Multi-Cloud Engineering Program

Parte 01 — Principios, estrategia y adopción cloud

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	01 de 23
Clases	013–024
Evaluaciones	12
Horas estimadas	52

Índice

Parte 01 — Principios, estrategia y adopción cloud

← Parte 00 · Índice completo · Parte 02 →

Descargar: Esta parte en PDF · Manual integral

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

Decidir cuándo, por qué y cómo adoptar nube con criterios técnicos y económicos.

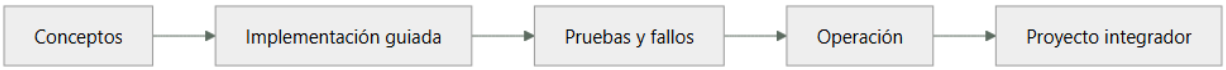
Resultados de la parte

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

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

Secuencia



Clases

ID	Clase	Laboratorio	Horas
013	Definición NIST y características esenciales de cloud	foundation	4
014	Regiones, zonas de disponibilidad, puntos de presencia y edge	architecture	4
015	IaaS, PaaS, SaaS, CaaS y FaaS	decision	4
016	Elasticidad, escalabilidad, disponibilidad y resiliencia	reliability	4
017	Tenancy, cuentas, suscripciones, proyectos y jerarquías	governance	4
018	Identidad, roles, políticas y federación	iam	4
019	Modelo de responsabilidad compartida por servicio	security	4
020	TCO, costos variables, unit economics y FinOps	finops	4
021	Well-Architected y atributos de calidad	architecture	4
022	Cloud Adoption Framework y modelo operativo	governance	4
023	Descubrimiento y clasificación de workloads	migration	4
024	Proyecto: decisión de migración sustentada con ADR	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

- Cloud Strategy — Gregor Hohpe.
- Cloud FinOps — Storment y Fuller.
- Accelerate — Forsgren, Humble y Kim.

013 — Definición NIST y características esenciales de cloud

Parte: 01 — Principios, estrategia y adopción cloud Nivel: inicial-intermedio · Horas estimadas: 4 Laboratorio: foundation · Estado: EXECUTABLECORE

Propósito

Convertir la definición de NIST en una herramienta de decisión y no en un dato de examen. Cada una de las cinco características esenciales genera una consecuencia arquitectónica concreta, y la quinta —el servicio medido— es la que convierte cada decisión técnica del resto del programa en una decisión económica.

Resultados de aprendizaje

Al finalizar podrás:

1. Auditar un servicio contra las cinco características y determinar si es cloud o hosting con nombre nuevo.
2. Derivar de cada característica al menos una consecuencia de diseño verificable.
3. Situar un despliegue en el modelo correcto —público, privado, híbrido o comunitario— por su frontera de control, no por su ubicación.
4. Explicar por qué la agrupación de recursos implica riesgo de vecino ruidoso y qué controles lo acotan.
5. Justificar con la definición por qué «nube privada» y «centro de datos virtualizado» no son sinónimos.

Conceptos centrales

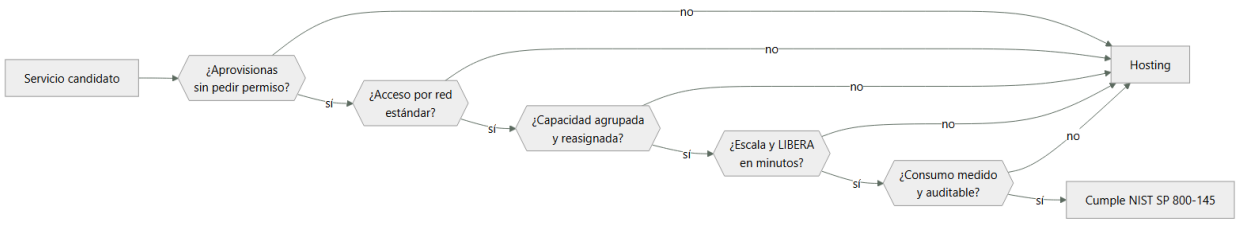
Concepto	Comprensión verificable
autoservicio bajo demanda	Capacidad de aprovisionar sin interacción humana con el proveedor. Su prueba es operativa: si hace falta un ticket o una llamada, no se cumple, y el tiempo de aprovisionamiento deja de ser un parámetro de arquitectura.
agrupación de recursos	Modelo en el que el proveedor reasigna capacidad física entre clientes de forma opaca. Es lo que abarata el servicio y lo que introduce el riesgo de interferencia entre inquilinos.
elasticidad rápida	Aprovisionar y liberar en minutos siguiendo la demanda, en ambas direcciones. Sin la dirección de bajada no hay elasticidad: hay aprovisionamiento rápido, que no genera ahorro.
servicio medido	Medición automática y transparente del consumo, con capacidad de auditarla. Es la característica que convierte la arquitectura en una función de coste.
modelo de despliegue	Quién controla la infraestructura y para quién opera: público, privado, comunitario o híbrido. Se define por la frontera de control y de inquilinos, no por dónde están los servidores.

Modelo mental

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

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

■ Flujo de razonamiento



Desarrollo

1. Cinco características, cinco consecuencias de diseño

La definición del NIST SP 800-145 (Mell y Grance, 2011) no es descriptiva: es una lista de comprobación con consecuencias. Un servicio debe cumplir las cinco; fallar una lo saca de la categoría.

Característica	Prueba operativa	Consecuencia de diseño
Autoservicio bajo demanda	¿Aprovisionas sin ticket?	El tiempo de aprovisionamiento pasa de semanas a minutos, así que la capacidad deja de ser una decisión anual
Acceso amplio por red	¿Protocolos estándar, cualquier cliente?	La red es la única frontera: no hay «estar dentro»
Agrupación de recursos	¿El proveedor reasigna sin avisarte?	Tu rendimiento depende de vecinos que no ves
Elasticidad rápida	¿Escala y libera en minutos?	La capacidad ociosa se vuelve coste evitable
Servicio medido	¿Consumo auditable por unidad?	Cada bucle mal escrito aparece en la factura

La columna derecha es la que importa. La cuarta fila tiene un matiz que decide proyectos: elasticidad exige las dos direcciones. Un sistema que escala en 2 minutos pero tarda 3 días en liberar capacidad —porque nadie se atreve a apagar— no produce ahorro. Es aprovisionamiento rápido, no elasticidad, y su economía es la de un centro de datos.

La quinta fila es la que reorganiza la ingeniería: en un centro de datos propio, un algoritmo ineficiente consume capacidad ya pagada; en cloud, consume dinero medible atribuible a un equipo. Ese cambio de incentivo es el origen de FinOps, y por eso la clase 011 estableció la unidad de coste antes de entrar en esta parte.

2. El modelo de despliegue lo define el control, no la ubicación

La confusión más extendida de esta parte: creer que «privado» significa «en mis instalaciones». NIST define los cuatro modelos por para quién opera la infraestructura, no por dónde está:

Modelo	Inquilinos	Puede estar
Público	Cualquiera	Solo en el proveedor
Privado	Una sola organización	En tus instalaciones o en un tercero
Comunitario	Varias organizaciones con requisitos compartidos	En cualquiera de ellas o en un tercero
Híbrido	Combinación con portabilidad de datos entre ellas	Ambos

De ahí dos consecuencias que se discuten mal en los comités:

1. Una nube privada alojada en un tercero sigue siendo privada. El criterio es la exclusividad de inquilino, no la propiedad del edificio.
2. Un centro de datos virtualizado no es una nube privada. Le suele faltar autoservicio (hay que pedir la máquina), elasticidad (no se libera) y medición (no se factura por consumo). Cumple una de cinco.

Y el híbrido tiene un requisito que casi nadie aplica: NIST exige portabilidad de datos y aplicaciones entre las partes. Tener servidores propios y además una cuenta en un proveedor no es híbrido: es tener dos cosas separadas. Sin la portabilidad, no hay modelo híbrido, hay coexistencia. Esa distinción reaparecerá con consecuencias de coste en la parte 13.

3. Agrupación de recursos: el precio tiene una contrapartida

La agrupación es lo que hace barata la nube: el proveedor amortiza el mismo hardware entre muchos clientes con picos que no coinciden. La contrapartida es el vecino ruidoso: tu rendimiento depende de cargas que no ves ni controlas.

Se manifiesta en tres recursos, con síntomas distintos:

Recurso	Síntoma	Mitigación
CPU	Latencia irregular sin cambio de carga propia	Instancias con núcleos dedicados
E/S de disco	IOPS por debajo de lo esperado a ratos	Volúmenes con IOPS aprovisionadas
Red	Throughput variable entre instancias	Instancias con ancho de banda garantizado

La variabilidad, no la media, es lo que hay que medir. Un proveedor puede cumplir su media anunciada y aun así producir un p99 inaceptable, porque la interferencia es esporádica. Es el mismo argumento de la clase 012 sobre por qué la media miente.

El caso especial es el crédito de CPU de las instancias ráfaga: acumulan crédito mientras están ociosas y lo gastan al trabajar. Cuando se agota, el rendimiento cae a una fracción del nominal —por ejemplo, al 20 %— sin que ninguna métrica de la aplicación explique el desplome. Diagnosticarlo exige mirar el saldo de créditos, que es una métrica del proveedor y no del sistema operativo. Es la primera vez en el programa en que una métrica externa a la máquina es imprescindible para explicar su comportamiento.

4. Auditar un servicio con la definición

La utilidad práctica de NIST es cortar discusiones de márketing. Tres casos frecuentes, auditados:

«Nuestro hosting gestionado es cloud privado.»

Característica	¿Cumple?
Autoservicio	No: hay que abrir ticket
Acceso por red	Sí
Agrupación	Parcial
Elasticidad	No: alta en horas, baja en días
Medición	No: cuota fija mensual

Una de cinco. Es hosting. La consecuencia no es semántica: si compras esperando elasticidad, el ahorro previsto no llegará y el caso de negocio era falso desde el principio.

«Somos híbridos porque tenemos servidores y una cuenta en un proveedor.» Falta la portabilidad de datos y aplicaciones entre ambos. Sin ella no hay conmutación posible, así que tampoco hay la continuidad que se estaba justificando.

«Un servicio SaaS con precio por asiento no es cloud porque no se mide por consumo.» Falso: el asiento es la unidad de medida, y es auditable. NIST exige medición apropiada al tipo de servicio, no facturación por segundo.

El patrón: la definición sirve para verificar que la propiedad por la que estás pagando existe de verdad, no para clasificar por gusto.

5. Qué no dice NIST, y por qué importa

Tan útil como lo que define es lo que deja fuera, porque marca dónde el criterio debe ponerlo el arquitecto:

- No dice nada de seguridad. Un servicio puede cumplir las cinco características y ser inseguro. La seguridad se reparte con el modelo de responsabilidad compartida de la clase 019.
- No dice nada de coste. Cumplir la definición no implica ser barato; implica ser medible. Que salga a cuenta depende de la utilización, como se calculó en la clase 011.
- No dice nada de fiabilidad. No hay ninguna característica sobre disponibilidad. Esa la fija el SLA, que es un documento distinto con exclusiones propias.
- No dice nada de portabilidad, salvo en el modelo híbrido. Cumplir NIST es compatible con un bloqueo total de proveedor.

La última es la más relevante para un programa multi-cloud: la definición no protege del bloqueo de proveedor. Un servicio perfectamente conforme puede usar una API propietaria sin equivalente en otro sitio. La portabilidad es una

decisión de arquitectura con coste propio —se estudiará en las partes 12 y 13—, no una propiedad que venga incluida.

Y el documento tiene 14 años. Serverless, contenedores gestionados y funciones no existían como categorías cuando se escribió; encajan sin problema en IaaS/PaaS, pero la frontera entre modelos es hoy más difusa de lo que sugiere la clasificación en tres niveles.

Ejemplo trabajado

El comité de CloudShop evalúa dos ofertas para sacar su plataforma del centro de datos actual, y las dos se presentan como «nube privada». Se auditan contra las cinco características antes de mirar el precio.

Oferta A — proveedor local, «cloud privado gestionado».

```
autoservicio     ticket con SLA de 4 h laborables       ✗
acceso por red   VPN y API REST propia                  ✓
agrupación       hardware dedicado por cliente          ✗ (no hay agrupación)
elasticidad      alta en 4 h, baja con preaviso de 30 d ✗
medición         cuota fija mensual + extras            ✗
```

Una de cinco. Es alojamiento dedicado. No es que sea mala opción: es que el caso de negocio presentado se apoyaba en ahorro por elasticidad, y esta oferta no puede producirlo por construcción.

Oferta B — hiperescalar, cuenta dedicada con conectividad privada.

```
autoservicio     API y consola, sin intervención        ✓
acceso por red   protocolos estándar                    ✓
agrupación       multi-inquilino con aislamiento lógico ✓
elasticidad      minutos en ambas direcciones           ✓
medición         por segundo, con etiquetas y export    ✓
```

Cinco de cinco. Es nube pública con conectividad privada, no nube privada: comparte hardware con otros inquilinos. La distinción importa porque el requisito regulatorio del cliente exigía «aislamiento de inquilino» y aquí hay agrupación.

