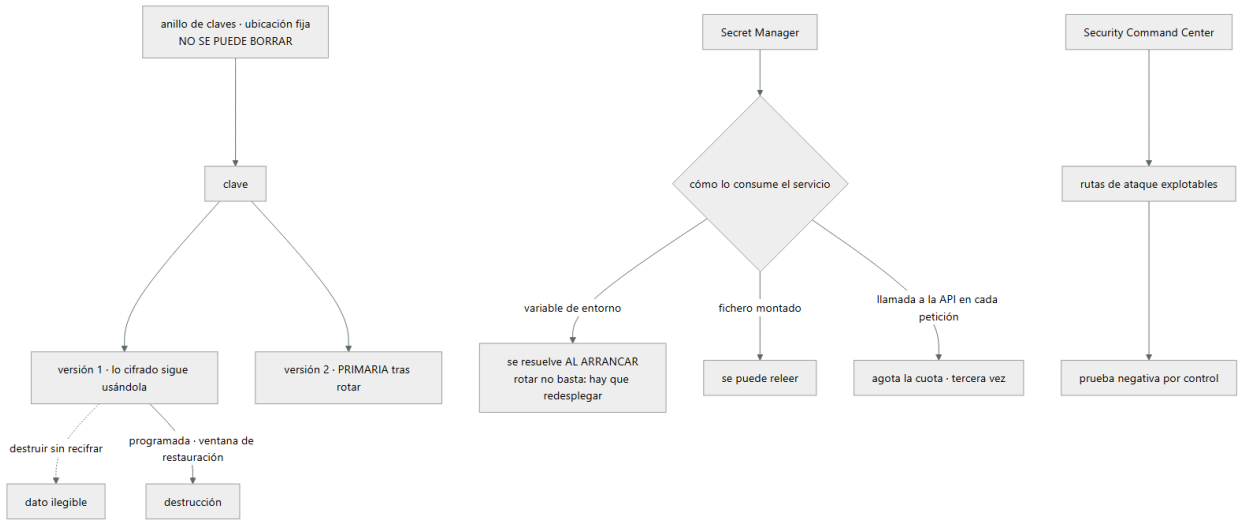
Concepto	Comprensión verificable
anillo de claves	Agrupación de claves ligada a una ubicación. No se puede borrar, ni él ni las claves que contiene: solo se destruyen versiones. Una jerarquía mal pensada se queda para siempre.
versión de clave	Material criptográfico concreto. La rotación crea una versión primaria nueva y no reescribe lo ya cifrado, que sigue dependiendo de la anterior.
destrucción programada	Una versión no se borra al instante: entra en un plazo —24 horas por defecto— durante el cual se puede restaurar. Es la red de seguridad ante el error irreversible.
nivel de protección	Software, HSM o externo. El externo mantiene la clave fuera de Google, lo que satisface requisitos de soberanía y acopla la disponibilidad del dato a la de un sistema ajeno.
referencia a secreto	Forma en que un servicio obtiene un secreto. Como variable de entorno se resuelve al arrancar la instancia; montado como fichero puede releerse en cada lectura.
simulación de rutas de ataque	Cálculo de los caminos por los que un atacante podría llegar a un recurso valioso. Prioriza por explotabilidad real, no por número de hallazgos.

Modelo mental

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

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. Claves: lo que no se borra y lo que sí se destruye

La jerarquía de Cloud KMS tiene tres niveles y una propiedad que sorprende:

```
anillo de claves    ligado a una ubicación · NO SE PUEDE BORRAR
  clave             NO SE PUEDE BORRAR
    versión         sí se puede destruir, con plazo
```

Las dos primeras líneas son literales: un anillo creado por error se queda en el proyecto para siempre. No cuesta dinero vacío y sí ensucia el inventario, así que la estructura conviene decidirla antes de crear la primera:

```
anillo por ENTORNO y UBICACIÓN, no por equipo
  kr-prod-europe-west1
    clave por PROPÓSITO
      k-almacenamiento   cifra buckets
      k-datos            cifra Cloud SQL
      k-aplicacion       cifra a nivel de aplicación
```

El criterio para separar claves es el mismo de la clase 046 y tiene consecuencias operativas: una clave es la unidad de revocación. Todo lo que cifre la misma clave se vuelve inaccesible a la vez cuando se revoca. Si eso incluye la evidencia de auditoría —como ocurrió en el simulacro de la clase 048—, el radio de impacto es mayor del previsto.

La destrucción de una versión tiene una protección que conviene conocer porque salva errores:

```
$ gcloud kms keys versions destroy 3 --key k-almacenamiento \
    --keyring kr-prod-europe-west1 --location europe-west1
# → estado DESTROY_SCHEDULED durante el plazo configurado

$ gcloud kms keys versions restore 3 --key k-almacenamiento \
    --keyring kr-prod-europe-west1 --location europe-west1
```

El plazo por defecto es de 24 horas y puede fijarse hasta 120 días al crear la clave. Es el equivalente funcional de la protección contra purga de la clase 046, con una diferencia importante: aquí es el comportamiento por defecto, no un interruptor que alguien tiene que acordarse de activar.

Y los niveles de protección, en orden de control y de acoplamiento:

```
software   la clave la custodia Google, en software
HSM        módulo de hardware certificado, gestionado por Google
externo    la clave vive FUERA de Google, en un gestor propio o de un tercero
           Google nunca la tiene: pide la operación criptográfica
```

El externo satisface requisitos de soberanía que ningún otro nivel cubre, y trae una consecuencia que hay que aceptar por escrito antes de elegirlo: la disponibilidad del dato pasa a depender de un sistema que no está en esta nube. Si el gestor externo no responde, los datos no se pueden leer. Es un intercambio legítimo cuando la obligación existe y es un riesgo gratuito cuando se elige por prudencia genérica.

Y la rotación, con la precisión que la clase 046 ya estableció y que aquí tiene una consecuencia concreta:

```
$ gcloud kms keys update k-almacenamiento --keyring kr-prod-europe-west1 \
    --location europe-west1 --rotation-period 90d --next-rotation-time 2026-09-01T00:00:00Z
```

```
rotar    crea una versión nueva y la hace primaria
         lo NUEVO se cifra con ella
no rota  lo ya cifrado, que sigue dependiendo de la versión anterior
```

De ahí la regla que evita el error irreversible: una versión antigua no se destruye hasta haber recifrado lo que dependía de ella, y comprobarlo no es opcional:

```
$ gcloud kms keys versions list --key k-almacenamiento \
    --keyring kr-prod-europe-west1 --location europe-west1 \
    --format="table(name.basename(), state, createTime)"
```

En los servicios que soportan clave del cliente, el recifrado se dispara reescribiendo el objeto o mediante la operación que el servicio ofrezca. Y hasta que termine, destruir la versión antigua deja datos ilegibles — que es la forma más cara de descubrir que «rotado» no significa «ya no se usa».

2. Secretos: cuándo se resuelven decide si la rotación sirve

Secret Manager guarda secretos con versiones inmutables y un alias latest. Y la decisión que produce el incidente de esta clase no es dónde se guarda, sino cómo lo consume el servicio.

Hay tres formas y no son equivalentes:

```
A · variable de entorno con referencia
    la plataforma resuelve el valor AL ARRANCAR la instancia
    → rotar el secreto NO cambia nada hasta que arranque una instancia nueva

B · fichero montado con referencia
    el fichero se puede releer
    → una rotación se recoge sin redesplegar, si la aplicación relee

C · llamada a la API desde la aplicación
    control total, y hay que gestionar caché y cuota
```

La forma A es la más usada porque es la más cómoda, y su comportamiento sorprende siempre:

```
$ gcloud run deploy svc-pedidos \
    --set-secrets "DB_PASSWORD=db-password:latest"
```

latest se evalúa cuando la instancia arranca. Con instancias vivas de antes de la rotación y otras nuevas, el servicio usa dos contraseñas a la vez durante horas, y la mitad de las peticiones falla. El síntoma —errores intermitentes de autenticación que se corrigen solos con el tiempo— es difícil de atribuir.

Las dos correcciones, según el caso:

```
rotación planificada    desplegar una revisión nueva como último paso
                        de la rotación: es un cambio de tráfico, segundos
rotación de emergencia  fichero montado y relectura ante fallo de autenticación
```

Y la segunda línea repite el patrón de dos credenciales de la clase 046, que sigue siendo la única forma de rotar sin ventana de fallo:

```
1. dos credenciales activas: primaria y secundaria
2. rotar la secundaria, que nadie usa
3. desplegar la revisión que apunta a la secundaria ya rotada
4. rotar la antigua primaria, ahora libre
```

La forma C tiene un riesgo conocido y es la tercera vez que aparece en el programa: leer el secreto en cada petición agota la cuota del proyecto y el fallo se manifiesta como error de la base de datos, no del gestor de secretos. La corrección es la misma que en la clase 046: cachear en memoria con vencimiento y releer solo ante un fallo de autenticación.

Dos decisiones más que se toman al crear el secreto y no se cambian:

```
$ gcloud secrets create db-password \
    --replication-policy user-managed --locations europe-west1,europe-west4
```

La política de replicación gestionada por el usuario permite elegir las regiones, que es el mismo requisito de residencia que la clase 053 resolvió eligiendo la birregión y que la clase 041 no podía resolver. Es coherente con el resto de la plataforma y hay que decidirlo al crear, porque después no cambia.

Y una limitación que conviene decir sin adornos, porque es una diferencia real frente a otra plataforma del programa: Secret Manager no rota por sí solo. Puede publicar una notificación en un tema de Pub/Sub según un calendario, y la rotación —conectarse al motor, generar la credencial nueva, guardarla como versión— la escribe el equipo.

```
$ gcloud secrets update db-password \
    --next-rotation-time 2026-09-01T03:00:00Z --rotation-period 2592000s \
    --topic projects/$P/topics/rotacion-secretos
```

Eso es un disparador, no una rotación. Compararlo con la rotación integrada de otros gestores es justo: aquí hay que construir la función que la ejecuta, y conviene presupuestarlo en lugar de descubrirlo.

3. Priorizar por ruta de ataque, no por número de hallazgos

Security Command Center hace tres cosas y se paga por niveles:

```
inventario de recursos y hallazgos de configuración   nivel básico
detección de amenazas (eventos, contenedores, máquinas)  niveles de pago
simulación de RUTAS DE ATAQUE y cumplimiento normativo   niveles de pago
```

La tercera es la que no tenía equivalente en las dos plataformas anteriores y la que cambia cómo se prioriza.

Un escáner de configuración produce cientos de hallazgos y los ordena por severidad genérica. La simulación de rutas hace otra cosa: parte de los recursos que declaras valiosos y calcula los caminos por los que alguien podría llegar a ellos, atravesando permisos, suplantaciones, reglas de red y servicios expuestos.

```
escáner clásico     "este bucket no tiene registro de acceso"   severidad media
                    "esta máquina tiene IP pública"             severidad media
                    …320 hallazgos más

ruta de ataque      internet → svc-informes (sin autenticar)
                            → cuenta de servicio sa-informes
                            → suplantación de sa-datos
                            → lectura de gs://cls-facturas
                    exposición del recurso valioso: ALTA
```

Esa segunda forma es exactamente el tipo de cadena que la clase 050 obligó a buscar a mano con el analizador de políticas. Aquí se calcula sola y se recalcula, que es la diferencia entre una auditoría puntual y un control continuo: una cuenta de servicio creada la semana siguiente a la auditoría aparece sin que nadie repita el ejercicio.

La priorización que se deduce es más honesta que una puntuación:

```
se corrige primero lo que ESTÁ en una ruta hacia un recurso valioso
después, lo que aumentaría la superficie si algo más fallara
al final, lo que es una buena práctica sin camino conocido
```

Y conviene decir lo que cuesta, porque es una partida real: los niveles de pago se facturan en proporción al gasto de la organización o por recurso protegido, así que la decisión es de alcance —qué carpetas y qué proyectos— exactamente como los planes de la clase 046. Activarlo en toda la organización sin mirar el inventario repite el error que allí costó 2.460 dólares al mes.

Y las detecciones tienen la misma exigencia que cualquier otro control:

```
activar una detección no demuestra nada
lo demuestra provocarla y ver el hallazgo
```

Una prueba segura y estándar para la detección de amenazas en la capa de identidad es crear una clave de cuenta de servicio en un proyecto donde la política de la clase 050 lo prohíbe, y comprobar que aparece el hallazgo correspondiente; para la de contenedores, ejecutar un binario desde un directorio temporal dentro de un contenedor. Lo importante no es la prueba concreta sino la regla: la fecha de la última verificación de cada detección debe existir y ser reciente.

4. La lista de pruebas negativas de la parte 04

La clase 046 estableció que un control sin prueba negativa es una afirmación. Cerrar la parte 04 exige la lista equivalente, y conviene compararla con la de allí porque las pruebas se parecen y los mensajes de error no:

```
control                             prueba negativa que lo demuestra
────────────────────────────────────────────────────────────────────────────────────────────────
bucket no público                   conceder a allUsers → 412, violación de
                                    restricción                       (049, 053)

sin claves de cuenta de servicio    crear una clave → denegado por política (050)
```

```
federación acotada por sujeto       desde otro repositorio → permiso denegado (050)

salida a internet cerrada           curl con tiempo límite → agota            (051)

firewall dirigido por identidad     prueba de conectividad → UNREACHABLE      (051)

Cloud SQL sin IP pública            conexión directa → tiempo agotado         (054)

servicio interno no público         curl sin testigo → 403                    (055)

cola de fallidos operativa          publicar un mensaje que falla →
                                    aparece en el tema de fallidos            (056)

auditoría de acceso a datos activa  leer un dato y encontrar la entrada       (057)

destrucción de versión de clave     programar y restaurar dentro del plazo

rotación de secreto sin corte       rotar con tráfico en curso: 0 errores

recuperación de bucket              borrar y restaurar uno de prueba          (053)
```

Doce afirmaciones con doce comprobaciones. Y la comparación de los tres proveedores en un aspecto concreto —qué protege la plataforma por defecto y qué hay que activar— es lo que el proyecto de la clase 060 tiene que poder resumir:

```
borrado de contenedor protegido      opcional  2 interruptores  por defecto
auditoría de administrador           90 días   90 días         400 días, gratis
auditoría de acceso a datos          opcional  opcional        apagada, cara
claves: destrucción con ventana      opcional  opcional        por defecto
creación de credenciales de larga
  duración bloqueable por política     sí          sí              sí
rutas de ataque calculadas             —           —           integrado
rotación de secretos integrada         sí       parcial         hay que construirla
```

La última fila es la única en la que esta plataforma queda por detrás de forma clara, y merece figurar como riesgo declarado en lugar de omitirse.

Y la prueba de segundo orden que la clase 046 introdujo sigue siendo la más rentable: comprobar que el control sigue activo pasado un tiempo. Aquí el mecanismo son las políticas de organización de la clase 049 y los hallazgos continuos de Security Command Center, que detectan la desviación en horas en vez de en la siguiente auditoría. Los controles no se revierten por malicia: se revierten porque alguien recreó un recurso desde una plantilla vieja, y esa es una causa que ninguna norma escrita evita.

5. Lo que este programa ya puede afirmar sobre seguridad en la nube

Con tres plataformas recorridas, hay conclusiones que ya no dependen del proveedor y conviene enunciarlas, porque son las que se llevan a las partes 11 y 22.

Primera: las credenciales de larga duración son el problema, y las tres plataformas ofrecen la misma salida. Rol de instancia, identidad administrada, cuenta de servicio adjunta; federación con sujeto acotado para lo externo. La prueba negativa ha sido idéntica tres veces y ha fallado dos de tres la primera vez que se ejecutó.

Segunda: el permiso suma y lo que resta vive en otro sistema. Políticas de control de servicios, Azure Policy, políticas de organización y de denegación. Los tres producen un error que no habla de permisos, y leerlo ahorra buscar en el sistema equivocado. Tres vocabularios, un método.

Tercera: el gobierno guarda la puerta y no limpia la casa. En las tres, imponer una restricción no corrige lo existente, y la secuencia correcta ha sido siempre inventariar, corregir y después imponer. Hacerlo al revés produjo en la clase 046 un panel en verde con catorce recursos incumpliendo.

Cuarta: el control de la clave es control real y responsabilidad real. Poder revocar significa poder dejar un dato ilegible, y el radio de impacto se estima mal siempre que un mismo recurso sirve a varios propósitos — como demostró el simulacro de la clase 048.

Quinta, y la que más veces se salta: un informe que dice «se ha habilitado X» no es comparable con uno que dice «se ha habilitado X y este es el error que devuelve al intentar lo que impide». La primera forma es una intención. La segunda es evidencia, y es la única que sobrevive a una auditoría y a un cambio de proveedor.

Lo que queda por comprobar, y que la clase 060 tiene que responder con datos: si la lista de doce pruebas negativas de esta parte se puede generar a partir de la línea base en lugar de escribirse a mano cada vez. Si la respuesta es que sí, el trabajo de seguridad de la cuarta plataforma será una fracción del de esta — que es exactamente la promesa que este programa hizo en la parte 01 y que ya se puede empezar a medir.

Ejemplo trabajado

CloudShop cierra la seguridad de su plataforma en Google Cloud. Llega con la línea base de la clase 048 y la lista de pruebas negativas de la 046, así que tres controles se montan sin incidentes. Los cuatro problemas de este mes son mecanismos nuevos mal entendidos.

Lo que se montó sin incidentes, por venir escrito:

```
sin credenciales de larga duración: bloqueadas por política   (046 → 050)
inventariar, corregir y después imponer                        (046 → 049)
separar por propósito lo que comparte clave                    (048 → 058)
cada control con su prueba negativa                            (046 → 058)
```

La tercera venía directamente del simulacro de la clase 048, donde revocar una clave detuvo tres servicios en vez de uno. Aquí las claves se separaron desde el principio:

```
k-facturas       cifra el bucket de facturas
k-auditoria      cifra el archivo de registros
k-datos          cifra Cloud SQL
```

Problema 1 — la mitad de las instancias con la contraseña antigua.

Se rota la contraseña de la base de datos a las 03:00. A las 09:20 siguen apareciendo errores intermitentes de autenticación.

```
$ gcloud run services describe svc-pedidos \
    --format="value(spec.template.spec.containers[0].env)"
DB_PASSWORD → secretKeyRef: db-password:latest
```

La referencia se resuelve al arrancar la instancia. Las instancias vivas desde antes de las 03:00 seguían con la contraseña anterior; las nuevas, con la nueva. Seis horas con dos credenciales en circulación.

```
referencia al secreto              variable de entorno   fichero montado
relectura ante fallo de autenticación      no                 sí
rotación                             cambiar la versión   dos credenciales:
                                                          rotar la secundaria,
                                                          desplegar, rotar la otra
errores durante la rotación siguiente    3.412              0
```

Problema 2 — errores 500 que parecían de la base de datos. Tercera vez.

```
RESOURCE_EXHAUSTED: Quota exceeded for quota metric
  'Access secret versions' of service 'secretmanager.googleapis.com'
```

Un servicio nuevo leía el secreto en cada petición. Es el mismo incidente de la clase 046 con Key Vault y de la 051 con los puertos de traducción: el mismo defecto de código produce el mismo fallo en la tercera plataforma.

```
lecturas del gestor de secretos   1 por petición   1 cada 30 min
relectura ante fallo de auth            no              sí
errores 500 en el pico                4,1 %           0,0 %
```

Problema 3 — una versión de clave destruida por error, recuperada en la ventana.

Durante una limpieza, alguien programó la destrucción de la versión 2 de k-facturas creyendo que la rotación a la versión 3 la había dejado sin uso.

```
$ gcloud kms keys versions list --key k-facturas --keyring kr-prod-europe-west1 \
    --location europe-west1 --format="table(name.basename(), state)"
2   DESTROY_SCHEDULED
3   ENABLED
```

El 71 % de los objetos del bucket seguían cifrados con la versión 2. Destruirla los habría dejado ilegibles de forma permanente.

```
$ gcloud kms keys versions restore 2 --key k-facturas \
    --keyring kr-prod-europe-west1 --location europe-west1
```

```
plazo de destrucción programada            24 h (por defecto)   30 días
recifrado tras rotar                         no había            proceso
```

```
                                                        automatizado
comprobación previa a destruir                ninguna   obligatoria:
                                                        objetos por versión
objetos aún cifrados con la versión 2          71 %           0 %
```

La ventana de destrucción salvó el caso. Es el mismo papel que la protección contra purga de la clase 046 y con una diferencia importante: aquí venía puesta. El equipo la amplía a 30 días de todos modos, porque 24 horas no cubren un fin de semana.

Problema 4 — una ruta de ataque que la auditoría de la clase 050 no podía ver.

Security Command Center señala un camino con exposición alta:

```
internet → svc-informes (desplegado sin autenticación hace 9 días)
         → cuenta de servicio sa-informes-v2
         → suplantación de sa-datos
         → lectura de gs://cls-facturas
```

La auditoría de suplantaciones se había hecho tres semanas antes y estaba limpia. sa-informes-v2 y el servicio sin autenticar se crearon después.

```
revisión de rutas de suplantación        puntual, manual   continua
servicios sin autenticación                   2 de 9         0 de 9
cadenas de suplantación hacia datos             1              0
tiempo desde la creación hasta la detección     —            4 horas
```

La diferencia no es la herramienta: es que una auditoría es una foto y un hallazgo continuo es una película. La clase 050 obligó a mirar los caminos; esta hace que se miren solos.