Se cuantifica el impacto de la característica que falta en A, con los datos de utilización reales:

```
capacidad de pico             100 unidades, 3 h/día
capacidad de base              20 unidades, 21 h/día
utilización media = (100×3 + 20×21)/(100×24) = 30 %

Oferta A (sin elasticidad): se paga el pico las 24 h
  100 × 24 × 30 × 0,85 USD                 = 61.200 USD/mes
Oferta B (con elasticidad): se paga el consumo
  (100×3 + 20×21) × 30 × 1,00 USD          = 21.600 USD/mes
```

39.600 USD/mes de diferencia, y no viene del precio unitario —que es peor en B— sino de la característica 4. A cobra un 15 % menos por unidad y cuesta casi el triple, porque cobra por capacidad reservada.

Decisión y límite explícito: se elige B, y el requisito de aislamiento de inquilino se resuelve con instancias de tenencia dedicada solo para las cargas que lo exigen —un 12 % del total—, no para toda la plataforma. Riesgo residual declarado: la tenencia dedicada elimina la agrupación en esa fracción, así que ahí no habrá ahorro por elasticidad y su coste unitario será el de A.

La lección del ejercicio: las cinco características no son un dato de examen, son las cinco preguntas que revelan si el caso de negocio se sostiene.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-01-cloud-principles-strategy-adoption/013-definicion-nist-y-caracteristicas-esenciales-de-cloud/lab.py
```

El laboratorio selecciona el motor de práctica foundation 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-nist 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 mapa con fronteras, entradas, salidas y supuestos. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye matriz-nist 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 firma un contrato esperando ahorro por elasticidad y la factura no baja	El servicio escala hacia arriba pero no libera capacidad; falla la característica 4	Exige la prueba en ambas direcciones antes de firmar: aprovisionar y liberar, con tiempos medidos.
Se llama nube privada a un centro de datos virtualizado	Se confundió virtualización con las cinco características	Audita las cinco; virtualizar cumple una y el caso de negocio suele apoyarse en las otras cuatro.
Se declara arquitectura híbrida sin poder conmutar entre las partes	NIST exige portabilidad de datos y aplicaciones, y solo había coexistencia	Si no hay portabilidad demostrada, no cuentes con la continuidad que justificaba el modelo.
El rendimiento cae sin que ninguna métrica del sistema operativo lo explique	Agotamiento de créditos de CPU en una instancia ráfaga: es una métrica del proveedor	Vigila el saldo de créditos junto a las métricas del sistema; la interferencia entre inquilinos no se ve desde dentro.
Se asume que cumplir NIST implica portabilidad entre proveedores	La definición no dice nada de portabilidad salvo en el modelo híbrido	Trata la portabilidad como decisión de arquitectura con coste propio, no como propiedad incluida.

■ 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. Un servicio escala en 2 minutos y libera capacidad en 3 días. ¿Cumple la característica de elasticidad? ¿Qué consecuencia económica tiene?
2. ¿Puede una nube privada estar alojada en un tercero? ¿Qué criterio decide el modelo de despliegue?
3. ¿Qué le falta a un centro de datos virtualizado para cumplir la definición, y cuántas de las cinco cumple?
4. Un SaaS cobra por asiento y no por segundo. ¿Incumple la característica de servicio medido? Justifica.

5. Nombra dos propiedades que NIST no garantiza y que suelen darse por supuestas al elegir proveedor.

Referencias

- Mell, P. y Grance, T. (2011). The NIST Definition of Cloud Computing, SP 800-145 — las cinco características, tres modelos de servicio y cuatro de despliegue. <https://doi.org/10.6028/NIST.SP.800-145>
- Liu, F. et al. (2011). NIST Cloud Computing Reference Architecture, SP 500-292 — actores, roles y sus relaciones. <https://doi.org/10.6028/NIST.SP.500-292>
- Hohpe, G. (2020). Cloud Strategy: A Decision-Based Approach — por qué la definición sirve para decidir y no para clasificar.
- Armbrust, M. et al. (2010). A View of Cloud Computing. Communications of the ACM 53(4), 50-58 — obstáculos y oportunidades, incluida la variabilidad de rendimiento. <https://doi.org/10.1145/1721654.1721672>
- Barroso, L., Hölzle, U. y Ranganathan, P. (2018). The Datacenter as a Computer, 3.ª ed., cap. 7 — economía de la agrupación de recursos. <https://doi.org/10.2200/S00874ED3V01Y201809CAC046>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 01 en PDF · Manual integral

Anterior	Índice	Siguiente
← 012 · Proyecto: servicio local reproducible y observable	Parte 01 · Programa	014 · Regiones, zonas de disponibilidad, puntos de presencia y edge →

Evaluación — 013 Definición NIST y características esenciales de 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 matriz-nist 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.

014 — Regiones, zonas de disponibilidad, puntos de presencia y edge

Parte: 01 — Principios, estrategia y adopción cloud Nivel: inicial-intermedio · Horas estimadas: 4 Laboratorio: architecture · Estado: EXECUTABLECORE

Propósito

Entender la geografía de una nube como una jerarquía de dominios de fallo con latencias y precios distintos entre niveles. Colocar un componente en el nivel equivocado produce o bien una factura de transferencia inesperada, o bien un sistema que cae entero cuando falla un edificio. Esta clase fija el vocabulario y la aritmética que usarán las partes 13, 16 y 17.

Resultados de aprendizaje

Al finalizar podrás:

1. Distinguir región, zona de disponibilidad y punto de presencia por su dominio de fallo, su latencia y su precio de transferencia.
2. Calcular la disponibilidad compuesta de un despliegue en una, dos y tres zonas.
3. Anticipar el coste de transferencia de un diseño antes de desplegarlo, sabiendo qué cruces se cobran.
4. Justificar cuándo el edge aporta y cuándo solo añade complejidad.
5. Reconocer que las zonas son etiquetas por cuenta y qué implica al comparar despliegues entre cuentas.

Conceptos centrales

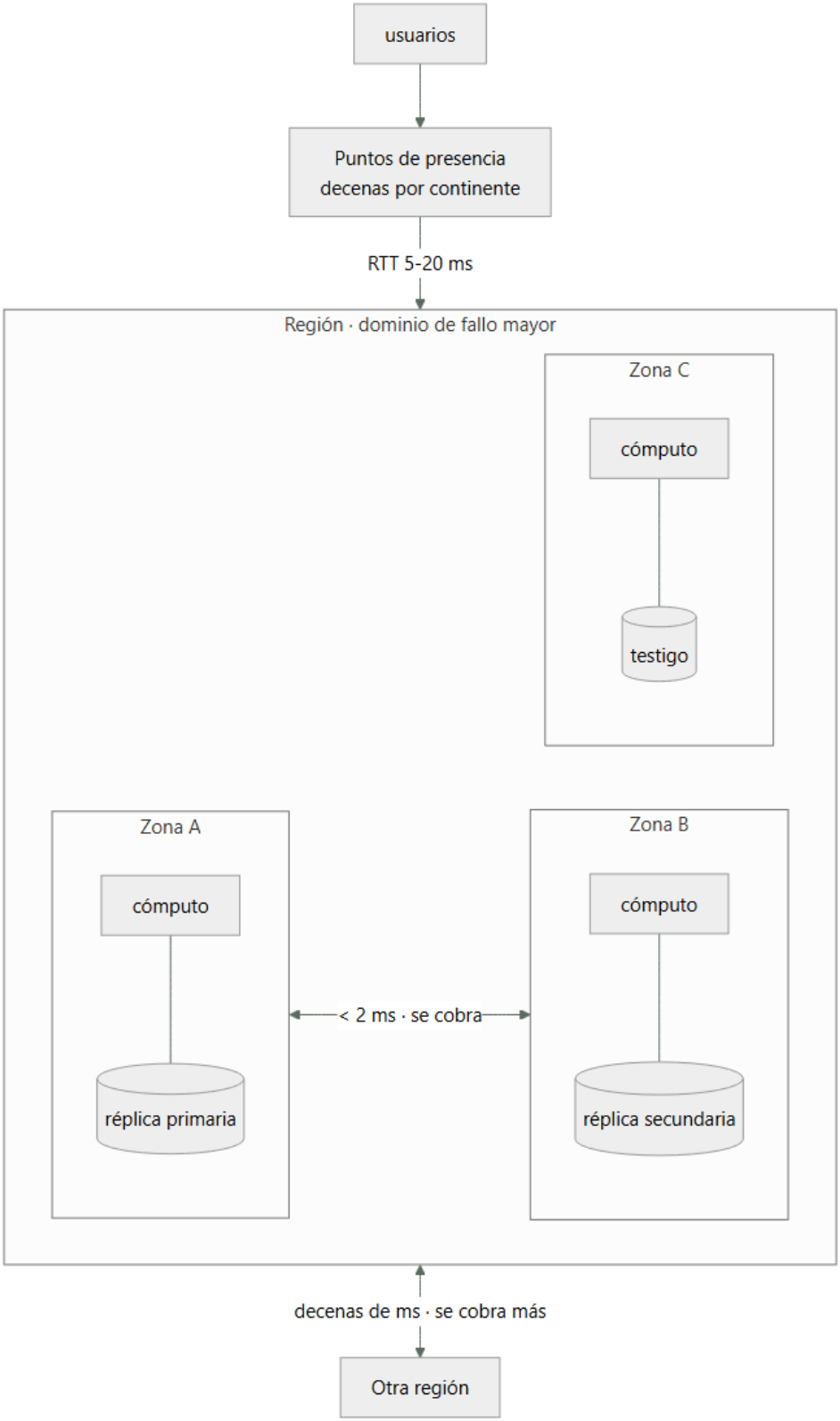
Concepto	Comprensión verificable
región	Área geográfica con varias zonas independientes. Es la unidad de soberanía del dato y de aislamiento de servicios: un fallo regional es el mayor dominio de fallo que un proveedor reconoce.
zona de disponibilidad	Uno o más centros de datos con energía, refrigeración y red independientes dentro de una región, separados kilómetros pero conectados con latencia de un dígito de milisegundos.
punto de presencia	Ubicación de borde que termina conexiones y sirve contenido cacheado cerca del usuario. No ejecuta tu aplicación completa: reduce el RTT del primer tramo.
dominio de fallo	Conjunto de componentes que caen juntos. El diseño consiste en elegir qué comparte dominio y qué no, porque redundancia dentro del mismo dominio no es redundancia.
transferencia de salida	Tráfico que sale hacia otra zona, región o internet. Es asimétrico: la entrada suele ser gratis y la salida se cobra, lo que convierte la topología en una decisión de coste.

Modelo mental

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

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. Tres niveles, tres órdenes de magnitud

La jerarquía no es organizativa: cada salto cambia latencia, dominio de fallo y precio en un orden de magnitud.

Nivel	Latencia entre pares	Cae junto por	Transferencia
Dentro de una zona	< 0,5 ms	Rack, sala	Normalmente gratis

Nivel	Latencia entre pares	Cae junto por	Transferencia
Entre zonas de una región	0,5-2 ms	Desastre regional	Se cobra, precio bajo
Entre regiones	10-150 ms	Nada común	Se cobra, precio alto
Punto de presencia a usuario	5-20 ms	Un PoP concreto	Salida a internet, el más caro

La fila de zonas es la que sostiene la alta disponibilidad: 2 ms es lo bastante bajo para replicar de forma síncrona una base de datos, y lo bastante independiente para que un incendio, un corte eléctrico o una inundación no alcancen a las dos. Ese es exactamente el punto de equilibrio que buscan los proveedores al situarlas: suficientemente lejos para no compartir riesgo físico, suficientemente cerca para no romper la replicación síncrona.

Entre regiones, los 10-150 ms hacen inviable la replicación síncrona: cada escritura pagaría el RTT completo. Por eso la replicación entre regiones es asíncrona, y por eso siempre tiene un RPO mayor que cero. No es una limitación del producto: es la velocidad de la luz de la clase 001 aplicada al diseño de datos.

2. Disponibilidad compuesta: la aritmética de las zonas

Si un componente tiene disponibilidad a y se despliegan n copias en dominios de fallo independientes, la disponibilidad del conjunto es:

```
A(n) = 1 − (1 − a)■
```

Con una disponibilidad por zona de 99,5 %:

```
1 zona:  1 − 0,005¹ = 99,5 %      → 3,6 h de caída al mes
2 zonas: 1 − 0,005² = 99,9975 %   → 1,1 min
3 zonas: 1 − 0,005³ = 99,999988 % → 0,3 s
```

El salto de una a dos zonas divide la indisponibilidad por 200. El de dos a tres, por otras 200 — pero en términos absolutos pasa de 1,1 minutos a 0,3 segundos, una mejora que casi ningún negocio percibe. La tercera zona rara vez se justifica por disponibilidad; se justifica por quórum, que es otra cosa: los sistemas de consenso necesitan mayoría, y con dos zonas no hay mayoría posible si cae una.

Y la fórmula tiene una condición que se incumple a diario: exige independencia. Tres réplicas de cómputo en tres zonas, todas apuntando a una única base de datos en la zona A, tienen la disponibilidad de la zona A. La redundancia se calcula sobre el componente menos redundante, no sobre el más visible.

```
cómputo 3 zonas (99,999988 %) × base de datos 1 zona (99,5 %) = 99,5 %
```

El cómputo redundante no aporta nada mientras la base sea única. Es el error de diseño más común de esta parte.

3. La transferencia es asimétrica y decide topologías

El precio de mover un byte depende de qué frontera cruza, y la asimetría entrada/salida moldea las arquitecturas más de lo que se suele reconocer:

Cruce	Precio orientativo por GB
Entrada desde internet	0,00 USD
Dentro de la misma zona	0,00 USD
Entre zonas de una región	0,01 USD (cada sentido)
Entre regiones	0,02-0,09 USD
Salida a internet	0,05-0,12 USD
Salida vía CDN	0,02-0,085 USD

Dos consecuencias prácticas:

El tráfico entre zonas se cobra en ambos sentidos. Un diseño que reparte peticiones aleatoriamente entre tres zonas hace que dos tercios de las llamadas internas crucen zona. Con 40 TB mensuales de tráfico interno:

```
reparto aleatorio: 2/3 × 40 TB × 2 sentidos × 0,01 USD/GB = 546 USD/mes
afinidad de zona:  ~5 % cruza zona                        =  41 USD/mes
```

La afinidad de zona —que una petición se resuelva dentro de la zona donde entró— ahorra un orden de magnitud, a cambio de complicar el balanceo y de perder algo de uniformidad de carga. Es una decisión, no una optimización obvia.

La salida a internet es el precio más alto y el menos vigilado. Un servicio que devuelve respuestas de 200 KB en lugar de 20 KB multiplica por diez la partida más cara de la factura. Comprimir y paginar no son solo mejoras de latencia.

4. Edge: para qué sirve y para qué no

Un punto de presencia reduce el RTT del primer tramo, y eso solo ayuda si el RTT es el componente dominante. Con lo visto en la clase 006, la petición en frío cuesta unos 4 RTT.

Sirve para: contenido estático cacheable, terminación de TLS cerca del usuario, absorción de picos y de ataques volumétricos, y lógica de borde muy corta —reescrituras, redirecciones, comprobación de un token—.

No sirve para: reducir el tiempo de una consulta a la base de datos, ni para nada que necesite estado consistente, ni cuando la respuesta es personalizada y no cacheable. En esos casos el borde añade un salto y empeora la latencia.

```
sin edge:  usuario ■■90 ms■■ origen                    = 4 × 90  = 360 ms
con edge:  usuario ■■10 ms■■ PoP ■■80 ms■■ origen
  contenido cacheado en el PoP:  4 × 10               =  40 ms   ✓
  contenido NO cacheable:        3 × 10 + 1 × 90 + 10 = 130 ms   ✓ (aún mejor: TLS al borde)
  respuesta dinámica sin reutilizar conexión al origen: puede superar los 360 ms  ✗
```

La tercera línea es la que sorprende: el edge mejora incluso contenido dinámico si mantiene conexiones calientes hacia el origen, porque ahorra el establecimiento TCP y TLS. Si no las mantiene, añade un tramo y no ahorra nada. Por eso el origin shielding no es un extra: es lo que hace que el borde funcione para contenido no cacheable.

5. Las zonas son etiquetas por cuenta

Un detalle operativo que produce errores difíciles de diagnosticar: en varios proveedores, el nombre de zona que ves —us-east-1a— está mapeado de forma distinta en cada cuenta. La us-east-1a de tu cuenta y la de otra pueden ser centros de datos físicos diferentes.

El mapeo se hizo para repartir la carga: si todos los clientes eligen «la primera zona», la primera zona física se satura. Aleatorizar la correspondencia por cuenta distribuye esa preferencia.

Las consecuencias:

1. Comparar despliegues entre cuentas por nombre de zona no tiene sentido. «Ambos están en 1a» no significa que compartan riesgo ni que estén cerca.
2. Colocar recursos de dos cuentas en la misma zona física —para latencia o para coste de transferencia— exige el identificador estable de zona (use1-az4), no el nombre.
3. Un incidente que el proveedor comunica por zona física afecta a nombres distintos según la cuenta.

Además, no todas las regiones tienen el mismo número de zonas —las hay con 3 y con 6— ni todos los servicios están en todas ellas. Diseñar «para tres zonas» y desplegar en una región de dos rompe el supuesto de disponibilidad calculado antes, sin ningún aviso.

Ejemplo trabajado

CloudShop debe elegir topología para su plataforma de pedidos con dos requisitos: 99,95 % mensual y RPO de 15 minutos. Se evalúan tres opciones con la misma aritmética.

Disponibilidad objetivo traducida a tiempo:

```
99,95 % mensual → 0,0005 × 30 × 24 × 60 = 21,6 min de caída permitida
```

Opción 1 — una zona. Con 99,5 % por zona:

```
A = 99,5 %  →  3,6 h/mes    ✗ incumple por un factor de 10
```

Opción 2 — dos zonas, base de datos con réplica síncrona.

```
cómputo:  1 − 0,005² = 99,9975 %
base de datos síncrona entre zonas (latencia 1,4 ms medida)
A compuesta ≈ 99,995 %  →  1,3 min/mes    ✓ cumple con holgura
RPO = 0 (síncrona)                         ✓
```

Coste de transferencia entre zonas, con 40 TB mensuales de tráfico interno:

```
sin afinidad: 2/3 × 40.000 GB × 2 × 0,01 = 533 USD/mes
con afinidad de zona: ~5 %               =  40 USD/mes
```

Opción 3 — dos regiones, réplica asíncrona.

```
A ≈ 99,999 %                                ✓
RPO: retraso de replicación medido, p95 = 4,2 min   ✓ dentro de los 15
transferencia entre regiones: 40 TB × 0,02 = 800 USD/mes
coste de capacidad duplicada:            ≈ 6.500 USD/mes
```

Comparación final contra los requisitos:

```
Opción 1 · 1 zona          99,5 %    0        —               ✗
Opción 2 · 2 zonas        99,995 %   0        40 USD          ✓
Opción 3 · 2 regiones     99,999 %   4,2 min  7.300 USD       ✓
```

Se elige la opción 2. La 3 cumple mejor y cuesta 180 veces más; la mejora de 1,3 min a 0,3 min mensuales no la percibe ningún cliente y el requisito es 21,6.

Antes de cerrar se verifica el error clásico —redundancia sobre el componente menos redundante—:

```
$ aws rds describe-db-instances --query 'DBInstances[0].[MultiAZ,AvailabilityZone,
SecondaryAvailabilityZone]'
[true, "us-east-1a", "us-east-1c"]        # la base SÍ cruza zona
$ aws ec2 describe-availability-zones --query 'AvailabilityZones[].[ZoneName,ZoneId]' --output text
us-east-1a  use1-az4
us-east-1c  use1-az6                       # zonas físicas distintas: independientes
```

Se registra el identificador estable, no el nombre, porque el nombre no es comparable entre cuentas.

Límite explícito declarado en el ADR: la opción 2 no sobrevive a un fallo regional completo. Ese riesgo se acepta conscientemente; si el negocio cambia el requisito a continuidad regional, la decisión se revisa y el sobrecoste de 7.300 USD/mes pasa a estar justificado. Escribirlo evita que dentro de un año alguien descubra el límite durante un incidente.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-01-cloud-principles-strategy-adoption/014-regiones-zonas-de-disponibilidad-puntos-de-presencia-y-edge/lab.py
```

El laboratorio selecciona el motor de práctica architecture 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 mapa-de-topologia 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 diagrama acompañado por decisiones y trade-offs. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye mapa-de-topologia 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 despliega cómputo en tres zonas y el sistema cae igual cuando falla una	La base de datos seguía en una sola zona: la disponibilidad la fija el componente menos redundante	Calcula la disponibilidad compuesta de la cadena completa, no del componente más visible.
Aparece una partida de transferencia inesperada de cientos de dólares	El tráfico interno cruza zonas y se cobra en ambos sentidos	Aplica afinidad de zona para el tráfico este-oeste y mide qué fracción cruza.
Se añade un CDN y la latencia de las respuestas dinámicas empeora	El borde añade un salto y no mantenía conexiones calientes al origen	Activa origin shielding y conexiones persistentes, o no pases el tráfico dinámico por el borde.
Dos cuentas creen estar en la misma zona y no lo están	El nombre de zona se mapea distinto por cuenta	Usa el identificador estable de zona para cualquier comparación o colocación entre cuentas.
Un diseño calculado para tres zonas se despliega en una región que solo tiene dos	Se asumió un número de zonas uniforme entre regiones	Verifica zonas disponibles y servicios ofrecidos por región antes de fijar el modelo de disponibilidad.

■ 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. Con 99,5 % por zona, ¿cuánta indisponibilidad mensual queda con una, dos y tres zonas? ¿Qué justifica la tercera?
2. Tres réplicas de cómputo en tres zonas contra una base de datos en una sola. ¿Cuál es la disponibilidad del conjunto?
3. ¿Por qué la replicación entre regiones es asíncrona y qué implica eso para el RPO?
4. ¿En qué caso concreto un punto de presencia empeora la latencia de una respuesta dinámica?
5. ¿Por qué no puedes comparar la zona 1a de dos cuentas distintas, y qué identificador sí es comparable?

Referencias

- AWS (2024). Regions, Availability Zones, and Local Zones — identificadores estables de zona y su mapeo por cuenta. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html>
- Microsoft (2024). Azure regions and availability zones — modelo de zonas y servicios con soporte por región. <https://learn.microsoft.com/en-us/azure/reliability/availability-zones-overview>
- Google Cloud (2024). Geography and regions — jerarquía región/zona y ubicación de recursos. <https://cloud.google.com/docs/geography-and-regions>
- Beyer, B. et al., eds. (2016). Site Reliability Engineering, cap. 22 «Addressing Cascading Failures» — independencia de dominios de fallo. <https://sre.google/sre-book/addressing-cascading-failures/>

- Brewer, E. (2012). CAP Twelve Years Later: How the Rules Have Changed. IEEE Computer 45(2) — por qué la latencia entre regiones fuerza replicación asíncrona. <https://doi.org/10.1109/MC.2012.37>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 01 en PDF · Manual integral

Anterior	Índice	Siguiente
← 013 · Definición NIST y características esenciales de cloud	Parte 01 · Programa	015 · IaaS, PaaS, SaaS, CaaS y FaaS →

Evaluación — 014 Regiones, zonas de disponibilidad, puntos de presencia y edge

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 mapa-de-topologia 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.

015 — IaaS, PaaS, SaaS, CaaS y FaaS

Parte: 01 — Principios, estrategia y adopción cloud Nivel: inicial-intermedio · Horas estimadas: 4 Laboratorio: decision · Estado: EXECUTABLECORE

Propósito

Elegir modelo de servicio por lo que cede y lo que retiene, no por comodidad aparente. Cada escalón hacia arriba entrega operación al proveedor y a cambio impone sus límites: cuotas, tiempos máximos, runtimes soportados y una forma concreta de bloqueo. Esta clase da el criterio con el que se decidirá entre EC2, Fargate y Lambda —o sus equivalentes— en las partes 17, 18 y 19.

Resultados de aprendizaje

Al finalizar podrás:

1. Situar una carga en el modelo adecuado a partir de su perfil de ejecución y su tolerancia a límites.
2. Calcular el punto de cruce económico entre función y contenedor permanente para una carga dada.
3. Anticipar qué límites duros impone cada modelo y cuáles no se pueden negociar.
4. Distinguir bloqueo de datos, de API y operativo, y estimar el coste de salida de cada uno.
5. Justificar por qué CaaS y FaaS no son escalones sucesivos sino respuestas a preguntas distintas.

Conceptos centrales

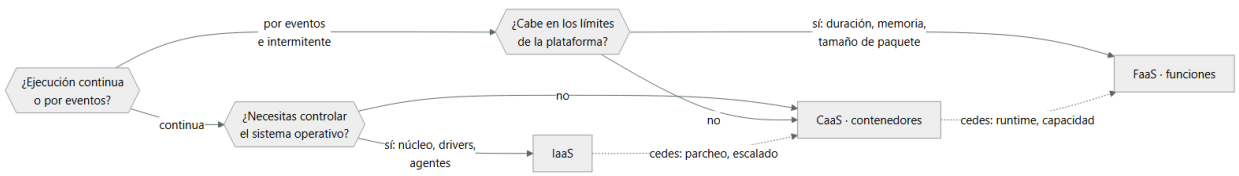
Concepto	Comprensión verificable
arranque en frío	Latencia adicional cuando no hay instancia caliente y hay que crear el entorno de ejecución. Va de decenas de milisegundos a varios segundos según runtime y tamaño del paquete, y solo afecta a una fracción de las invocaciones.
concurrencia	Ejecuciones simultáneas. En FaaS es la unidad de escalado y de cuota; en CaaS y IaaS se deriva del número de instancias y de trabajadores por instancia.
bloqueo de proveedor	Coste de migrar a otro proveedor. No es binario: se descompone en datos, API y operación, y cada componente tiene un coste de salida distinto.
plano de control	Parte del servicio que gestiona el ciclo de vida —crear, escalar, actualizar—. Al subir de modelo se cede el plano de control y con él la capacidad de intervenir cuando falla.
granularidad de facturación	Unidad mínima que se cobra: hora, segundo o milisegundo con memoria asignada. Determina si una carga intermitente es barata o ruinosa.

Modelo mental

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

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. Cada escalón cede operación y recibe límites

Subir de modelo no es «más fácil»: es un intercambio explícito. Se entrega trabajo operativo y se aceptan restricciones que antes no existían.