Y una decisión documentada como riesgo aceptado.

Se evaluó el gestor de claves externo para las facturas, por un requisito de soberanía que resultó ser una preferencia y no una obligación:

```
lo que aporta   la clave nunca está en Google
lo que cuesta   la disponibilidad del dato depende de un sistema externo
                y su indisponibilidad hace ilegible el bucket
decisión        no se adopta; se documenta con la condición que
                obligaría a revisarlo: una obligación contractual explícita
```

Resumen del cierre de seguridad:

```
claves por propósito                          1 para todo       3
plazo de destrucción de versiones               24 h          30 días
objetos cifrados con una versión programada
  para destrucción                              71 %            0 %
errores durante una rotación de secreto        3.412            0
errores 500 por cuota del gestor de secretos    4,1 %          0,0 %
servicios sin autenticación                    2 de 9          0 de 9
rutas de ataque hacia datos valiosos              1              0
controles con prueba negativa ejecutada         0 de 12       12 de 12
```

La lección que esta clase traslada al proyecto de la clase 060: los cuatro problemas fueron de mecanismos nuevos, y dos de ellos eran el mismo defecto de código que ya había fallado en dos plataformas anteriores. Lo que evitó los otros tres controles fue una lista escrita, no la experiencia de nadie. La seguridad de la tercera plataforma costó menos porque la segunda dejó doce pruebas escritas, y ese es exactamente el rendimiento que este programa prometió medir.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-04-gcp-core-platform/058-cloud-kms-secret-manager-y-security-command-center/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-gcp 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-gcp 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 rotar un secreto, parte de las peticiones sigue fallando durante horas	La referencia como variable de entorno se resuelve al arrancar la instancia	Móntalo como fichero y relee ante fallo de autenticación, o despliega una revisión nueva como último paso de la rotación.
Errores 500 que parecen de la base de datos y no lo son	Se lee el secreto en cada petición y se agota la cuota del proyecto	Cachea en memoria con vencimiento y relee solo ante un fallo de autenticación.
Destruir una versión de clave deja datos ilegibles	La rotación no recifra lo existente: sigue dependiendo de la versión anterior	Comprueba cuántos objetos usan cada versión antes de destruir, amplía el plazo de destrucción y automatiza el recifrado.
Un anillo o una clave creados por error no se pueden eliminar	En Cloud KMS solo se destruyen versiones; anillos y claves son permanentes	Decide la jerarquía antes de crear la primera: anillo por entorno y ubicación, clave por propósito y unidad de revocación.
Una auditoría de permisos limpia deja pasar una ruta de acceso a datos	La auditoría es una foto y los recursos nuevos se crean después	Usa la simulación de rutas de ataque como control continuo y corrige primero lo que está en un camino hacia un recurso valioso.
Se espera rotación automática de secretos y no ocurre	Secret Manager notifica según un calendario pero no ejecuta la rotación	Escribe la función de rotación disparada por la notificación y presupuesta ese trabajo; declara el riesgo mientras no exista.

■ 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 no se puede borrar nunca en Cloud KMS y qué implica para el diseño de la jerarquía?
2. Rotas una clave. ¿Qué queda cifrado con la versión anterior y cuándo puedes destruirla?

3. ¿Cuándo se resuelve un secreto referenciado como variable de entorno, y qué incidente produce durante una rotación?
4. ¿En qué se diferencia priorizar por ruta de ataque de priorizar por puntuación de seguridad?
5. Enumera cinco controles de la parte 04 con la prueba negativa concreta que demuestra cada uno.

Referencias

- Google Cloud (2025). Cloud KMS resource hierarchy — anillos, claves y versiones, y qué no se puede eliminar. <https://cloud.google.com/kms/docs/resource-hierarchy>
- Google Cloud (2025). Key rotation — versión primaria, datos ya cifrados y recifrado. <https://cloud.google.com/kms/docs/key-rotation>
- Google Cloud (2025). Use Secret Manager with Cloud Run — referencias como variable de entorno y como volumen montado. <https://cloud.google.com/run/docs/configuring/services/secrets>
- Google Cloud (2025). Secret rotation — notificaciones programadas y qué hay que construir. <https://cloud.google.com/secret-manager/docs/secret-rotation>
- Google Cloud (2025). Attack path simulations in Security Command Center — exposición de recursos valiosos y priorización. <https://cloud.google.com/security-command-center/docs/attack-exposure-learn>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 04 en PDF · Recorrido de Google Cloud en PDF · Manual integral

Anterior	Índice	Siguiente
← 057 · Cloud Logging, Monitoring, Trace y Audit Logs	Parte 04 · Programa	059 · Terraform y despliegues reproducibles en GCP →

Evaluación — 058 Cloud KMS, Secret Manager y Security Command Center

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-gcp 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.

059 — Terraform y despliegues reproducibles en GCP

Parte: 04 — Google Cloud: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: iac · Estado: EXECUTABLECORE

Propósito

Declarar la infraestructura de Google Cloud con Terraform, entendiendo la pieza que la clase 047 no tenía y que lo cambia todo: el archivo de estado. Da lo que a Bicep le faltaba —saber qué gestiona este código, destruirlo ordenadamente, detectar que alguien borró algo— y trae tres problemas propios: hay que guardarlo, hay que bloquearlo, y contiene los secretos en claro. A eso se suman dos trampas del proveedor que destruyen recursos sanos sin que nadie lo pida.

Resultados de aprendizaje

Al finalizar podrás:

1. Explicar qué capacidades da el archivo de estado y qué obligaciones impone su custodia.
2. Garantizar que lo que se aplica es exactamente lo que se revisó, y no una nueva planificación.
3. Evitar la destrucción de recursos ajenos al cambio por reindexación y por estado compartido.
4. Autenticar la canalización sin ninguna clave, con federación y suplantación.
5. Comparar un modelo con estado y otro sin él con criterios operativos, no de preferencia.

Conceptos centrales

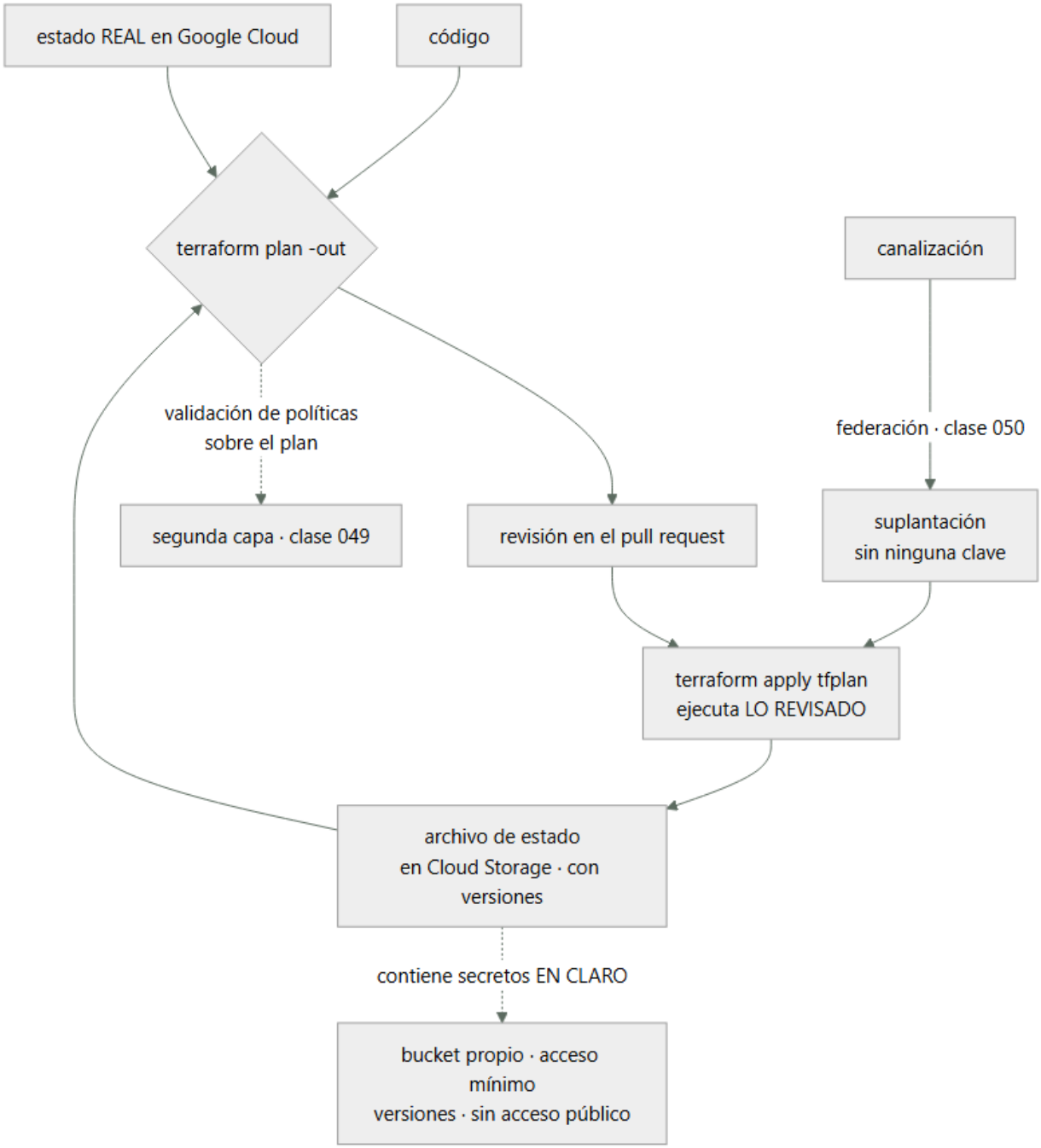
Concepto	Comprensión verificable
archivo de estado	Registro de lo que este código gestiona y de sus atributos. Permite destruir, detectar desviación e importar — y guarda valores sensibles en claro, así que es el fichero más delicado del proyecto.
bloqueo de estado	Exclusión mutua para que dos ejecuciones simultáneas no escriban a la vez. Sin él, dos canalizaciones concurrentes pueden corromper el estado.
plan guardado	Resultado de la planificación escrito a fichero. Aplicar ese fichero garantiza que se ejecuta lo revisado; sin él, apply vuelve a planificar y puede hacer otra cosa.
reindexación por count	Las instancias de un recurso con count se identifican por posición. Quitar una del medio desplaza a todas las siguientes, y Terraform destruye y recrea las que no cambiaron.
suplantación del proveedor	Autenticación de Terraform como una cuenta de servicio sin ninguna clave, partiendo de la identidad de la canalización. Es la aplicación directa de la clase 050.
desviación	Diferencia entre lo declarado y lo que existe de verdad. Con estado se detecta en la planificación; sin estado, un borrado ajeno pasa desapercibido.