	IaaS	CaaS	FaaS
Tú operas	SO, parches, escalado, runtime	Imagen y réplicas	Solo el código
El proveedor opera	Hardware, hipervisor	+ SO y orquestación	+ capacidad y escalado
Unidad de escalado	Instancia (minutos)	Tarea (decenas de s)	Invocación (ms)
Facturación	Por segundo, encendido	Por segundo, con tarea viva	Por ms × memoria, solo al ejecutar
Límite duro típico	Cuota de instancias	Cuota de tareas	Duración máxima
Estado local	Persistente	Efímero por tarea	Efímero, no garantizado

La fila de límites duros es la decisiva y la que más tarde se descubre. En FaaS existe un techo de duración por invocación —15 minutos en AWS Lambda, 60 en Cloud Run functions de 2.ª generación, 10 en el plan de consumo de Azure Functions— que no es negociable con soporte. Una carga que hoy tarda 8 minutos y crece un 20 % anual choca contra el techo en tres años, y la migración no será un ajuste de configuración: será reescribir el modelo de ejecución.

La fila de facturación explica el resto: FaaS solo cobra mientras ejecuta. Para una carga que corre 50 ms cada 10 minutos, eso es la diferencia entre pagar 0,4 segundos al día y pagar 24 horas.

2. El punto de cruce económico se calcula, no se opina

La comparación entre función y contenedor permanente tiene un umbral concreto. Con precios orientativos:

```
FaaS:  0,0000166667 USD por GB-segundo + 0,20 USD por millón de invocaciones
CaaS:  ~0,04 USD por hora de 1 vCPU + 2 GB (tarea permanente)
```

Para una función de 512 MB que tarda 200 ms, con N invocaciones al mes:

```
coste FaaS(N) = N × 0,2 s × 0,5 GB × 0,0000166667 + N × 0,0000002
              = N × (1,6667e-6 + 2e-7) = N × 1,8667e-6

coste CaaS    = 730 h × 0,04 = 29,20 USD/mes (fijo)

umbral: N = 29,20 / 1,8667e-6 ≈ 15,6 millones de invocaciones/mes
                              ≈ 6 invocaciones por segundo sostenidas
```

Por debajo de 6 invocaciones por segundo sostenidas, la función sale más barata; por encima, el contenedor. Y el umbral se desplaza con la duración: si la función tardara 2 segundos en vez de 200 ms, el cruce bajaría a 1,56 millones —0,6 por segundo—, porque el coste de FaaS es lineal en el tiempo de ejecución.

Dos matices que el cálculo simple omite:

- El contenedor permanente necesita al menos dos réplicas para tolerar fallos, así que su coste real se duplica y el umbral sube.
- FaaS escala a cero, así que en entornos de pruebas o cargas estacionales su ventaja es mayor que la que sugiere la media.

3. El arranque en frío importa menos de lo que se teme y más de lo que se mide

El arranque en frío se cita como el argumento contra FaaS y casi siempre se estima mal en ambas direcciones.

Afecta solo a las invocaciones que no encuentran instancia caliente. Con tráfico sostenido, esa fracción es pequeña:

```
invocaciones/mes            2.000.000
arranques en frío medidos      12.000     → 0,6 %
latencia p50 en caliente          45 ms
latencia p50 en frío             820 ms

impacto en p50 global: despreciable
impacto en p99: el p99 ES el arranque en frío si supera el 1 %
```

La segunda línea es la que se olvida: con un 0,6 % de arranques en frío, el p99 global no los ve; con un 1,5 %, el p99 pasa a ser el arranque en frío. La pregunta correcta no es «¿hay arranque en frío?» sino «¿qué percentil de mi SLO cae dentro de esa fracción?».

Los factores que la determinan, por orden de impacto:

1. Tamaño del paquete: hay que descargarlo y descomprimirlo. Un artefacto de 250 MB arranca en frío mucho peor que uno de 5 MB.
2. Runtime: los que necesitan inicializar una máquina virtual —JVM, .NET— pagan más que los interpretados o compilados a nativo.
3. Conectividad a red privada: adjuntar la función a una VPC añadía segundos históricamente; hoy es del orden de decenas de milisegundos, pero conviene medirlo y no asumirlo.

Mitigaciones reales: concurrencia aprovisionada —que elimina el ahorro de escalar a cero y cambia la aritmética anterior—, reducir el paquete, y mantener caliente solo la ruta crítica.

4. Bloqueo: tres componentes con costes de salida distintos

«Bloqueo de proveedor» se usa como una sola cosa y son tres, con coste de salida muy desigual:

Tipo	Ejemplo	Coste de salida
De datos	Formato propietario, egreso de 40 TB	Alto: se paga por GB y lleva tiempo
De API	SDK específico incrustado en el dominio	Medio: reescribir adaptadores
Operativo	Runbooks, formación, herramientas del equipo	El más alto y el menos contabilizado

El orden habitual de preocupación es el inverso al orden de coste. Se discute mucho sobre no usar un servicio propietario y nada sobre que el equipo solo sabe operar una consola.

El bloqueo crece con el modelo de servicio, pero no de forma uniforme:

- IaaS: bajo en API —una máquina virtual es una máquina virtual—, alto en datos si el volumen es grande.
- CaaS: bajo si la imagen es OCI estándar y el orquestador es Kubernetes; alto si se usan extensiones propietarias del proveedor.
- FaaS: alto en API —el modelo de eventos, los disparadores y el formato de entrada son específicos— pero el código de negocio puede aislarse tras un adaptador fino.

La estrategia práctica no es evitar el bloqueo, que es imposible y caro: es saber cuánto cuesta salir y decidir a sabiendas. Un servicio propietario que ahorra 30.000 USD al año con un coste de salida estimado de 40.000 es una decisión razonable; el problema es no haber calculado nunca el segundo número.

5. CaaS y FaaS no son escalones, son respuestas distintas

La presentación habitual como escalera —IaaS, PaaS, CaaS, FaaS, cada uno «más gestionado»— sugiere que FaaS es la meta. No lo es: responden a preguntas diferentes.

CaaS responde a: quiero portabilidad del artefacto y control del entorno de ejecución, sin gestionar sistemas operativos. La imagen OCI corre igual en cualquier sitio; el coste es que sigues decidiendo réplicas, límites y sondas.

FaaS responde a: quiero que la unidad de escalado sea la petición y no pagar por capacidad ociosa. El coste es aceptar los límites de la plataforma y un modelo de eventos específico.

Una carga puede ser mala candidata para FaaS y excelente para CaaS aunque sea moderna y sin estado: basta con que su duración supere el techo, que necesite conexiones persistentes —WebSocket largo— o que su patrón de tráfico sea sostenido y alto, donde el umbral calculado antes favorece al contenedor.

Y al revés: una tarea programada que corre 3 segundos cada hora es ruinosa como contenedor permanente —730 horas facturadas para 0,73 horas de trabajo, un 0,1 % de utilización— y trivial como función.

El criterio es el perfil de ejecución, no la modernidad del modelo.

Ejemplo trabajado

CloudShop tiene tres cargas que hoy corren en máquinas virtuales y quiere decidir modelo para cada una. Se aplican las dos preguntas del árbol y la aritmética.

Carga A — generación de miniaturas al subir una imagen.

```
perfil       por eventos, 8.400 invocaciones/día, 1,2 s cada una, 512 MB
límites      duración 1,2 s « techo de 15 min          ✓ cabe
coste FaaS   8.400×30 × 1,2 s × 0,5 GB × 1,6667e-6 + invocaciones
             = 252.000 × 1,0e-6 + 0,05 = 0,30 USD/mes
coste CaaS   2 réplicas × 29,20 = 58,40 USD/mes
utilización  252.000 × 1,2 s / (730×3600) = 11,5 %
```

FaaS, por un factor de 195. La utilización del 11,5 % es exactamente el caso que la función resuelve.

Carga B — API de catálogo.

```
perfil       continua, 610 peticiones/s en pico, 45 ms, 512 MB
umbral calculado: ~6 inv/s → 610 está 100 veces por encima
coste FaaS   610×2,6e6 s/mes × 0,045 s × 0,5 × 1,6667e-6 ≈ 1.190 USD/mes
coste CaaS   por ley de Little: L = 610 × 0,045 = 27,5 concurrentes
             a ρ=0,7 → 40 concurrentes → 4 tareas de 16 → 4 × 29,20 = 117 USD/mes
```

CaaS, por un factor de 10. Tráfico sostenido y alto: el contenedor gana con holgura.

Carga C — procesamiento nocturno de conciliación contable.

```
perfil       una vez al día, 38 min actuales, crecimiento 20 % anual
límites      38 min > techo de 15 min de FaaS          ✗ no cabe
             y en 3 años serán 66 min
necesita     acceso a un driver de base de datos legado con biblioteca nativa
```

Dos exclusiones independientes: excede el techo de duración y necesita control del entorno para una biblioteca nativa. La segunda también descarta CaaS con imágenes mínimas sin capacidad de instalar dependencias del sistema.

```
resultado    IaaS o CaaS con imagen completa; se elige CaaS como tarea programada
             coste: 38 min/día × 30 = 19 h/mes × 0,04 = 0,76 USD/mes
             frente a una VM encendida 730 h = 29,20 USD/mes
```

Decisión final y bloqueo estimado:

```
A · miniaturas       FaaS      0,30 USD   alto (eventos)    ~2 días de trabajo
B · API catálogo     CaaS    117,00 USD   bajo (OCI)        ~0: imagen portable
C · conciliación     CaaS      0,76 USD   bajo              ~0
```

El bloqueo alto de A se acepta explícitamente: el adaptador de eventos son unas 40 líneas y el resto de la lógica es agnóstica. Se registra en el ADR que la lógica de negocio no debe importar tipos del SDK del proveedor, que es lo que convierte dos días en dos meses cuando llega la migración.