Modelo mental

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

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 estado: lo que resuelve y lo que obliga

La clase 047 terminó con una lista de lo que se pierde al no tener estado. Terraform lo tiene, así que esa lista se convierte en capacidades:

```
saber qué gestiona este código   el estado es la lista de pertenencia
destruir ordenadamente           terraform destroy, en orden de dependencias
detectar que alguien BORRÓ algo  la planificación lo muestra como cambio
adoptar lo creado a mano         import, sin recrear nada
```

La tercera es la que más se echa de menos sin estado: si alguien elimina un recurso desde la consola, un modelo sin estado simplemente lo vuelve a crear en el siguiente despliegue, sin decir que había desaparecido. Con estado, la planificación lo señala:

```
# google_compute_firewall.permitir_datos has been deleted
  - resource "google_compute_firewall" "permitir_datos" { … }
Plan: 1 to add, 0 to change, 0 to destroy.
```

Y las tres obligaciones que trae, en orden de importancia:

Contiene secretos en claro. Cualquier valor que haya pasado por un recurso —una contraseña generada, una clave, un certificado— queda escrito. No hay cifrado a nivel de campo:

```
$ terraform state pull | jq -r '.resources[].instances[].attributes | select(.password)' | head -3
```

Eso convierte al estado en el fichero más sensible del repositorio, y su custodia en una decisión de seguridad, no de almacenamiento:

```
terraform {
  backend "gcs" {
    bucket = "cls-tfstate-prod"
    prefix = "red"
  }
}
```

```
bucket propio, en un proyecto de plataforma
acceso uniforme y prevención de acceso público   (clase 053)
versionado activo: un estado corrupto se recupera
IAM mínimo: quien puede leer el estado puede leer los secretos
clave del cliente si el requisito lo exige        (clase 058)
```

Hay que bloquearlo. Dos ejecuciones a la vez sobre el mismo estado lo corrompen. El backend de Cloud Storage lo gestiona, y conviene comprobarlo la primera vez en lugar de suponerlo: una segunda ejecución debe esperar o fallar con un error de bloqueo, no continuar.

Hay que decidir su granularidad, y esa decisión es la misma que en la clase 047 con otra pieza. Allí el radio de impacto lo marcaba el grupo de recursos; aquí lo marca el estado:

```
un estado para toda la plataforma
  → terraform destroy en pruebas puede tocar lo que comparte estado
  → cada planificación consulta cientos de recursos: lenta
  → dos equipos se bloquean mutuamente

un estado por unidad de despliegue
  cls-tfstate-prod/red
  cls-tfstate-prod/datos
  cls-tfstate-prod/tienda
  → cada uno se destruye y se planifica por separado
```

Y para conectarlos, salidas remotas o —mejor— fuentes de datos que consultan Google Cloud directamente, que no crean acoplamiento entre estados:

```
data "google_compute_subnetwork" "tienda" {
  name    = "snet-tienda-euw1"
  region  = "europe-west1"
  project = var.proyecto_red
}
```

Una dependencia por dato es más robusta que una por estado: si el estado de red se recrea, el consumidor no se entera.

2. Aplicar lo revisado, y las dos formas de destruir sin querer

El plan guardado no es opcional. Sin él, apply vuelve a planificar en el momento de ejecutarse, y entre la revisión y la ejecución pueden haber cambiado cosas: una fuente de datos, una versión de imagen, un recurso tocado a mano.

```
$ terraform plan -out=tfplan -lock-timeout=5m
$ terraform show -no-color tfplan > plan.txt      # esto es lo que se revisa
$ terraform apply tfplan                          # esto ejecuta EXACTAMENTE eso
```

La diferencia con terraform apply a secas es que el segundo pide confirmación sobre un plan nuevo que nadie ha revisado. En una canalización con aprobación manual, eso rompe la garantía entera: se aprueba un texto y se ejecuta otro.

Y ahora las dos formas de destruir recursos sanos, que son las que producen los incidentes de esta clase.

La primera: count y la reindexación. Las instancias creadas con count se identifican por su posición:

```
resource "google_compute_firewall" "reglas" {
  count = length(var.reglas)
  name  = var.reglas[count.index].nombre
  # …
```

```
}
```

Quitar el segundo elemento de una lista de cinco desplaza a los tres siguientes, y Terraform los ve como cambios de identidad:

```
  # google_compute_firewall.reglas[1] must be replaced
  # google_compute_firewall.reglas[2] must be replaced
  # google_compute_firewall.reglas[3] must be replaced
Plan: 3 to add, 0 to change, 4 to destroy.
```

Se pidió quitar una regla y se van a recrear tres que no cambiaron. Con reglas de firewall el efecto es una ventana sin protección; con bases de datos o discos, es pérdida de datos.

La corrección es identificar por clave estable en lugar de por posición:

```
resource "google_compute_firewall" "reglas" {
  for_each = { for r in var.reglas : r.nombre => r }
  name     = each.value.nombre
  # …
}
```

Con foreach, quitar un elemento afecta solo a ese elemento. La regla de equipo es corta y ahorra incidentes: count únicamente para «cero o uno»; para colecciones, siempre foreach.

La segunda: el estado compartido y destroy. Es el mismo incidente que en la clase 047 causó el modo completo, con otro mecanismo: terraform destroy en un directorio cuyo estado incluye recursos compartidos se los lleva.

Las protecciones, que se ponen antes y no después:

```
resource "google_sql_database_instance" "pedidos" {
  # …
  deletion_protection = true            # protección del propio proveedor
  lifecycle { prevent_destroy = true }  # Terraform se niega a planificar su borrado
}
```

Las dos capas hacen cosas distintas: preventdestroy hace fallar la planificación —así que el error aparece en la revisión, antes de ejecutar nada—, y deletionprotection lo impide en el servicio aunque alguien lo intente por otra vía. Y para el proyecto entero existe un gravamen que impide su eliminación, que es el equivalente al bloqueo de recurso de la clase 042:

```
$ gcloud resource-manager liens create --restrictions=resourcemanager.projects.delete \
    --parent=projects/cls-datos-prod-euw1-01 --reason="produccion"
```

Y una comprobación que conviene automatizar en la canalización: rechazar cualquier plan que destruya recursos no esperados.

```
$ terraform show -json tfplan | jq -r '
    .resource_changes[] | select(.change.actions | index("delete"))
    | .address' | tee borrados.txt
$ [ ! -s borrados.txt ] || echo "revisar: el plan destruye recursos"
```

No se trata de prohibir los borrados, sino de que ninguno pase sin que alguien lo haya leído.

3. Autenticar sin claves, que es lo que la clase 050 exigía

La forma cómoda de autenticar Terraform en una canalización es una clave de cuenta de servicio en un secreto del sistema de CI. Es exactamente lo que la clase 050 eliminó, y lo que la política de organización de la clase 049 impide crear.

La forma correcta encadena las dos piezas ya montadas:

```
- uses: google-github-actions/auth@v2
  with:
    workload_identity_provider: projects/418293047512/locations/global/workloadIdentityPools/cls-ci/
providers/github
    service_account: tf-plan@cls-plataforma.iam.gserviceaccount.com
```

Y dentro de Terraform, la suplantación permite que la identidad de planificación y la de aplicación sean distintas:

```
provider "google" {
  project                     = var.proyecto
  region                      = var.region
  impersonate_service_account = var.cuenta_terraform
}
```

```
tf-plan@…    permisos de LECTURA sobre todo lo que gestiona
             se usa en los pull requests: puede planificar, no puede cambiar
tf-apply@…   permisos de escritura acotados
             solo desde la rama principal, con la condición de atributo de la clase 050
```

Esa separación es la que hace seguro publicar el plan en el pull request: la identidad que lo genera no puede aplicar nada, así que una rama maliciosa no consigue más que leer. Sin ella, ejecutar plan desde una rama arbitraria es dar permiso de escritura a quien pueda abrir un pull request.

Y dos particularidades del proveedor de Google que producen fallos intermitentes si no se anticipan:

Las API hay que habilitarlas y tardan en propagarse (clase 049):

```
resource "google_project_service" "servicios" {
  for_each = toset(["run.googleapis.com", "sqladmin.googleapis.com",
                    "secretmanager.googleapis.com"])
  project            = var.proyecto
  service            = each.value
  disable_on_destroy = false
}
```

disableondestroy = false importa: al destruir el directorio, deshabilitar una API afecta a todo el proyecto, incluido lo que gestionan otros. Es un borrado con radio de impacto mayor del que el código sugiere.

Hay dos proveedores, estable y beta. Un recurso o un atributo puede existir solo en beta, y mezclarlos exige declararlo en el recurso:

```
resource "google_some_resource" "x" {
  provider = google-beta
  # …
}
```

Y conviene saber lo que implica: un recurso en beta puede cambiar de forma incompatible entre versiones del proveedor. Fijar la versión del proveedor deja de ser una buena práctica y pasa a ser un requisito:

```
terraform {
  required_version = "~> 1.9"
  required_providers {
    google      = { source = "hashicorp/google",      version = "~> 6.12" }
    google-beta = { source = "hashicorp/google-beta", version = "~> 6.12" }
  }
}
```

Y el fichero de bloqueo de dependencias se versiona en el repositorio, por la misma razón que cualquier otro: dos ejecuciones del mismo código en semanas distintas deben usar el mismo proveedor.

4. La canalización y la segunda capa de gobierno

El flujo completo, que es el entregable de esta clase:

```
en el pull request
  1. terraform fmt -check        formato
  2. terraform validate          sintaxis y referencias
  3. tflint                      errores propios del proveedor
  4. terraform plan -out=tfplan  con la identidad de LECTURA
  5. validación de políticas sobre el plan
  6. publicar el plan en el pull request

al fusionar en la rama principal
  7. terraform apply tfplan      con la identidad de escritura, desde main
```

El paso 5 es el que convierte esto en gobierno y no solo en automatización. La clase 049 dejó las políticas de organización, que actúan en el momento de la llamada a la API; una validación sobre el plan actúa antes, en la revisión, y puede expresar reglas que una política de organización no expresa:

```
package terraform.cloudshop

deny[msg] {
  r := input.resource_changes[_]
  r.type == "google_storage_bucket"
  not r.change.after.uniform_bucket_level_access
  msg := sprintf("%s: falta acceso uniforme a nivel de bucket", [r.address])
}
```

```
deny[msg] {
  r := input.resource_changes[_]
  r.type == "google_sql_database_instance"
  r.change.after.settings[_].ip_configuration[_].ipv4_enabled
  msg := sprintf("%s: no debe tener IP pública (clase 054)", [r.address])
}
```

Son las dos capas de la clase 048 otra vez, con otros nombres: el código fija el estado al crear y la política vigila después, y ahora se añade una tercera que avisa antes de crear, en la revisión, cuando corregir cuesta menos.

Y el resultado de las tres capas juntas, expresado como decisiones de las clases anteriores que quedan declaradas en un solo recurso:

```
resource "google_storage_bucket" "facturas" {
  name                        = "cls-facturas"
  location                    = "EU"
  storage_class               = "STANDARD"
  uniform_bucket_level_access = true                      # clase 053
  public_access_prevention    = "enforced"                # clases 049, 053
  versioning { enabled = true }                           # clase 053

  soft_delete_policy { retention_duration_seconds = 604800 }

  lifecycle_rule {                                         # clase 053
    condition { days_since_noncurrent_time = 90 }
    action    { type = "Delete" }
  }

  encryption { default_kms_key_name = google_kms_crypto_key.facturas.id }  # 058
  labels = local.etiquetas                                 # clase 049

  lifecycle { prevent_destroy = true }
}
```

Siete decisiones de cinco clases distintas, en un recurso, imposibles de olvidar al crear el siguiente. Esa es la función real de la infraestructura como código en este programa, y es idéntica a la que la clase 047 enunció con otra herramienta.

5. Con estado o sin él: la comparación honesta

Con las clases 047 y 059 hechas, se puede comparar sin partidismo. La parte 07 profundiza; aquí interesa el criterio operativo.

	Bicep / ARM (047)	Terraform (059)
Estado	No: pregunta a la nube	Fichero que hay que custodiar
Saber qué gestiona el código	No, salvo pilas de despliegue	Sí
Detectar un borrado ajeno	No	Sí, en la planificación
Destrucción ordenada	Modo completo o pilas	destroy, por dependencias
Secretos	No quedan en ningún estado	En claro en el estado
Ejecuciones simultáneas	Sin problema	Requieren bloqueo
Multiproveedor	No	Sí
Adoptar recursos existentes	Automático	import explícito
Fidelidad de la previsión	what-if, con ruido	Plan, más fiel, con known after apply

Dos filas resumen el intercambio: el estado compra visibilidad y cobra custodia. Saber qué gestionas y detectar desviación son capacidades operativas de primer orden; a cambio, el fichero más sensible del proyecto pasa a ser responsabilidad tuya.

Y hay una fila que no está en la tabla porque no es técnica y suele decidir: Terraform vale para los tres proveedores de este programa. Un equipo que opera en más de una nube paga el coste de aprender una herramienta en vez de tres, y —más importante— puede aplicar el mismo flujo de revisión, la misma validación de políticas y la misma canalización en todas. Eso es exactamente el tipo de contrato portable que la clase 048 pedía identificar, y aparece en la capa que

más trabajo repetido genera.

Lo que no hace portable a Terraform, y conviene decirlo para no crear una expectativa falsa: los recursos son específicos del proveedor. Un módulo de red de Google Cloud no vale en Azure. Lo portable es el método —planificar, revisar, validar, aplicar, versionar—, no el código.

Y dos límites que comparten las dos herramientas y que la parte 07 desarrollará:

```
no hay transacción      un fallo a mitad deja lo ya aplicado
                        → las plantillas deben ser reejecutables
no cubren el día dos    ninguna gestiona una migración de datos,
                        un cambio de esquema ni un despliegue progresivo
```

La segunda es la que más veces se olvida al defender la infraestructura como código: describe el estado deseado, no el camino para llegar a él cuando el camino importa. Para eso están las clases de entrega continua de la parte 08.

Ejemplo trabajado

CloudShop pasa su plataforma de Google Cloud a Terraform. El equipo llega con la experiencia de la clase 047 y evita dos de sus incidentes; los cuatro que aparecen son propios del modelo con estado.

Lo que se evitó por venir escrito de la clase 047:

```
un estado por unidad de despliegue, no uno compartido
decidir una sola forma por tipo de recurso hijo
plan obligatorio y revisado antes de aplicar
nada de secretos en salidas
```

Incidente 1 — catorce personas podían leer todas las contraseñas.

Una revisión de accesos sobre el bucket de estado:

```
$ gcloud storage buckets get-iam-policy gs://cls-tfstate-prod \
    --format="value(bindings.members)" | tr ';' '\n' | wc -l
14
$ terraform state pull | grep -c '"password"'
7
```

El bucket había heredado los permisos del proyecto de plataforma, donde catorce personas tenían lectura. Siete contraseñas en claro, incluidas las de las bases de datos de producción.

```
proyecto del bucket de estado          plataforma      proyecto propio
principales con lectura                    14               2
versionado del bucket                       no              sí
prevención de acceso público           no aplicada        aplicada
cifrado con clave del cliente               no        sí (clase 058)
contraseñas rotadas tras el hallazgo         —           las 7
```

La rotación no era opcional: un secreto que estuvo accesible se considera comprometido, exactamente como en la clase 047 con el historial de despliegues.

Incidente 2 — se pidió quitar una regla y se recrearon tres.

```
Plan: 3 to add, 0 to change, 4 to destroy.
```

La revisión del pull request lo detectó antes de aplicar, porque el plan se publicaba y alguien lo leyó. Las reglas de firewall estaban declaradas con count sobre una lista, y quitar el segundo elemento desplazó a los tres siguientes.

```
identificación de las instancias         por posición   por nombre (`for_each`)
recursos afectados al quitar uno              4              1
ventana sin regla de firewall              ~40 s por regla     ninguna
comprobación de borrados en el plan          ninguna      falla si hay borrados
                                                          no declarados
```

Incidente 3 — se aprobó un plan y se ejecutó otro.

Una aplicación en producción creó una instancia con un tipo de máquina distinto al revisado. La causa:

```
# la canalización hacía esto
$ terraform plan          # se publica y se aprueba
$ terraform apply -auto-approve   # ← vuelve a planificar
```

Entre la revisión y la ejecución, una fuente de datos que resolvía la última imagen devolvió una versión nueva.

```
plan                                  sin guardar        -out=tfplan
```

```
aplicación                          replanifica         aplica el fichero
diferencias entre lo aprobado y lo aplicado   1              0
plan publicado en el pull request          sí               sí
```

Incidente 4 — la canalización usaba una clave, contra la política.

```
$ gcloud asset search-all-resources --scope organizations/$ORG_ID \
    --asset-types iam.googleapis.com/ServiceAccountKey \
    --query "NOT name:*/keys/system-managed*" --format="value(name)"
projects/cls-plataforma/serviceAccounts/terraform@…/keys/9f2c…
```

Una clave creada antes de aplicar la política de la clase 050, guardada en el sistema de CI. Y con permisos de escritura, usada también en los pull requests: cualquiera que abriera uno podía ejecutar una planificación con credenciales de escritura.

```
autenticación                         clave en el CI    federación (050)
identidad en pull request           escritura          lectura, tf-plan@
identidad en la rama principal      la misma           escritura, tf-apply@
condición de atributo                 ninguna     repositorio + rama main
claves de cuenta de servicio              1                0
```

Y la prueba negativa, la cuarta vez que este programa la ejecuta:

```
plan desde una rama de trabajo         funciona, solo lectura     ✓
apply desde una rama de trabajo        permiso denegado           ✓
apply desde main                       funciona                   ✓
```

Incidente 5 — destroy en pruebas tocó algo compartido.

El directorio de pruebas incluía el recurso googleprojectservice de las API, con el valor por defecto:

```
google_project_service.servicios["run.googleapis.com"] will be destroyed
```

Deshabilitar esa API afecta al proyecto entero, no solo a lo que gestiona ese directorio. La destrucción del entorno de pruebas dejó sin servicio a dos despliegues que compartían proyecto.

```
disable_on_destroy                      true (defecto)      false
proyectos por entorno                   compartido       uno por entorno
protección de recursos críticos           ninguna     prevent_destroy +
                                                      deletion_protection + gravamen
duración del incidente                    26 min             —
```

Resumen del paso a infraestructura declarada:

```
principales con acceso al estado                14              2
secretos en claro accesibles                     7              0
recursos destruidos sin pedirlo               4 en 1 cambio     0
planes aplicados distintos del revisado          1              0
claves de cuenta de servicio en la canalización  1              0
estados de Terraform                             1              4
duración de una planificación                 3 min 40 s      38 s
reglas de política validadas sobre el plan       0             11
```

La lección que esta clase traslada al proyecto de la clase 060 y a la parte 07: el archivo de estado da tres capacidades que Bicep no tenía y cobra por ellas con una responsabilidad concreta —es el fichero que contiene todos los secretos, y su control de acceso es un control de seguridad de primer nivel, no una configuración de almacenamiento—. Y las dos formas de destruir sin querer que aparecieron aquí, la reindexación y el estado compartido, son la misma lección que el modo completo de la clase 047 con otro mecanismo: una herramienta declarativa hace exactamente lo que dice el código, y el código dice más cosas de las que parece.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-04-gcp-core-platform/059-terraform-y-despliegues-reproducibles-en-gcp/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-gcp 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-gcp 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
Los secretos de producción son legibles por más gente de la prevista	El archivo de estado guarda los valores en claro y su bucket heredó permisos amplios	Bucket propio en un proyecto de plataforma, IAM mínimo, versionado, sin acceso público, y rota lo que estuvo expuesto.
Quitar un elemento de una lista destruye y recrea varios recursos	Las instancias con count se identifican por posición y se reindexan	Usa foreach con una clave estable; reserva count para el caso de cero o uno.
Se aplica algo distinto de lo que se revisó	apply sin plan guardado vuelve a planificar en el momento de ejecutarse	terraform plan -out=tfplan y terraform apply tfplan; publica el plan revisado en el pull request.
Cualquiera que abre un pull request puede modificar infraestructura	La canalización usa una única identidad con permisos de escritura para planificar y aplicar	Dos cuentas: lectura para planificar en ramas, escritura solo desde la principal, con federación y condición de atributo.
Destruir un entorno de pruebas deja sin servicio a otro despliegue	El estado incluía recursos de alcance de proyecto, como la habilitación de una API	Un proyecto por entorno, disableondestroy = false, y protecciones en los recursos críticos.
Dos ejecuciones simultáneas corrompen el estado	No hay bloqueo configurado o se está usando un backend que no lo soporta	Usa el backend de Cloud Storage y comprueba que una segunda ejecución espera o falla, en vez de suponerlo.

■ 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 tres capacidades da el archivo de estado que la clase 047 no tenía, y qué obligación impone cada una?
2. ¿Por qué terraform apply sin plan guardado rompe la garantía de una aprobación manual?

3. Explica la reindexación por count con un ejemplo y di cuándo sigue siendo correcto usarlo.
4. ¿Por qué la identidad que planifica en un pull request no debe poder aplicar?
5. ¿Qué hace portable a Terraform entre proveedores y qué no lo hace?