Ahorro total frente a las tres máquinas virtuales actuales (3 × 29,20 = 87,60 USD, más operación): el coste baja a 118 USD pero la carga B pasa de una VM a cuatro tareas, dimensionadas por la ley de Little en vez de por costumbre. El ahorro real no está en la factura sino en dejar de parchear tres sistemas operativos.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-01-cloud-principles-strategy-adoption/015-iaas-paas-saas-caas-y-faas/lab.py
```

El laboratorio selecciona el motor de práctica decision 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-de-servicios 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 decisión ponderada y un ADR. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye matriz-de-servicios 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 carga migrada a funciones empieza a fallar cuando crece	Se acercó al techo de duración por invocación, que no es negociable	Comprueba el margen contra el techo con la proyección de crecimiento antes de elegir FaaS.
El coste de las funciones supera al de los contenedores que sustituyeron	El tráfico sostenido está por encima del punto de cruce	Calcula el umbral con tu duración y memoria reales; con tráfico alto y continuo, el contenedor gana.
Se activa concurrencia aprovisionada y desaparece el ahorro esperado	La concurrencia aprovisionada elimina el escalado a cero, que era la fuente del ahorro	Aprovisiona solo la ruta crítica y recalcula el umbral con ese coste fijo incluido.
El p99 se dispara aunque el p50 esté bien	La fracción de arranques en frío supera el 1 % y pasa a dominar el percentil del SLO	Mide la fracción real y compárala con el percentil que fija tu SLO, no con la media.
Migrar de proveedor cuesta meses pese a usar contenedores portables	El bloqueo operativo —runbooks, herramientas, formación— no se contabilizó	Estima los tres tipos de bloqueo por separado; el operativo suele ser el mayor.

■ 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. Una función de 512 MB tarda 200 ms. ¿A partir de cuántas invocaciones por segundo sale más barato un contenedor permanente, y cómo cambia si tardara 2 s?
2. ¿Por qué un 0,6 % de arranques en frío no afecta al p99 y un 1,5 % sí?
3. Una carga sin estado, moderna y de 40 minutos. ¿Es candidata a FaaS? Justifica con el límite concreto.
4. Ordena los tres tipos de bloqueo por coste de salida y explica por qué el orden de preocupación suele ser el inverso.
5. Una tarea corre 3 segundos cada hora. ¿Qué utilización tendría como contenedor permanente y qué modelo corresponde?

Referencias

- Mell, P. y Grance, T. (2011). The NIST Definition of Cloud Computing, SP 800-145 — los tres modelos de servicio originales. <https://doi.org/10.6028/NIST.SP.800-145>
- AWS (2024). Lambda quotas — techo de duración, memoria, tamaño de paquete y concurrencia. <https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html>
- Google Cloud (2024). Cloud Run functions: quotas and limits — límites de la 1.ª y 2.ª generación. <https://cloud.google.com/functions/quotas>
- Jonas, E. et al. (2019). Cloud Programming Simplified: A Berkeley View on Serverless Computing — límites estructurales del modelo de funciones. <https://doi.org/10.48550/arXiv.1902.03383>
- Hohpe, G. (2020). Cloud Strategy, cap. sobre bloqueo — descomposición del coste de salida en datos, API y operación.
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 01 en PDF · Manual integral

Anterior	Índice	Siguiente
← 014 · Regiones, zonas de disponibilidad, puntos de presencia y edge	Parte 01 · Programa	016 · Elasticidad, escalabilidad, disponibilidad y resiliencia →

Evaluación — 015 IaaS, PaaS, SaaS, CaaS y FaaS

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-de-servicios 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.

016 — Elasticidad, escalabilidad, disponibilidad y resiliencia

Parte: 01 — Principios, estrategia y adopción cloud Nivel: inicial-intermedio · Horas estimadas: 4 Laboratorio: reliability · Estado: EXECUTABLECORE

Propósito

Separar cuatro palabras que se usan como sinónimos y significan cosas distintas: elasticidad, escalabilidad, disponibilidad y resiliencia. Confundirlas produce sistemas que escalan pero no sobreviven, o que sobreviven pero cuestan el triple. Aquí se establece la aritmética —ley de Amdahl, ley universal de escalabilidad, disponibilidad compuesta— con la que se dimensionará el resto del programa.

Resultados de aprendizaje

Al finalizar podrás:

1. Definir las cuatro propiedades por lo que garantizan y por lo que no, con un ejemplo de sistema que cumple una y falla otra.
2. Aplicar la ley de Amdahl para acotar la ganancia máxima de paralelizar una carga.
3. Explicar por qué a partir de cierto punto añadir nodos empeora el rendimiento, usando la ley universal de escalabilidad.
4. Dimensionar un autoescalado con umbrales, periodos de enfriamiento y margen de arranque que no oscile.
5. Distinguir MTBF de MTTR y justificar por qué reducir el segundo suele ser más barato que aumentar el primero.

Conceptos centrales

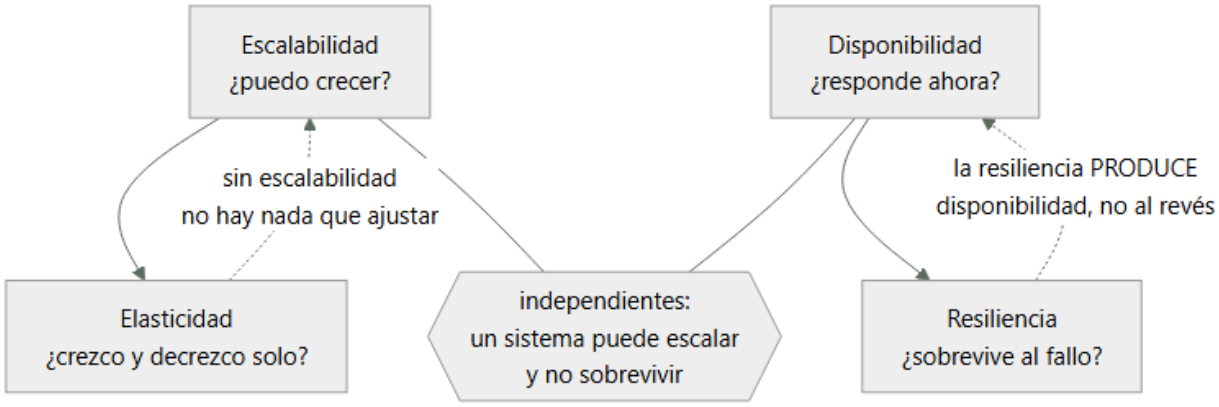
Concepto	Comprensión verificable
escalabilidad	Capacidad de aumentar la capacidad al añadir recursos. Es una propiedad del diseño y tiene un techo determinado por la fracción no paralelizable y por el coste de coordinación.
elasticidad	Capacidad de ajustar la capacidad a la demanda de forma automática y en ambas direcciones. Presupone escalabilidad, pero un sistema escalable puede no ser elástico si nadie automatiza el ajuste.
disponibilidad	Fracción del tiempo en que el sistema responde correctamente. Se mide como MTBF/(MTBF+MTTR), lo que revela que hay dos palancas y no una.
resiliencia	Capacidad de seguir prestando servicio, aunque degradado, mientras algo falla, y de recuperarse después. Es lo que hace que un fallo no se convierta en una caída.
coherencia	En la ley universal de escalabilidad, el coste de mantener sincronizados N nodos. Crece con N^2 y es lo que hace que añadir nodos llegue a empeorar el rendimiento total.

Modelo mental

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

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. Cuatro propiedades, cuatro preguntas distintas

El vocabulario se usa mal a diario y cada confusión tiene un coste concreto:

Propiedad	Pregunta	Se mide con	Falla cuando
Escalabilidad	¿Puedo atender más añadiendo recursos?	Rendimiento frente a nodos	Hay un cuello serializado
Elasticidad	¿Se ajusta solo, en ambos sentidos?	Tiempo de reacción y utilización	Solo escala hacia arriba
Disponibilidad	¿Responde ahora?	Tiempo útil / tiempo total	Un componente único cae
Resiliencia	¿Sigue sirviendo mientras falla algo?	Comportamiento bajo fallo inyectado	Todo es esencial

Las combinaciones que existen en la práctica y demuestran que son independientes:

- Escalable pero no disponible: 200 réplicas sin estado contra una única base de datos. Atiende cualquier volumen y cae entera cuando cae la base.
- Disponible pero no elástico: tres zonas sobredimensionadas al triple, siempre encendidas. Nunca cae y desperdicia el 70 % del gasto.
- Elástico pero no resiliente: escala en 90 segundos y toda dependencia es esencial. Si el servicio de recomendaciones tarda, el catálogo entero devuelve 500.
- Resiliente pero no escalable: degrada con elegancia, tiene circuit breakers y plazos, y su cuello de botella serializado le impide pasar de 400 peticiones por segundo hagas lo que hagas.

Cada una necesita un trabajo distinto. Pedir «que escale» cuando el problema es de resiliencia produce más réplicas de algo que sigue cayendo igual.

2. La ley de Amdahl pone el techo

Si una fracción p del trabajo se puede paralelizar y el resto no, la aceleración con N recursos es:

```
S(N) = 1 / ((1 − p) + p/N)

límite cuando N → ∞:   S(∞) = 1 / (1 − p)
```

La consecuencia es brutal y contraintuitiva:

Fracción paralelizable	Aceleración máxima	Con 16 nodos
50 %	2×	1,88×
90 %	10×	6,4×
95 %	20×	9,1×
99 %	100×	13,9×

Con un 95 % paralelizable, el 5 % serializado limita la ganancia a 20× por muchos nodos que pongas. Y con 16 nodos solo se obtiene 9,1× —un 57 % de eficiencia—: la mitad del gasto no produce rendimiento.

Aplicado a una petición web, la fracción serializada suele estar en sitios concretos y localizables:

- Una transacción que toma un bloqueo sobre una fila muy consultada.
- Un contador global incrementado en cada petición.
- Una secuencia de base de datos para generar identificadores.
- Un límite de tasa implementado con un único contador central.

Encontrar y eliminar el 5 % serializado da más rendimiento que triplicar los nodos. Ese es el trabajo de ingeniería que un autoescalado no puede sustituir.

3. La ley universal de escalabilidad: cuando añadir nodos empeora

Amdahl explica el techo pero no explica algo que se observa en producción: a partir de cierto número de nodos el rendimiento baja. Gunther lo modela añadiendo un término de coherencia:

```
C(N) = N / (1 + σ(N − 1) + κN(N − 1))

σ = contención (serialización, como en Amdahl)
κ = coherencia (coste de sincronizar entre nodos, crece con N²)
```

Con $\sigma = 0{,}03$ y $\kappa = 0{,}0001$:

N	C(N)	Eficiencia
10	7,8×	78 %
20	12,7×	64 %
40	17,4×	44 %
57	18,6×	33 % ← máximo
80	18,2×	23 %
120	16,6×	14 %

A partir de 57 nodos, cada nodo adicional reduce el rendimiento total. El término κN^2 es el coste de que todos se pongan de acuerdo: réplicas que sincronizan estado, nodos de caché que se invalidan entre sí, miembros de un clúster que intercambian latidos.

Esto explica un patrón que de otro modo parece magia: partir un clúster de 100 nodos en cuatro de 25 mejora el rendimiento agregado, porque κ actúa dentro de cada partición y no entre ellas. Es el fundamento de la celdas y del particionado que aparecerán en la parte 12.

Ajustar σ y κ a un sistema real requiere medir el rendimiento con varios tamaños de clúster. El valor del modelo no es predecir con precisión, sino saber que el óptimo existe y es finito.

4. Disponibilidad: dos palancas, una mucho más barata

La definición operativa revela algo que la cifra de «nueves» oculta:

```
A = MTBF / (MTBF + MTTR)

MTBF = tiempo medio entre fallos
MTTR = tiempo medio de reparación
```

Con MTBF de 30 días y MTTR de 4 horas:

```
A = 720 / (720 + 4) = 99,45 %
```

Para llegar a 99,95 % hay dos caminos:

```
A) subir MTBF a 8.000 h (11 meses sin fallos) manteniendo MTTR de 4 h
B) bajar MTTR a 22 min manteniendo MTBF de 30 días
```

El camino A exige hacer el sistema 11 veces más fiable; el B, detectar y recuperar 11 veces más rápido. El segundo casi siempre es más barato y, sobre todo, es alcanzable: automatizar la conmutación, tener runbooks ejecutables y

alertar sobre síntomas en vez de causas.

Esto reordena las prioridades de inversión. Un equipo que gasta en hacer improbable el fallo obtiene rendimientos decrecientes; uno que gasta en detectar rápido y recuperar automáticamente mejora la disponibilidad de forma lineal con el esfuerzo.

Y hay un tercer factor que la fórmula no muestra: reducir el radio de impacto cambia el numerador de la ecuación de negocio. Un fallo que afecta al 5 % de los usuarios durante 4 horas no es equivalente a uno que afecta al 100 %, aunque la disponibilidad medida como tiempo sea idéntica. Por eso las celdas y las implantaciones progresivas mejoran la experiencia sin tocar ni MTBF ni MTTR.

5. Autoescalado que no oscila

Un autoescalado mal configurado produce oscilación: añade capacidad, la métrica baja, retira capacidad, la métrica sube, y el ciclo se repite consumiendo más de lo que ahorra.

Cuatro parámetros y su razón:

```
umbral de subida    70 %   ← por la clase 011: por encima, la latencia se dispara
umbral de bajada    40 %   ← histéresis: NO 65 %, o entra en ciclo
enfriamiento        300 s  ← > tiempo de arranque + estabilización
margen de arranque  120 s  ← lo que tarda una instancia en servir tráfico
```

La histéresis —la banda entre 40 % y 70 %— es lo que evita el ciclo. Si los umbrales estuvieran en 70 % y 65 %, al añadir una instancia sobre nueve la utilización caería de 71 % a 64 % y dispararía inmediatamente la bajada.

El margen de arranque impone un límite físico a lo que el autoescalado puede resolver: si una instancia tarda 120 segundos en servir tráfico, un pico que se duplica en 30 segundos no se puede absorber escalando. Solo hay tres respuestas posibles:

1. Capacidad preexistente suficiente para el pico (cara pero simple).
2. Encolar y absorber el pico con latencia mayor, no con más capacidad.
3. Rechazar con 429 y Retry-After, protegiendo lo que ya está en curso.

La tercera es la que casi nadie implementa y la que evita el colapso: degradar el servicio a una fracción de los usuarios es preferible a caer para todos. Es resiliencia, no escalabilidad, y por eso este punto pertenece a esta clase y no a la de capacidad.

Ejemplo trabajado

El catálogo de CloudShop sufre degradación en las promociones. La propuesta del equipo es «poner más réplicas»: de 12 a 40. Se comprueba con la aritmética antes de gastar.

Mediciones con distintos tamaños de clúster, a carga saturante:

```
nodos    rendimiento (rps)    eficiencia
  4          3.100              100 %
  8          5.520               89 %
 12          7.190               77 %
 16          8.320               67 %
 24          9.410               51 %
```

Ajustando la ley universal de escalabilidad a estos puntos:

```
σ ≈ 0,042    (contención)
κ ≈ 0,00035  (coherencia)

N óptimo = √((1 − σ)/κ) = √(0,958/0,00035) ≈ 52 nodos
C(52) ≈ 11.900 rps
C(40) ≈ 11.400 rps
C(24) ≈  9.410 rps
```

Pasar de 12 a 40 nodos cuesta 3,3 veces más y produce 1,59 veces el rendimiento. La eficiencia cae del 77 % al 40 %: se pagan 28 nodos para obtener el trabajo de 11.

Se busca el origen de σ = 0,042 antes de aceptar ese gasto:

```
-- consulta más frecuente durante el pico
SELECT stock FROM productos WHERE id = $1 FOR UPDATE;
```

Un bloqueo pesimista sobre la fila del producto en promoción. Todas las peticiones del artículo destacado se serializan sobre la misma fila: ese es el 4,2 % que la ley detectaba sin saber dónde estaba.

Corrección: se sustituye por un contador particionado en 16 fragmentos con reconciliación asíncrona, patrón que reaparecerá en la parte 12.

```
nueva medición tras el cambio:
nodos    rendimiento    eficiencia
 12         9.980          92 %
 16        12.900          89 %
 24        18.100          83 %

σ ≈ 0,008   κ ≈ 0,00009
N óptimo = √(0,992/0,00009) ≈ 105 nodos
```

Con 16 nodos se obtienen 12.900 rps: más que con 40 nodos del diseño anterior (11.400), a un 40 % del coste.