Referencias

- HashiCorp (2025). State: purpose and sensitive data — qué guarda el estado y por qué se protege. <https://developer.hashicorp.com/terraform/language/state/sensitive-data>
- HashiCorp (2025). count and foreach — identificación por posición frente a clave estable. <https://developer.hashicorp.com/terraform/language/meta-arguments/foreach>
- HashiCorp (2025). Command: plan and saved plan files — planificar, guardar y aplicar exactamente lo revisado. <https://developer.hashicorp.com/terraform/cli/commands/plan>
- Google Cloud (2025). Terraform GCS backend and state management — bucket de estado, bloqueo y versionado. <https://cloud.google.com/docs/terraform/resource-management/store-state>
- Google Cloud (2025). Authenticate Terraform with Workload Identity Federation — suplantación sin claves en la canalización. <https://cloud.google.com/blog/products/identity-security/enabling-keyless-authentication-from-github-actions>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 04 en PDF · Recorrido de Google Cloud en PDF · Manual integral

Anterior	Índice	Siguiente
← 058 · Cloud KMS, Secret Manager y Security Command Center	Parte 04 · Programa	060 · Proyecto: aplicación de tres capas en Google Cloud →

Evaluación — 059 Terraform y despliegues reproducibles en GCP

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-gcp 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.

060 — Proyecto: aplicación de tres capas en Google Cloud

Parte: 04 — Google Cloud: plataforma esencial Nivel: intermedio · Horas estimadas: 8 Laboratorio: capstone · Estado: EXECUTABLECORE

Propósito

Integrar las once clases anteriores en una plataforma de Google Cloud que funcione, se observe, falle de forma controlada y cueste algo explicable — y calificar con evidencia la hipótesis que escribió la clase 048. Esa hipótesis acertó en dos de tres afirmaciones y falló en la tercera, y el fallo es más valioso que los aciertos: la familia de incidentes más cara de esta parte no fue la que se predijo, y corregir la ley con lo observado es lo que se lleva a la parte 05.

Resultados de aprendizaje

Al finalizar podrás:

1. Trazar cada componente hasta la clase que lo decidió y la alternativa descartada.
2. Calificar la hipótesis de la clase 048 con evidencia, incluida la parte que resultó falsa.
3. Enunciar las leyes que ya se han observado en tres plataformas independientes.
4. Provocar tres fallos y medir detección, impacto y recuperación, incluida la recuperación bloqueada.
5. Comparar tres plataformas por costo por unidad y explicar por qué la mayor partida es la misma en las tres.

Conceptos centrales

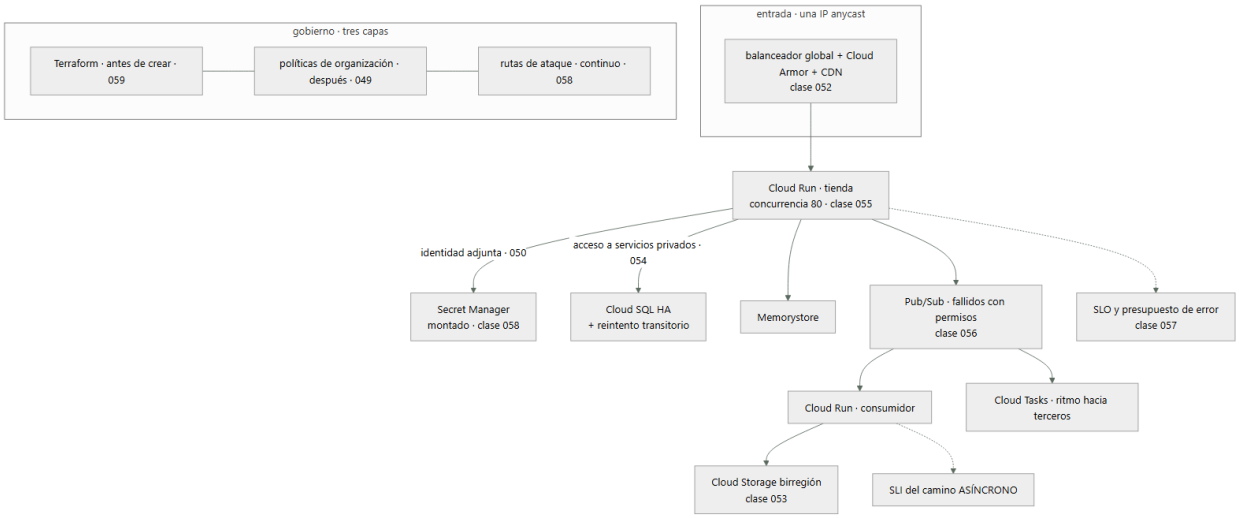
Concepto	Comprensión verificable
hipótesis calificada	Predicción escrita antes de la evidencia y evaluada después, incluida la parte errónea. Sin la parte errónea no es una hipótesis: es un resumen.
ley observada	Comportamiento que ha aparecido en tres plataformas independientes con tres mecanismos distintos. Deja de ser una peculiaridad del proveedor y pasa a ser una propiedad del problema.
recuperación bloqueada	Situación en la que la herramienta necesaria para reparar depende de lo que se ha roto. Es el fallo que ningún plan detecta sin ensayarlo.
conmutación parcial	El tráfico se traslada de región y los datos no. La infraestructura responde correctamente y el servicio incumple su objetivo.
SLI del camino asíncrono	Indicador de frescura del trabajo en segundo plano. Un SLO de la API no lo cubre: un consumidor detenido no afecta a ninguna petición HTTP.
costo por unidad de negocio	Única comparación accionable entre plataformas. Compara arquitecturas antes que proveedores, y por eso la mayor partida suele coincidir.

Modelo mental

Un proyecto de Google Cloud es la unidad práctica de API, cuota, IAM y facturación; la organización aporta la política heredable.

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 entregada es el resultado de once decisiones con su alternativa descartada. La cuarta columna es la que solo se puede escribir después de recorrer la parte:

Componente	Requisito	Alternativa descartada	Trampa que evita
Proyectos por servicio y entorno	Cuotas y superficie (049)	Tres proyectos por entorno	La cuota de región que bloquea una campaña
Cuenta de servicio dedicada por carga	Privilegio mínimo (050)	La cuenta por defecto	Ejecutar con Editor sin saberlo
Federación con condición de atributo	Sin claves (050)	Clave en el sistema de CI	Que cualquier repositorio obtenga credenciales
Una VPC global, compartida	Simplicidad y gobierno (051)	Una red por equipo	Cinco redes en la sombra sin revisar
Firewall dirigido por cuenta de servicio	Integridad del control (051)	Etiquetas de red	Abrir un puerto poniéndose una etiqueta
Acceso privado a Google en todas las subredes	Costo y aislamiento (051)	Salida por NAT	Pagar 108 USD/mes por tráfico que no debía salir
Cloud Run con concurrencia 80	Costo y densidad (055)	Concurrencia 1	Pagar seis veces por costumbre
Cloud SQL con reintento transitorio	Continuidad (054)	Confiar en la alta disponibilidad	Perder peticiones en cada conmutación
Pub/Sub con fallidos y permisos	Trabajo visible (056)	Fallidos solo configurados	Una red de seguridad que no existe
SLO con presupuesto de error	Alertas útiles (057)	Umbrales de recurso	340 avisos y 6 incidentes
Terraform con estado por unidad	Radio de impacto (059)	Un estado compartido	Destruir lo que comparte estado

Y cuatro decisiones tomadas en contra de la traducción literal desde Azure, que son las que un revisor cuestionará:

1. No se replicó el diseño de concentrador y radios.
   La VPC es global: los emparejamientos y las tablas de rutas sobraban (051).

2. No se agruparon servicios en pocos proyectos grandes.
   Las cuotas son por proyecto y los proyectos son baratos: la granularidad
   fina es una decisión de capacidad, no de orden (049).

3. No se usó una única cuenta de almacenamiento por entorno.
   Aquí la ubicación y la clase son propiedades del bucket y del objeto,
   no de una cuenta compartida (053).

4. No se copió la política de ciclo de vida de la parte 03.

```
El precio por operación sube al enfriar, y la política traducida
habría costado más de lo que ahorraba (053).
```

La cuarta es la que mejor resume el riesgo de la portabilidad mal entendida: un patrón correcto en una plataforma puede destruir valor en otra sin que nada falle.

2. La hipótesis de la clase 048, calificada

La clase 048 dejó escrito, para poder equivocarse de forma comprobable:

> 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.

Tres afirmaciones. Dos se cumplieron y una es falsa.

Afirmación 1 — el contrato se conserva. CIERTA, con dos matices.

```
contrato                                  ¿se conservó?
identidad de carga sin secretos            sí · cuenta de servicio adjunta
sujeto de federación acotado               sí · y volvió a fallar la 1.ª vez
frontera de aislamiento del entorno        sí · proyecto, pero BARATO
salida declarada con costo por GB          sí · Cloud NAT
acceso privado a servicios gestionados     sí · acceso privado a Google
entrega al menos una vez                   MATIZADO ↓
cola de fallidos con alerta a cero         sí · pero hay que darle permisos
idempotencia en el manejador               REFORZADO ↓
clave de partición irreversible            sí · tercera forma, silenciosa
prueba negativa por control                sí · doce ejecutadas
costo por unidad de negocio                sí
```

Los dos matices son mejoras del contrato, no excepciones:

```
entrega al menos una vez
  Pub/Sub ofrece entrega EXACTAMENTE UNA VEZ dentro de una región.
  El contrato se refina: lo que no existe en ninguna plataforma
  es el EFECTO exactamente una vez.

idempotencia en el manejador
  gana una razón nueva: la reproducción de mensajes (056).
  Ya no se justifica solo por los duplicados que ocurren,
  sino por los que se provocan a propósito al reparar un daño.
```

Afirmación 2 — las excepciones serán otras diez. CIERTA.

Ninguna de las diez de Azure se repitió, y aparecieron diez propias:

```
el proyecto es barato y cambia la granularidad          049
se pueden crear proyectos fuera de la organización      049
la facturación está fuera de la jerarquía y su
  exportación no es retroactiva                         049
la cuenta de servicio es identidad Y recurso            050
la VPC es global                                        051
el descuento por uso sostenido se aplica solo           052
la reparación automática destruye infraestructura sana  052
el precio por operación sube al enfriar la clase        053
concurrencia por instancia y CPU solo durante la petición 055
el archivo de estado contiene los secretos en claro     059
```

Afirmación 3 — el peor incidente será «funcionar por el camino equivocado». FALSA.

Y es el resultado más útil de la parte. En Google Cloud, los equivalentes de aquel fallo fueron todos ruidosos:

```
sin acceso privado a Google         la llamada falla       → 11 min hasta corregir
sin índice en Firestore             la consulta falla      → visible al desplegar
Cloud SQL sin ruta privada          la conexión falla      → visible al desplegar
```

Lo que sí produjo los incidentes más caros fue otra familia:

```
la cola de fallidos que no recibía nada por falta de permisos    (056)
  1.841 mensajes atascados durante tres semanas, sin una sola alerta,
  porque la ausencia de mensajes se interpretó como buena señal

la regla de ciclo de vida que costó 210 USD para ahorrar 5,22    (053)
  nada falló: cobró
```

```
la estimación con precios de lista, que sobrevaloró el gasto     (052)
  nada falló: la conclusión era falsa
```

Las tres tienen una forma común que la clase 048 no supo nombrar. La ley corregida:

> El fallo más caro no es el que ocurre por el camino equivocado. Es el de un mecanismo que parece estar haciendo algo y no lo está — y su caso más peligroso es aquel en el que la ausencia de señal se interpreta como buena noticia.

Eso incluye el camino privado silencioso de la clase 039, la cola de fallidos vacía de la 056, los registros apagados de la 045 y el control de seguridad revertido de la 046. Cuatro caras del mismo problema, en tres plataformas.

Y la consecuencia operativa, que es lo que se lleva a la parte 05: todo mecanismo de protección necesita una prueba que lo obligue a actuar. No basta con que no salten alarmas: hay que provocar la condición y ver la reacción. Una cola de fallidos sin un mensaje envenenado de prueba, una copia de seguridad sin una restauración y un antimalware sin un fichero de prueba son la misma afirmación sin evidencia.

3. Diez leyes ya observadas en tres plataformas

Un comportamiento que aparece tres veces con tres mecanismos distintos ha dejado de ser una peculiaridad del proveedor. Estas diez ya cumplen ese criterio, y son el activo que hace barata la cuarta plataforma:

```
 1. El permiso suma; lo que resta vive en OTRO sistema, con otro error.
    SCP · Azure Policy · políticas de organización y de denegación

 2. El gobierno guarda la puerta y no limpia la casa.
    Inventariar → corregir → imponer. Al revés, panel en verde con
    catorce recursos incumpliendo (046).

 3. Los puertos de traducción de salida se agotan, y el error apunta al destino.
    NAT gateway · puertos repartidos del balanceador · 64 por máquina

 4. El caudal de disco es el MÍNIMO de dos límites, y casi nunca es el que compraste.
    volumen/instancia · disco/máquina · GB del disco/vCPU

 5. La unidad de orden es también la unidad de serialización.
    FIFO · sesiones · particiones · claves de ordenación · claves primarias

 6. La entrega exactamente una vez se compra; el efecto se construye en el manejador.
    Y con reproducción, la idempotencia deja de ser opcional.

 7. La conmutación de una base de datos gestionada es funcionamiento normal.
    El cliente reintenta o pierde peticiones, con las alarmas en verde (048).

 8. Con versionado, borrar no libera nada hasta que una regla retire las versiones.
    Marcadores de borrado · versiones · versiones no actuales

 9. Una traza se corta en el salto que alguien escribió a mano.
    Y siempre es el interesante.

10. Leer un secreto en cada petición agota una cuota, y el error parece de la
    base de datos.  Key Vault · Secret Manager · gestores equivalentes
```

Diez leyes, cada una observada de forma independiente en tres implementaciones que no comparten código. Y una observación sobre su naturaleza que merece decirse: ninguna de las diez es un fallo de las plataformas. Todas son consecuencias de problemas reales —traducción de direcciones, orden distribuido, límites de E/S, entrega en red no fiable— que cualquier implementación tiene que resolver de alguna manera.

Eso convierte la lista en algo más útil que un resumen: es un cuestionario para la plataforma siguiente. Ante un proveedor nuevo, las diez preguntas ya están escritas y solo hay que averiguar cómo se llaman aquí las respuestas y dónde está la excepción.

Y dos afirmaciones que no han alcanzado ese estatus y conviene no dar por buenas:

```
"el archivo de estado es un problema"     depende de la herramienta, no del proveedor
"lo global es mejor que lo regional"      la VPC global simplifica y la
                                          conmutación parcial es un riesgo nuevo (abajo)
```

4. Tres fallos provocados y lo que enseñó cada uno

Los tres se eligieron para atacar supuestos distintos, y los tres dejaron un hallazgo que ninguna revisión de configuración habría dado.

Fallo 1 — pérdida de una zona. Se retiran las instancias de europe-west1-b y se fuerza la conmutación de Cloud SQL.

```
detección por alerta de consumo de presupuesto   1 min 10 s
errores durante la conmutación de la base           0   ← el reintento estaba (048)
p95 durante el episodio                          142 ms
recuperación completa                            2 min 05 s
```

Cero errores en la conmutación. Es la segunda vez que un problema conocido no cuesta nada, porque estaba escrito en el contrato.

Y el hallazgo, que es nuevo. Al perder capacidad en la región europea, el balanceador global desbordó tráfico a us-east1, tal como se había diseñado. Pero la base de datos seguía en Europa:

```
peticiones servidas desde us-east1              18.400
p99 de esas peticiones                          612 ms   (SLO: 500 ms)
causa                                           latencia transatlántica
                                                hacia la base de datos primaria
```

La conmutación funcionó en la capa de red y rompió el objetivo en la capa de datos. El desbordamiento por capacidad de la clase 052 mueve el cómputo y no mueve los datos, y ninguna de las dos piezas está mal configurada.

```
correcciones
1. réplica de lectura en us-east1, y lecturas dirigidas a la local
2. las escrituras siguen yendo a Europa: se acepta y se documenta
3. el escalador de capacidad de us-east1 se limita a 0,3:
   absorbe un pico, no sustituye a la región
4. el simulacro pasa a medir el p99 POR REGIÓN DE ORIGEN, no agregado
```

La cuarta es la que habría detectado el problema sin romper nada: el percentil agregado estaba dentro del objetivo porque solo el 12 % del tráfico venía de la otra región.

Fallo 2 — revocación de una clave del cliente. Se retira el permiso de la identidad sobre k-facturas.

```
servicios afectados esperados     1 (subida de facturas)
servicios afectados reales        1                       ← la separación funcionó
recuperación tras restaurar       4 min
```

La separación por propósito de la clase 058, que venía del simulacro de la clase 048, evitó el radio de impacto ampliado. Pero apareció otra cosa:

El hallazgo: la recuperación estaba bloqueada por lo que se había roto. El bucket del estado de Terraform usaba la misma clave, así que el intento de restaurar el permiso mediante la canalización falló:

```
Error: googleapi: Error 403: The caller does not have permission
  to use kms key … for bucket cls-tfstate-prod
```

La herramienta que arregla la plataforma dependía de la clave que se acababa de revocar. La recuperación se hizo a mano, con una identidad de emergencia, y tardó cuatro minutos porque alguien conocía el camino — no porque estuviera escrito.

```
correcciones
1. el bucket de estado usa una clave PROPIA, en otro anillo
2. procedimiento de recuperación con identidad de emergencia, escrito y ensayado
3. regla general añadida a la línea base:
   ninguna herramienta de recuperación puede depender de lo que recupera
```

Fallo 3 — consumidor detenido en silencio. El fallo que este programa ha provocado en las tres plataformas.

```
detección                      no hubo    4 min 20 s   3 min 40 s
mensajes acumulados            14         680          512
```

El hallazgo: la alerta de acumulación se disparó y la alerta de presupuesto de error no, exactamente como la clase 057 anticipó — un consumidor detenido no afecta a ninguna petición HTTP, así que el SLI de la API no se mueve. Al revisarlo, apareció el hueco real:

```
SLO definidos              1 · disponibilidad y latencia de la API
SLO del camino asíncrono   ninguno
```

La plataforma tenía un objetivo para lo que el usuario ve al pulsar y ninguno para lo que espera recibir después.

```
corrección
SLI de frescura: proporción de notificaciones enviadas en menos de 5 min
SLO 99 % en 28 días, con alerta por velocidad de consumo
→ el mismo mecanismo de la clase 057, aplicado a un camino que no es HTTP
```

Los tres hallazgos comparten forma con los de la clase 048 y merece señalarlo: la infraestructura aguantó los tres fallos, y en los tres había algo que ninguna revisión de configuración podía mostrar — una latencia entre capas, una dependencia circular de recuperación y un objetivo que nunca se definió.

5. La entrega, las tres plataformas y la pregunta que abre la parte 05

La entrega, sin conocimiento tácito.

```
infraestructura     Terraform, 4 estados, plan revisado y validado por políticas
línea base          24 afirmaciones, cada una con su prueba negativa
verificar.sh        ejecuta las 24 y devuelve código de salida
políticas           de organización, y validación sobre el plan
ADR                 11 decisiones con su alternativa descartada
riesgos residuales  5, con responsable y condición de revisión
SLO                 2: camino síncrono y camino asíncrono
consultas guardadas 4, enlazadas desde el manual de operación
procedimientos      3 fallos ensayados, con verificación posterior
línea base medida   rendimiento, costo y costo por pedido
```

Los cinco riesgos residuales:

```
1. las escrituras solo ocurren en Europa: la conmutación a us-east1 es parcial
2. Secret Manager no rota por sí solo: la función de rotación es código propio
3. el gestor de claves externo se descartó; se revisará si hay obligación contractual
4. dos cuentas de emergencia con acceso permanente, documentadas
5. los registros de acceso a datos solo están activos en 4 proyectos
```

La comparación de las tres plataformas, hecha de forma que signifique algo.

```
rps sostenidos                          987,4        984,1        985,7
p50                                    38,4 ms      41,2 ms      39,8 ms
p95                                    91,2 ms      96,8 ms      94,1 ms
p99                                   184,7 ms     203,5 ms     197,2 ms
p99/p50                                  4,8×         4,9×         5,0×
errores                                     0            0            0
```

No hay diferencia significativa de rendimiento a esta escala. Es la segunda vez que se mide y la conclusión se repite, así que ya se puede afirmar: a este volumen, el rendimiento no es un criterio de elección entre proveedores. Lo que decide son capacidades, costo por unidad, requisitos de residencia y competencias del equipo.

Y el costo, desglosado hasta que cada partida tenga nombre:

```
entrada y borde                          48       185        62
cómputo de aplicación                   164       344        84
base de datos relacional                210       390       390
caché                                    46        55        58
mensajería                               31        22        29
almacenamiento y salida                  42        48        34
salida a internet (NAT)                  58        69        68
telemetría                               52       110        88
seguridad y gobierno                     37        96        48
                                        ■■■■■     ■■■■■     ■■■■■
total                                   688       940       861
costo por pedido                   0,000702  0,000959  0,000879
```

Dos lecturas, y la segunda es la importante.