```
propuesta original         40    11.400        3,3×             285
tras eliminar el bloqueo   16    12.900        1,3×             806
```

Se fija el autoescalado con los parámetros calculados:

```
mínimo 12 · máximo 24 · subida al 70 % · bajada al 40 %
enfriamiento 300 s · margen de arranque medido 118 s
```

Y se añade lo que el autoescalado no puede resolver: el pico de la promoción se duplica en 25 segundos, por debajo de los 118 de arranque. Se implementa rechazo con 429 y Retry-After al superar el 85 % de utilización, que protege las peticiones en curso en vez de degradar todas.

La lección: la pregunta no era cuántas réplicas, sino cuál era σ. Encontrar el 4,2 % serializado valió más que triplicar el clúster.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-01-cloud-principles-strategy-adoption/016-elasticidad-escalabilidad-
disponibilidad-y-resiliencia/lab.py
```

El laboratorio selecciona el motor de práctica reliability 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 modelo-de-capacidad 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 escenario de fallo con objetivo y recuperación medida. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye modelo-de-capacidad 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 añaden nodos y el rendimiento total baja	El término de coherencia κN^2 supera la ganancia; se pasó el óptimo	Mide con varios tamaños, ajusta σ y κ, y particiona en clústeres menores en vez de crecer uno solo.
Escalar de 12 a 40 réplicas mejora un 60 % y cuesta un 230 % más	Hay una fracción serializada que la ley de Amdahl limita	Localiza el cuello serializado —bloqueos, contadores, secuencias— antes de escalar.
El autoescalado añade y quita instancias en ciclo continuo	Umbrales de subida y bajada demasiado próximos: no hay histéresis	Separa los umbrales (por ejemplo 70 % y 40 %) y fija un enfriamiento mayor que el arranque.
Un pico brusco tumba el servicio pese al autoescalado	El pico crece más rápido que el tiempo de arranque; escalar no llega a tiempo	Añade rechazo con 429 y Retry-After, o capacidad preexistente; el escalado no resuelve picos por debajo del margen de arranque.
Se invierte en fiabilidad y la disponibilidad apenas mejora	Se atacó el MTBF, con rendimientos decrecientes, en vez del MTTR	Reduce el tiempo de detección y recuperación: suele ser más barato y escala linealmente con el esfuerzo.

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. Con un 95 % del trabajo paralelizable, ¿cuál es la aceleración máxima y qué eficiencia obtienes con 16 nodos?
2. ¿Por qué partir un clúster de 100 nodos en cuatro de 25 puede mejorar el rendimiento agregado?
3. Un sistema tiene MTBF de 30 días y MTTR de 4 h. Da dos caminos para llegar a 99,95 % e indica cuál es más barato.
4. ¿Qué ocurre si los umbrales de subida y bajada del autoescalado están en 70 % y 65 %?
5. Un pico duplica el tráfico en 25 s y una instancia tarda 118 s en servir. ¿Qué tres respuestas quedan y cuál protege mejor a los usuarios en curso?

Referencias

- Amdahl, G. (1967). Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. AFIPS Conference Proceedings. <https://doi.org/10.1145/1465482.1465560>
- Gunther, N. (2007). Guerrilla Capacity Planning — ley universal de escalabilidad y ajuste de σ y κ.
- Beyer, B. et al., eds. (2016). Site Reliability Engineering, cap. 22 — degradación elegante y rechazo de carga. <https://sre.google/sre-book/addressing-cascading-failures/>
- Nygard, M. (2018). Release It!, 2.ª ed., caps. 4-5 — patrones de estabilidad frente a fallo en cascada.
- Fielding, R. y Reschke, J., eds. (2022). RFC 9110, sec. 15.5.29 — código 429 y cabecera Retry-After. <https://www.rfc-editor.org/rfc/rfc9110#status.429>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Evaluación — 016 Elasticidad, escalabilidad, disponibilidad y resiliencia

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 modelo-de-capacidad 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.

017 — Tenancy, cuentas, suscripciones, proyectos y jerarquías

Parte: 01 — Principios, estrategia y adopción cloud Nivel: inicial-intermedio · Horas estimadas: 4 Laboratorio: governance · Estado: EXECUTABLECORE

Propósito

Diseñar la jerarquía de recursos como el primer control de seguridad y de coste, no como una tarea administrativa. La frontera de cuenta es la única barrera que ningún error de permisos atraviesa, y las cuotas viven en ella: dos razones por las que esta decisión, tomada al principio, condiciona los siguientes cinco años.

Resultados de aprendizaje

Al finalizar podrás:

1. Traducir la jerarquía de los tres grandes proveedores a un modelo común de contenedor, política heredada y frontera de aislamiento.
2. Justificar la separación por cuenta frente a la separación por etiquetas usando radio de impacto y cuotas.
3. Explicar por qué una política heredada restrictiva no concede permisos y qué implica al depurar accesos.
4. Anticipar qué límites son por cuenta y cómo un vecino interno puede agotarlos.
5. Diseñar una estructura de organización que soporte crecimiento sin reorganizaciones destructivas.

Conceptos centrales

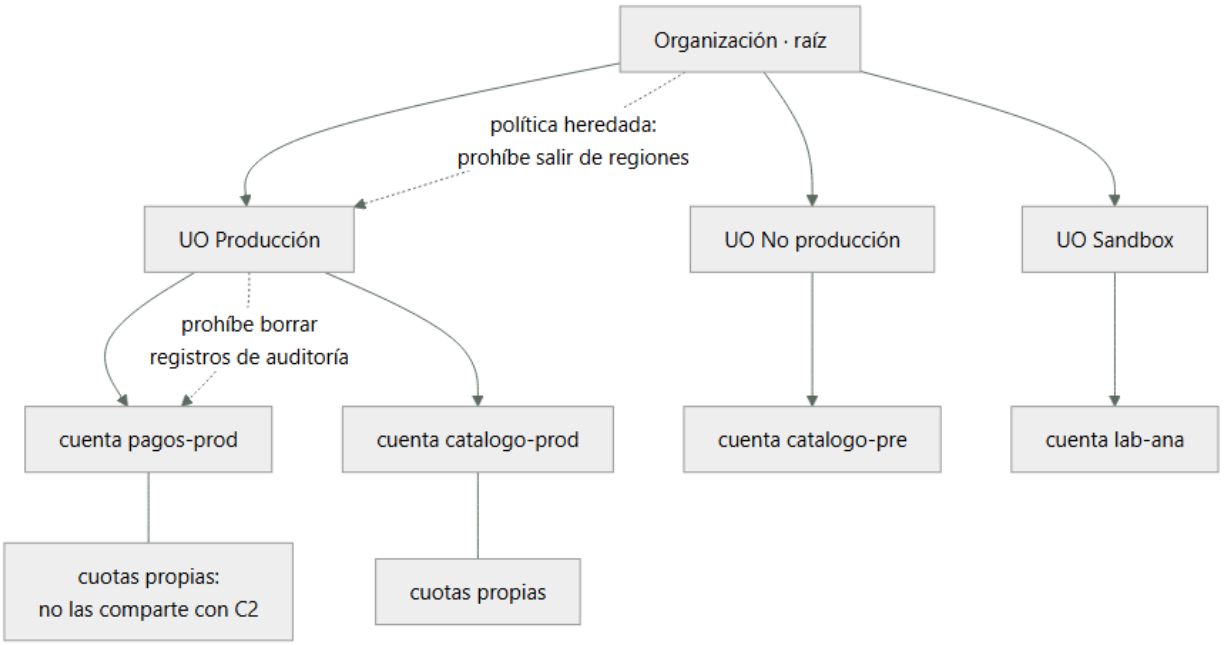
Concepto	Comprensión verificable
frontera de aislamiento	Límite que un permiso mal concedido no puede atravesar. En los tres grandes proveedores es la cuenta, suscripción o proyecto: dentro se puede fallar en la configuración; fuera hace falta un permiso explícito adicional.
política heredada	Restricción aplicada a un nodo superior de la jerarquía que acota lo máximo que se puede conceder debajo. No concede nada: solo limita el techo de lo que otras políticas pueden otorgar.
cuota	Límite de servicio asociado normalmente a la cuenta y la región. Es un recurso compartido entre todo lo que viva en esa cuenta, y por tanto una razón técnica —no organizativa— para separar.
etiqueta	Par clave-valor sobre un recurso, usado para atribución de coste y automatización. No es una frontera de seguridad: se puede modificar con permisos de escritura sobre el recurso.
unidad organizativa	Agrupador intermedio en la jerarquía que permite aplicar políticas a un conjunto de cuentas. Su diseño debe reflejar cómo se gobierna, no cómo está el organigrama, porque los organigramas cambian más rápido.

Modelo mental

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

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 modelo común bajo tres vocabularios

Los tres grandes proveedores implementan la misma idea con nombres distintos. Traducirla evita aprender tres veces lo mismo:

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

La fila en negrita es la que importa: es donde viven las cuotas, la facturación y el aislamiento real. Todo lo que esté por encima es organización; todo lo que esté por debajo es agrupación conveniente.

Dos diferencias que sí cambian el diseño:

- Azure tiene un nivel intermedio —el grupo de recursos— que agrupa recursos con ciclo de vida común dentro de una suscripción. No es frontera de aislamiento, pero sí unidad de borrado: eliminar el grupo elimina todo lo que contiene, lo que es útil para entornos efímeros y peligroso si se confunde con una frontera.
- Google Cloud permite anidar carpetas en varios niveles, mientras que AWS limita la profundidad de unidades organizativas a cinco. Diseñar jerarquías profundas es posible en uno y no en el otro.

2. Por qué la cuenta y no la etiqueta

La tentación inicial es usar una sola cuenta con etiquetas por entorno y equipo. Es más simple y es incorrecto por tres razones técnicas, no organizativas:

1. Radio de impacto. Una política de permisos mal escrita alcanza todo lo que hay en la cuenta. Con separación:

```
cuenta única:      1 error de permisos → producción y desarrollo
cuentas separadas: 1 error de permisos → un entorno de un producto
```

2. Cuotas. Casi todos los límites de servicio son por cuenta y región. Un entorno de pruebas que arranca 200 instancias para un experimento consume la cuota que producción necesitaría para escalar durante un pico. No es hipotético: es el

modo de fallo más frecuente de las cuentas compartidas, y no lo previene ningún permiso.

3. Atribución de coste. Una etiqueta puede faltar; una cuenta no. Si el 30 % de los recursos no está etiquetado, ese 30 % del gasto es inatribuible. La cuenta atribuye por construcción.

Y una razón que no es técnica pero pesa: las etiquetas se pueden cambiar con permisos de escritura sobre el recurso. Quien pueda modificar una instancia puede cambiar su etiqueta entorno: produccion a entorno: desarrollo. Una etiqueta no es una frontera de seguridad, es metadato.

Lo que sí resuelven bien las etiquetas: atribución fina dentro de una cuenta, automatización y ciclos de vida. Son complementarias, no alternativas.

3. Las políticas heredadas limitan, no conceden

Es la fuente de confusión más común al depurar accesos. Una política heredada define el techo de lo que se puede conceder por debajo; el permiso efectivo es la intersección:

```
permiso efectivo = política heredada ∩ permiso de identidad ∩ política de recurso
```

De ahí dos consecuencias:

1. Adjuntar una política heredada permisiva no da acceso a nadie. Si nadie tiene un permiso de identidad que lo conceda, no hay acceso.
2. Una denegación en cualquier nivel gana siempre. No hay forma de conceder por debajo lo que se denegó arriba, ni siquiera para el administrador de la cuenta.

Ejemplo de política heredada que impide salir de regiones aprobadas:

```
{
  "Effect": "Deny",
  "NotAction": ["iam:*", "organizations:*", "support:*"],
  "Resource": "*",
  "Condition": {"StringNotEquals": {"aws:RequestedRegion": ["us-east-1", "sa-east-1"]}}
}
```

El NotAction con servicios globales no es un detalle: IAM y facturación no tienen región, así que sin exceptuarlos la política bloquea la propia administración de la cuenta. Es el error que convierte un guardrail en una cuenta inutilizable.

Al depurar «no tengo permiso» hay que revisar los tres niveles, y en ese orden: primero si algo lo deniega arriba —porque ningún cambio abajo lo arreglará—, después el permiso de identidad, y por último la política del recurso.

4. Diseñar la jerarquía por gobierno, no por organigrama

El error estructural más caro: modelar unidades organizativas según el organigrama. Los organigramas se reorganizan cada 12-18 meses; mover cuentas entre unidades cambia las políticas heredadas que se les aplican, y eso puede romper cargas en producción.

El criterio robusto es agrupar por qué política se aplica, que cambia mucho más despacio:

```
raíz
├── Producción            políticas estrictas: regiones, borrado, cifrado obligatorio
│   ├── pagos-prod
│   └── catalogo-prod
├── No producción         políticas medias: regiones, límite de gasto
│   ├── catalogo-pre
│   └── pagos-pre
├── Sandbox               políticas laxas, presupuesto duro, borrado automático
│   └── lab-<persona>
└── Infraestructura       cuentas compartidas: red, registro de imágenes, auditoría
    ├── red-central
    └── auditoria
```

Si mañana el equipo de pagos se fusiona con el de catálogo, no hay que mover ninguna cuenta: las políticas que se les aplican no dependen de quién las gestiona.

Tres cuentas que conviene separar desde el principio:

- Auditoría: recibe los registros de todas las demás y solo permite escritura desde ellas. Si un atacante compromete una cuenta, no puede borrar la evidencia de otra.

- Red central: la conectividad compartida, para que su ciclo de vida no dependa de un producto.
- Gestión de identidad: donde viven los roles que se asumen hacia las demás.

Y una regla operativa: la cuenta raíz de la organización no debe alojar cargas de trabajo. Es la única que no puede tener políticas heredadas por encima.