La primera: Google Cloud queda entre las dos, un 25 % por encima de AWS y un 8 % por debajo de Azure. Las diferencias tienen nombre: el cómputo de aplicación es menos de la mitad que en Azure por la concurrencia de la clase 055, y el borde cuesta un tercio que la pasarela con WAF.

La segunda: la mayor partida es la misma en las tres, y en dos de ellas es idéntica. La base de datos relacional aprovisionada es entre el 30 % y el 45 % del total en todos los casos, porque es la misma decisión de arquitectura tomada tres veces. Y de ahí la conclusión que cierra las cuatro partes:

> Comparar proveedores compara tu arquitectura contigo mismo. La partida que domina la factura la eligió el equipo, no el proveedor.

Cambiar de nube habría movido el 25 %. Cambiar la decisión de datos —a un motor que escala a cero, a un modelo que no necesita una instancia encendida 720 horas al mes— movería más, y no depende de ningún proveedor.

Y la pregunta que abre la parte 05.

En las tres plataformas, la capa de aplicación acabó ejecutando contenedores: Cloud Run aquí, Container Apps allí, servicios gestionados de contenedor en AWS. Las tres convergieron sin que nadie lo planificara, y eso reorienta la pregunta:

> Si el contenedor es la unidad de despliegue en las tres nubes, ¿qué garantiza exactamente ese contrato y dónde deja de garantizar? ¿Qué parte de lo aprendido en estas cuatro partes es del proveedor, y qué parte es de la plataforma que ejecuta el contenedor?

La hipótesis que se escribe ahora, para poder equivocarse otra vez de forma comprobable:

> El contrato del contenedor —imagen, proceso, puerto, variables, señales— será portable de verdad, y las fugas estarán en los bordes: almacenamiento persistente, red, identidad y todo lo que ocurre antes del primer proceso y después de la señal de terminación. Y volverá a aparecer al menos una de las diez leyes de esta clase, con un mecanismo nuevo.

La parte 05 la califica.

Ejemplo trabajado

Entrega del capstone de la parte 04, con las cifras que se llevan a la parte 05.

Verificación completa. Las 24 afirmaciones de la línea base:

```
$ ./verificar.sh
✓ federación deniega desde otro repositorio      permiso denegado
✓ ninguna clave de cuenta de servicio            0 encontradas
✓ creación de claves bloqueada                   violación de restricción
✓ bucket no público                              412 · publicAccessPrevention
✓ recuperación de bucket                         50 objetos restaurados
✓ facturado converge con lo visible              430 GiB / 412 GiB
✓ URL firmada sin claves, 15 min                 caduca correctamente
✓ subred sin salida a internet                   tiempo agotado a los 5 s
✓ firewall dirigido por identidad                UNREACHABLE por 22
✓ acceso privado a Google activo                 resuelve interno
✓ Cloud SQL sin IP pública                       tiempo agotado
✓ reintento de errores transitorios              conmutación sin errores
✓ punto caliente de Spanner                      6.400 escrituras/s
✓ servicios internos no públicos                 403 sin testigo
✓ concurrencia verificada bajo carga             0 respuestas cruzadas
✓ cola de fallidos recibe                        mensaje envenenado llega
✓ acumulación en cola alerta                     aviso en 3 min 40 s
✓ auditoría de acceso a datos activa             entrada localizada
✓ traza completa entre servicios                 6 de 6 tramos
✓ destrucción de versión de clave                restaurada en la ventana
✓ rotación de secreto con tráfico                0 errores
✓ estado de Terraform con acceso mínimo          2 principales
✓ plan aplicado idéntico al revisado             tfplan verificado
✓ pedido con comilla simple y punto y coma       201
24/24 correctas
```

Línea base medida:

```
rps 985,7 · p50 39,8 ms · p95 94,1 ms · p99 197,2 ms · errores 0
L = 985,7 × 0,0398 = 39,2 de 50 solicitados → medida válida
p99/p50 = 5,0×
costo 861 USD/mes · 0,000879 USD/pedido
```

Los tres fallos, con lo aprendido en cada uno:

pérdida de zona	1 min 10 s	142 ms	0 peticiones 18.400 fuera del SLO	la conmutación mueve el cómputo y no los datos
revocación de clave	inmediata	—	1 servicio (lo previsto)	la herramienta de recuperación no puede depender de lo que recupera

```
consumidor detenido          3 min 40 s   sin cambio   512 mensajes    faltaba un SLO
                                                       retrasados      para el camino
                                                                       asíncrono
```

El hallazgo que justificó el capstone. Durante la pérdida de zona, todos los paneles agregados estuvieron dentro del objetivo:

```
p99 agregado durante el episodio     213 ms   (SLO 500 ms)      ✓ verde
presupuesto de error consumido        0,7 %                    ✓ verde
```

Y al desglosar por región de origen:

```
SELECT region_origen,
       APPROX_QUANTILES(latencia_ms, 100)[OFFSET(99)] AS p99,
       COUNT(*) AS peticiones
FROM peticiones
WHERE ts BETWEEN '2026-07-30 14:02' AND '2026-07-30 14:07'
GROUP BY region_origen
```

```
europe-west1   p99  148 ms   134.900 peticiones
us-east1       p99  612 ms    18.400 peticiones      ← fuera del SLO
```

Dieciocho mil cuatrocientas peticiones fuera del objetivo, invisibles en el agregado porque eran el 12 % del tráfico. La infraestructura hizo exactamente lo que se le pidió: desbordar a la región con capacidad. Nadie había comprobado qué latencia tendría esa región contra una base de datos que seguía en Europa.

```
síntoma observable   ninguno: percentil agregado en verde,
                     presupuesto de error casi intacto
consecuencia real    12 % de los usuarios con latencia 4× durante 5 minutos
causa                el desbordamiento mueve el cómputo, no los datos
qué lo destapó       desglosar por región, no el simulacro en sí
```

Repetido el simulacro con las cuatro correcciones:

```
p99 desde us-east1                       612 ms         171 ms
peticiones fuera del SLO                 18.400              0
escalador de capacidad de us-east1          1,0            0,3
SLO medidos por región de origen            no              sí
```

Se entrega a la parte 05 con:

```
diez leyes observadas en tres plataformas independientes
contrato portable de once filas, dos de ellas refinadas
la ley corregida sobre la familia de fallos más cara
24 afirmaciones y su guion de verificación
11 decisiones con alternativa descartada
5 riesgos residuales con responsable
comparación de costo por pedido en tres nubes, desglosada
```

Y la hipótesis escrita para la parte 05, que ya se puede calificar dentro de doce clases:

> El contrato del contenedor será portable de verdad y las fugas estarán en los bordes —almacenamiento, red, identidad, arranque y terminación—. Volverá a aparecer al menos una de las diez leyes, con un mecanismo nuevo.

La lección que esta parte deja al programa: la hipótesis de la clase 048 acertó en el contrato y en las excepciones, y falló en predecir qué familia de fallos costaría más. Corregir esa predicción con evidencia vale más que haberla acertado: la familia real —un mecanismo que parece estar protegiendo y no lo está— explica el fallo más caro de las tres partes, y su antídoto es una sola regla que ahora está escrita en la línea base: todo mecanismo de protección necesita una prueba que lo obligue a actuar.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-04-gcp-core-platform/060-proyecto-aplicacion-de-tres-capas-en-google-cloud/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-gcp 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-gcp 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
El percentil agregado está dentro del objetivo y hay usuarios claramente afectados	Una minoría del tráfico, desde otra región u origen, queda diluida en el agregado	Mide y alerta por región de origen y por segmento, no solo agregado; el simulacro debe desglosarlo.
El tráfico conmuta de región y el servicio incumple su objetivo igualmente	El desbordamiento por capacidad mueve el cómputo y no los datos	Réplica de lectura local, escalador de capacidad limitado en la región secundaria y decisión escrita sobre dónde se escribe.
La recuperación de un incidente falla porque la herramienta depende de lo que se rompió	El estado o la canalización usan el mismo recurso o la misma clave que se ha revocado	Aísla lo que recupera de lo que se recupera, y ensaya el procedimiento con una identidad de emergencia.
Un consumidor detenido no dispara ninguna alerta de objetivo de servicio	El SLI mide peticiones HTTP y el camino asíncrono no tiene objetivo propio	Define un SLI de frescura para el trabajo en segundo plano y aplícale el mismo mecanismo de presupuesto de error.
Una política correcta en otra plataforma cuesta dinero en esta	El modelo de precios cambia el punto de equilibrio, aunque el patrón sea el mismo	Recalcula la aritmética con los precios de la plataforma antes de trasladar cualquier política de ciclo de vida o de escalado.
Se elige proveedor por precio y la factura no cambia lo esperado	La mayor partida es una decisión de arquitectura idéntica en las tres plataformas	Desglosa hasta que cada partida tenga nombre; la decisión de datos suele mover más que la de proveedor.

■ 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. ¿Cuáles de las tres afirmaciones de la hipótesis de la clase 048 se cumplieron, y qué dice la ley corregida?
2. Enumera cuatro de las diez leyes observadas en tres plataformas y el mecanismo distinto que tuvo cada una.
3. Durante la pérdida de zona todo estaba en verde y 18.400 peticiones incumplían el SLO. ¿Por qué, y qué medición lo habría mostrado?
4. ¿Qué significa que la herramienta de recuperación dependa de lo que recupera, y cómo se evita?
5. ¿Por qué la mayor partida de costo coincide en las tres plataformas y qué implica eso al elegir proveedor?

Referencias

• Google Cloud (2025). Architecture Framework — los pilares como rejilla de revisión de la entrega. <https://cloud.google.com/architecture/framework>

• Google Cloud (2025). Patterns for scalable and resilient apps — desbordamiento entre regiones y ubicación de los datos. <https://cloud.google.com/architecture/scalable-and-resilient-apps>

• Google Cloud (2025). Disaster recovery planning guide — RTO, RPO y ensayo de procedimientos. <https://cloud.google.com/architecture/dr-scenarios-planning-guide>

• Google (2018). The Site Reliability Workbook, cap. 2 — SLI, SLO y alertas por consumo de presupuesto de error. <https://sre.google/workbook/implementing-slos/>

• Google Cloud (2025). Landing zone design in Google Cloud — jerarquía, red e identidad como línea base declarada. <https://cloud.google.com/architecture/landing-zones>

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

Descargar: Parte 04 en PDF · Recorrido de Google Cloud en PDF · Manual integral

Anterior	Índice	Siguiente
← 059 · Terraform y despliegues reproducibles en GCP	Parte 04 · Programa	061 · Imágenes, capas, registros y estándar OCI →

Evaluación — 060 Proyecto: aplicación de tres capas en Google Cloud

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-gcp 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.