5. Cuotas: el límite que se descubre en el peor momento

Las cuotas son el aspecto menos visible de la frontera de cuenta y el que más incidentes causa, porque solo se manifiestan bajo carga.

Categorías, con su comportamiento:

Tipo	Ejemplo	¿Ampliable?
Blanda por cuenta	Número de instancias por región	Sí, con solicitud y días de espera
Dura por cuenta	Cuentas por organización	No, o con excepción especial
Por recurso	Conexiones por base de datos	Depende del tamaño elegido
De tasa	Peticiones por segundo a una API	A veces, con ráfaga limitada

Dos comportamientos que sorprenden:

La ampliación no es instantánea. Solicitar más cuota puede tardar de horas a días. Si se descubre durante un incidente, la cuota no es una palanca disponible: hay que haberla ampliado antes. Por eso forma parte del plan de capacidad de la parte 21 y no de la respuesta a incidentes.

Las cuotas de tasa se aplican al plano de control. Un bucle que consulta el estado de un recurso cada segundo puede agotar el límite de llamadas a la API de gestión y provocar que fallen los despliegues de todo lo demás en esa cuenta, mientras las aplicaciones siguen funcionando con normalidad. El síntoma —«no puedo desplegar pero el servicio va bien»— desconcierta hasta que se conoce la causa.

La consecuencia de diseño: monitoriza el consumo de cuota como una métrica más, con alerta al 80 %. Es de las pocas métricas que predicen un fallo con días de antelación.

Ejemplo trabajado

CloudShop opera todo en una cuenta con etiquetas por entorno. Un despliegue de pruebas deja producción sin capacidad durante el Cyber Monday. Se reconstruye qué pasó y se rediseña.

La secuencia:

```
14:02  un experimento de carga en "desarrollo" arranca 180 instancias
14:09  el autoescalado de producción intenta subir de 16 a 24
14:09  falla: VcpuLimitExceeded
14:11  latencia p95 de producción: 91 ms → 2.400 ms
14:22  alguien identifica el experimento y lo detiene
14:26  producción recupera capacidad
```

24 minutos de degradación. Ningún permiso estaba mal: el experimento tenía derecho a arrancar instancias en su entorno. El problema es que «su entorno» era una etiqueta y la cuota es de la cuenta.

```
$ aws service-quotas get-service-quota --service-code ec2 --quota-code L-1216C47A \
    --query 'Quota.[QuotaName,Value]' --output text
Running On-Demand Standard instances    256
```

```
256 vCPU de cuota total en la región
producción en régimen              64 vCPU (16 instancias × 4)
producción en pico necesario       96 vCPU (24 × 4)
experimento                       180 vCPU
64 + 180 = 244 → quedan 12 vCPU: solo 3 de las 8 instancias que pedía
```

La etiqueta no separaba nada porque las cuotas no leen etiquetas.

Rediseño con separación por cuenta:

```
raíz
■■■ Producción          SCP: solo us-east-1 y sa-east-1; prohíbe borrar logs
```

```
■   ■■■ cloudshop-prod          cuota propia: 256 vCPU
■■■ No producción        SCP: mismas regiones; presupuesto 2.000 USD
■   ■■■ cloudshop-pre           cuota propia: 128 vCPU
■■■ Sandbox              SCP: presupuesto 200 USD; borrado a los 7 días
■   ■■■ lab-*                   cuota propia: 64 vCPU
■■■ Infraestructura
    ■■■ red-central
    ■■■ auditoria               recibe logs; nadie más puede borrarlos
```

Se verifica que la política heredada no rompe la administración:

```
$ aws organizations describe-policy --policy-id p-regiones --query 'Policy.Content' \
  | jq -r '.Statement[0].NotAction'
["iam:*", "organizations:*", "support:*", "budgets:*", "cloudfront:*"]
```

Los servicios globales quedan exceptuados; sin eso, la política habría bloqueado la gestión de identidades de todas las cuentas hijas.

Y se añade la métrica que habría avisado con días de antelación:

```
$ aws cloudwatch put-metric-alarm --alarm-name cuota-vcpu-prod \
    --metric-name ResourceCount --namespace AWS/Usage \
    --threshold 204 --comparison-operator GreaterThanThreshold   # 80 % de 256
```

Resultado en el siguiente evento de alto tráfico:

```
cuota compartida             sí (256 total)   no (256 solo prod)
degradación por vecino       24 min           0 min
gasto de sandbox atribuible  ~30 % sin asignar  100 % por cuenta
```

La separación no impidió que alguien arrancara 180 instancias: impidió que eso afectara a producción. Es exactamente el principio de radio de impacto de la clase 010, aplicado a la frontera que las cuotas respetan.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-01-cloud-principles-strategy-adoption/017-tenancy-cuentas-suscripciones-proyectos-y-jerarquias/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 arbol-de-recursos 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 arbol-de-recursos para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.
- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.

- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
Un entorno de pruebas deja sin capacidad a producción	Las cuotas son por cuenta y las etiquetas no las separan	Separa entornos por cuenta, suscripción o proyecto; la etiqueta no es una frontera.
Una política heredada bloquea la administración de la cuenta	La condición de región alcanzó a servicios globales como IAM y facturación, que no tienen región	Excluye los servicios globales con NotAction al restringir por región.
Se concede una política heredada permisiva y nadie obtiene acceso	Las políticas heredadas limitan el techo, no conceden permisos	Concede con permisos de identidad; usa la herencia solo como barrera superior.
Una reorganización de equipos rompe cargas en producción	La jerarquía se modeló según el organigrama y mover cuentas cambió sus políticas heredadas	Agrupa por régimen de gobierno —producción, no producción, sandbox—, que cambia mucho más despacio.
No se puede desplegar aunque las aplicaciones funcionen bien	Se agotó la cuota de tasa del plano de control, normalmente por un bucle de consulta	Monitoriza el consumo de cuota con alerta al 80 % y aplica retroceso en los bucles de consulta.

■ Seguridad, ética y costo

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

■ Preguntas de comprobación

1. ¿Cuál es la frontera de aislamiento real en AWS, Azure y Google Cloud, y por qué las etiquetas no la sustituyen?
2. Un desarrollador arranca 180 instancias en «desarrollo» y producción no puede escalar. ¿Qué mecanismo falló y cuál no?
3. Si adjuntas una política heredada que permite todo, ¿quién obtiene acceso? Justifica con la fórmula del permiso efectivo.
4. ¿Por qué restringir por región puede inutilizar una cuenta, y qué excepción lo evita?
5. ¿Por qué la ampliación de cuota no sirve como respuesta durante un incidente?

Referencias

- AWS (2024). Organizations: service control policies — evaluación de políticas heredadas y servicios globales. <https://docs.aws.amazon.com/organizations/latest/userguide/orgsmanagepoliciesscps.html>
- Microsoft (2024). Cloud Adoption Framework: resource organization — grupos de administración, suscripciones y grupos de recursos. <https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-setup-guide/organize-resources>
- Google Cloud (2024). Resource hierarchy — organización, carpetas, proyectos y herencia de políticas. <https://cloud.google.com/resource-manager/docs/cloud-platform-resource-hierarchy>
- AWS (2024). Organizing your AWS environment using multiple accounts — criterios de separación por cuenta. <https://docs.aws.amazon.com/whitepapers/latest/organizing-your-aws-environment/organizing-your-aws-environment.html>
- Beyer, B. et al., eds. (2018). The Site Reliability Workbook, cap. 11 — gestión de capacidad y cuotas. <https://sre.google/workbook/managing-load/>

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

Descargar: Parte 01 en PDF · Manual integral

Anterior	Índice	Siguiente
← 016 · Elasticidad, escalabilidad, disponibilidad y resiliencia	Parte 01 · Programa	018 · Identidad, roles, políticas y federación →

Evaluación — 017 Tenancy, cuentas, suscripciones, proyectos y jerarquías

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 arbol-de-recursos 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.

018 — Identidad, roles, políticas y federación

Parte: 01 — Principios, estrategia y adopción cloud Nivel: inicial-intermedio · Horas estimadas: 4 Laboratorio: iam · Estado: EXECUTABLECORE

Propósito

Construir el modelo de identidad que sostiene toda la seguridad cloud: cómo se evalúa una petición, por qué las credenciales de larga vida son el vector de brecha más frecuente y cómo eliminarlas con federación. Es la clase que hace posible que en las partes 08 y 17 un pipeline despliegue en producción sin que exista ningún secreto que robar.

Resultados de aprendizaje

Al finalizar podrás:

1. Reproducir el orden de evaluación de una petición y predecir el resultado ante políticas en conflicto.
2. Sustituir una clave de acceso permanente por una identidad federada con credenciales temporales.
3. Escribir una condición de confianza que impida que otro repositorio asuma tu rol.
4. Distinguir autenticación, autorización y auditoría, y qué registro responde a cada pregunta forense.
5. Aplicar privilegio mínimo de forma iterativa, partiendo de lo que el uso real demuestra necesario.

Conceptos centrales

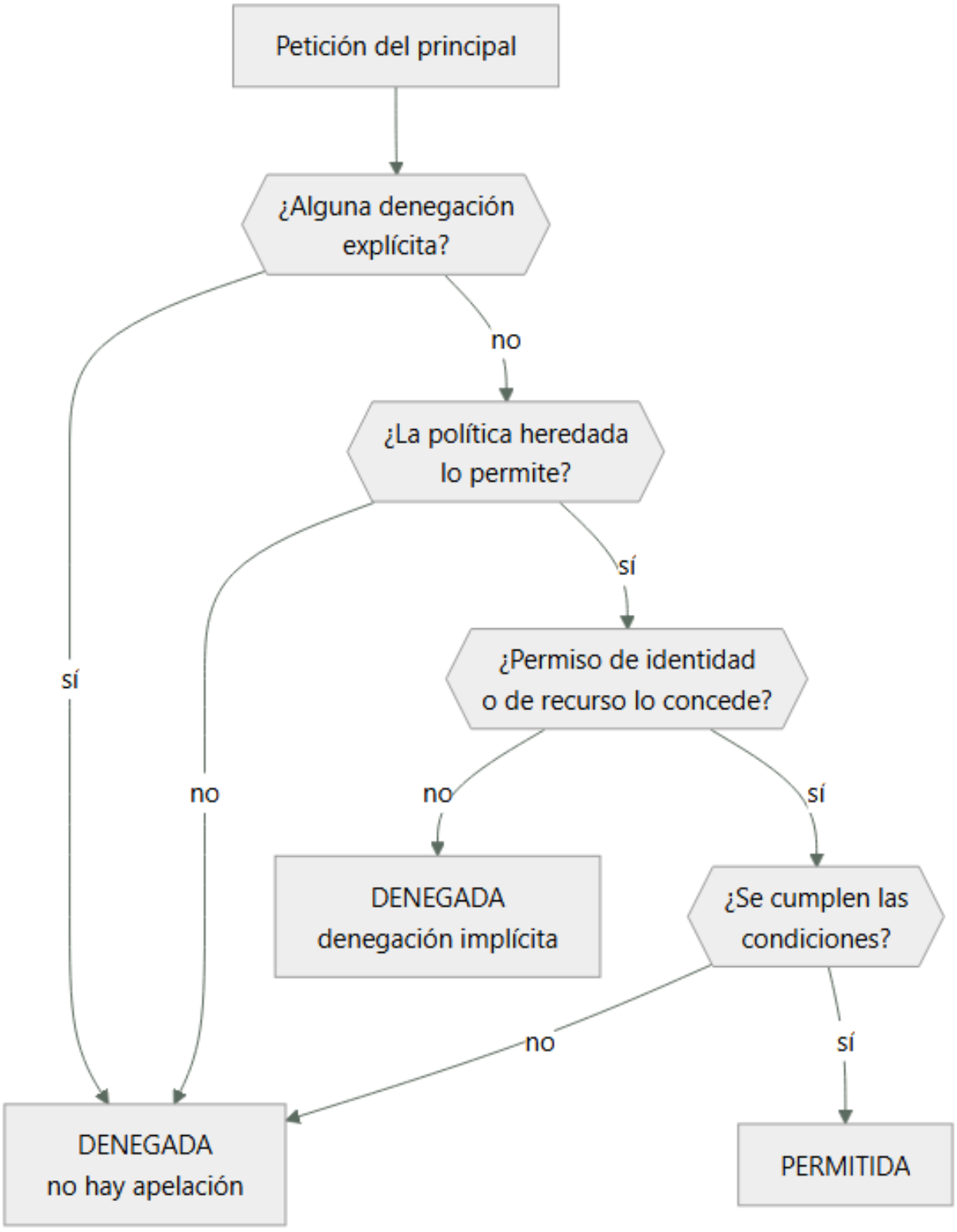
Concepto	Comprensión verificable
principal	Entidad que hace una petición: usuario, rol asumido, servicio o carga de trabajo federada. Lo que importa no es quién es, sino qué permisos tiene en el momento exacto de la llamada.
rol	Conjunto de permisos que se asume temporalmente, sin credenciales propias permanentes. Al asumirlo se reciben credenciales con caducidad, lo que acota la ventana de uso de una fuga.
política de confianza	Documento que declara quién puede asumir un rol. Es la puerta: los permisos dicen qué se puede hacer una vez dentro, la confianza dice quién entra.
federación de identidad	Mecanismo por el que un proveedor externo de identidad emite un token que la nube acepta a cambio de credenciales temporales. Elimina la necesidad de almacenar secretos de larga vida.
privilegio mínimo	Conceder solo lo que el uso demuestra necesario. No es un estado inicial sino un proceso: se parte de lo observado y se recorta, porque nadie acierta la lista completa por adelantado.

Modelo mental

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

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 orden de evaluación no es negociable

Toda petición pasa por la misma secuencia, y conocerla convierte «no tengo permiso» de misterio en diagnóstico de tres pasos:

1. Denegación explícita: si algo la deniega, termina ahí. Ninguna concesión posterior la revierte.

2. Barreras heredadas: políticas de organización que acotan el techo. Si no lo permiten, se deniega aunque el permiso de identidad lo conceda.
3. Concesión: algún permiso de identidad o de recurso debe permitirlo explícitamente.
4. Condiciones: origen, hora, etiquetas, MFA, red. Si no se cumplen, se deniega.

Sin una concesión explícita el resultado es denegación implícita, que es la respuesta por defecto. Esa asimetría —todo prohibido salvo lo concedido, y lo denegado no se puede reconceder— es lo que hace el modelo analizable.

La consecuencia práctica al depurar: empieza por buscar denegaciones, no concesiones. Añadir permisos a una identidad que choca contra una denegación de organización no cambia nada, y es donde se pierden las tardes.

El caso de la política de recurso merece atención: cuando el recurso está en otra cuenta, hacen falta las dos —la de identidad en la cuenta que llama y la de recurso en la que responde—. Una sola no basta, y el mensaje de error no siempre distingue cuál falta.

2. Las credenciales de larga vida son el problema, no la solución

Una clave de acceso permanente tiene tres propiedades que la convierten en el vector más explotado:

1. No caduca: sigue siendo válida años después de filtrarse.
2. Es portable: funciona desde cualquier lugar del mundo.
3. Es silenciosa: su uso indebido es indistinguible del legítimo sin análisis de comportamiento.

Los escáneres automáticos de repositorios públicos localizan credenciales expuestas en minutos. El patrón es conocido: un desarrollador sube por error un fichero de configuración, lo borra en el commit siguiente, y la credencial sigue en el historial de Git —por lo visto en la clase 003, los objetos no desaparecen— y ya fue recogida.

La alternativa es no tener nada que robar:

```
clave permanente:  AKIA... + secreto        válida hasta que alguien la revoque
credencial temporal: token de 1 h            inútil después, y ligada a condiciones
```

La jerarquía de preferencia, de mejor a peor:

Mecanismo	Vida	Dónde vive el secreto
Identidad de carga (metadata)	minutos	En ningún sitio
Federación OIDC	1 h	En ningún sitio: se intercambia un token firmado
Rol asumido con MFA	1-12 h	En la sesión
Clave permanente en gestor de secretos	indefinida	En el gestor
Clave permanente en variable de entorno	indefinida	En todas partes

Las dos primeras son cualitativamente distintas: no hay secreto que filtrar. Las demás solo reducen la probabilidad o la ventana.

3. Federación OIDC: desplegar sin secretos

El mecanismo que elimina las claves de los pipelines. El flujo, sin secretos compartidos en ningún punto:

```
1. El pipeline pide a su plataforma un token OIDC firmado que declara
   quién es: repositorio, rama, entorno, ejecución.
2. Lo presenta al proveedor cloud pidiendo asumir un rol.
3. El proveedor verifica la firma contra las claves públicas de la plataforma
   y comprueba que las declaraciones cumplen la política de confianza.
4. Devuelve credenciales temporales de 1 hora.
```

La política de confianza es donde se juega la seguridad, y donde se comete el error más peligroso:

```
{
  "Effect": "Allow",
  "Principal": {"Federated": "arn:aws:iam::123456789012:oidc-provider/token.actions.githubusercontent.
com"},
  "Action": "sts:AssumeRoleWithWebIdentity",
  "Condition": {
```

```
    "StringEquals": {
      "token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
      "token.actions.githubusercontent.com:sub": "repo:miorg/cloudshop:ref:refs/heads/main"
    }
  }
}
```

La condición sobre sub es obligatoria y debe ser precisa. Los dos errores frecuentes:

```
"sub": "repo:miorg/*"          → cualquier repositorio de tu organización
sin condición sobre sub        → CUALQUIER repositorio de GitHub del mundo
```

El segundo caso es una brecha total: basta con que alguien cree un repositorio público, ejecute una acción y pida asumir tu rol. La firma será válida porque viene de la misma plataforma; lo único que lo distingue es la declaración sub.

Usar StringLike con comodines requiere cuidado adicional: repo:miorg/cloudshop: permite cualquier rama y cualquier entorno, incluida una rama que un colaborador externo pueda crear en un fork con permisos de escritura.

4. Privilegio mínimo es un proceso, no un estado

Nadie acierta la lista exacta de permisos por adelantado. Intentarlo produce uno de dos resultados: o se concede de más «por si acaso», o se bloquea el trabajo y alguien acaba adjuntando la política de administrador «temporalmente».

El método que funciona parte de lo observado:

```
1. Conceder amplio en un entorno NO productivo, con registro activado.
2. Ejercitar todos los caminos: éxito, error, reintento, borrado.
3. Extraer del registro los permisos realmente usados.
4. Generar la política a partir de esa lista.
5. Validar en preproducción y recortar lo que sobre.
6. Repetir cada trimestre: los permisos no usados en 90 días se retiran.
```

El paso 2 es el que se olvida y el que causa incidentes: los caminos de error usan permisos distintos. Un rol que puede escribir en una cola pero no en su cola de mensajes fallidos funciona perfectamente hasta el primer fallo, y entonces pierde mensajes en silencio.

Los proveedores ofrecen herramientas para el paso 3: analizadores que generan políticas a partir de la actividad registrada, e informes del último acceso por servicio que permiten podar sin adivinar.

Dos límites honestos del método:

- Un permiso no usado en 90 días puede ser el de la recuperación anual ante desastres. Antes de retirar hay que comprobar contra los runbooks, no solo contra el registro.
- Los permisos de solo lectura no son inocuos. s3:GetObject sobre un bucket de facturas es exactamente la brecha de la clase 010. «Solo lectura» describe el verbo, no el impacto.

5. Autenticación, autorización y auditoría responden a preguntas distintas

Las tres se confunden y cada una deja rastro en un sitio diferente. Ante un incidente, saber cuál preguntar ahorra horas:

Pregunta forense	Concepto	Dónde mirar
¿Quién era?	Autenticación	Registro del proveedor de identidad
¿Qué podía hacer?	Autorización	Políticas vigentes en ese momento
¿Qué hizo?	Auditoría	Registro de llamadas a la API

La segunda fila esconde una dificultad real: las políticas cambian. Saber qué permisos tenía un principal en el instante del incidente exige historial de configuración, no el estado actual. Sin él, la reconstrucción es una conjetura.

Y la tercera tiene un requisito que la clase 017 ya anticipó: el registro de auditoría debe vivir en otra cuenta, con escritura permitida y borrado prohibido para todas las demás. Un atacante que compromete una cuenta y puede borrar su propio rastro convierte la auditoría en decorativa.

Tres señales que conviene alertar desde el primer día, porque preceden a casi todo incidente:

- Uso de la identidad raíz de la cuenta.

- Creación de una clave de acceso permanente.
- Cambio en una política de confianza —es cómo se abre la puerta de par en par.

La tercera es la menos vigilada y la más peligrosa: modificar una condición sub de una línea puede convertir un rol seguro en uno asumible por cualquiera, y no genera ninguna alerta si nadie la configuró.

Ejemplo trabajado

El pipeline de CloudShop despliega en producción usando una clave de acceso permanente guardada como secreto del repositorio. Se migra a federación OIDC.

Estado inicial y su riesgo:

```
$ aws iam list-access-keys --user-name ci-deploy
{"AccessKeyMetadata": [{"AccessKeyId": "AKIA...", "CreateDate": "2023-03-14", "Status": "Active"}]}
$ aws iam get-user --user-name ci-deploy --query 'User.PasswordLastUsed'
null
```

```
antigüedad de la clave       2 años y 5 meses
rotaciones                   0
copias conocidas             secreto del repo, portátil de 2 personas, un runbook
ventana de uso si se filtra  ilimitada
```

Paso 1 — registrar el proveedor OIDC (una sola vez por cuenta):

```
$ aws iam create-open-id-connect-provider \
    --url https://token.actions.githubusercontent.com \
    --client-id-list sts.amazonaws.com
```

Paso 2 — política de confianza, con la condición precisa. Primero se escribe la versión insegura para verla y descartarla:

```
"Condition": {"StringEquals": {"...:aud": "sts.amazonaws.com"}}
```

Se prueba desde un repositorio de laboratorio ajeno a la organización:

```
$ (desde otro repo cualquiera) aws sts assume-role-with-web-identity ...
→ ÉXITO. Cualquier repositorio de GitHub podía asumir el rol.
```

Se corrige acotando el sujeto al repositorio, la rama y el entorno:

```
"Condition": {
  "StringEquals": {
    "token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
    "token.actions.githubusercontent.com:sub": "repo:miorg/cloudshop:environment:produccion"
  }
}
```

Se repite la prueba negativa:

```
desde otro repositorio            → AccessDenied  ✓
desde miorg/cloudshop, rama pre   → AccessDenied  ✓
desde miorg/cloudshop, prod       → credenciales de 1 h  ✓
```

Paso 3 — recortar los permisos con el uso real. El rol tenía PowerUserAccess. Se extrae lo ejercitado en 30 días de despliegues:

```
$ aws accessanalyzer start-policy-generation --policy-generation-details ...
$ aws iam get-service-last-accessed-details --job-id ... \
    --query 'ServicesLastAccessed[?TotalAuthenticatedEntities>`0`].ServiceNamespace'
["s3", "cloudfront", "ecs", "ecr", "logs"]
```

```
permisos concedidos antes   ~8.000 acciones (PowerUserAccess)
servicios usados en 30 días  5
acciones usadas             34
política final              34 acciones sobre 6 recursos concretos
```

Se ejercitan también los caminos de error antes de aplicar —el paso que se olvida—: un despliegue que falla necesita ecs:StopTask y logs:PutLogEvents sobre el grupo de errores, que no aparecían en los despliegues correctos.

Paso 4 — retirar la clave y verificar que nada la usa:

```
$ aws iam update-access-key --user-name ci-deploy --access-key-id AKIA... --status Inactive
# 7 días de observación sin fallos
$ aws iam delete-access-key --user-name ci-deploy --access-key-id AKIA...
$ aws iam delete-user --user-name ci-deploy
```

Resultado:

```
credencial               permanente, 2,5 años   token de 1 h
secretos almacenados     4 copias conocidas     0
permisos                 ~8.000 acciones        34 acciones
superficie si se filtra  ilimitada              1 h, solo desde ese repo y entorno
```

El cambio decisivo no fue reducir permisos: fue que ya no existe nada que filtrar. Un atacante que consiga el token tiene una hora, y solo desde el contexto exacto que la condición sub describe.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-01-cloud-principles-strategy-adoption/018-identidad-roles-politicas-y-federacion/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 matriz-rbac 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 matriz-rbac 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
Cualquier repositorio externo puede asumir el rol de despliegue	La política de confianza no condiciona la declaración sub	Condiciona siempre sobre repositorio, rama o entorno; sin ello la firma válida basta para entrar.
Se añaden permisos y el acceso sigue denegado	Una denegación explícita o una barrera heredada gana sobre cualquier concesión	Al depurar, busca primero denegaciones y políticas de organización, no concesiones.
El pipeline funciona hasta que un despliegue falla, y entonces pierde información	Los caminos de error usan permisos distintos que no se ejercitaron al generar la política	Ejercita éxito, error, reintento y borrado antes de extraer la política del registro.
Tras un incidente no se puede saber qué permisos tenía el principal	Solo se conserva el estado actual de las políticas, no su historial	Activa historial de configuración; la autorización de ayer no se deduce de la política de hoy.
Un rol de solo lectura provoca una fuga de datos	Se asumió que leer es inocuo; el impacto depende del recurso, no del verbo	Clasifica por sensibilidad del dato: leer facturas no es equivalente a leer métricas.

■ 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. Una identidad tiene un permiso que concede la acción y una barrera heredada que no la incluye. ¿Cuál gana y por qué?
2. ¿Qué tiene que ocurrir para que un repositorio ajeno pueda asumir tu rol federado, y qué línea lo impide?
3. ¿Por qué generar una política solo con los permisos de los despliegues exitosos deja un fallo latente?
4. ¿Qué registro consultas para responder «qué podía hacer» frente a «qué hizo»?
5. Nombra tres eventos de identidad que conviene alertar desde el primer día y explica por qué el tercero es el menos vigilado.

Referencias

- AWS (2024). Policy evaluation logic — orden de evaluación, denegación explícita e implícita. <https://docs.aws.amazon.com/IAM/latest/UserGuide/referencepoliciesevaluation-logic.html>
- GitHub (2024). About security hardening with OpenID Connect — declaraciones del token y condiciones de confianza. <https://docs.github.com/en/actions/deployment/security-hardening-your-deployments/about-security-hardening-with-openid-connect>
- Sakimura, N. et al. (2014). OpenID Connect Core 1.0 — semántica de las declaraciones sub y aud. <https://openid.net/specs/openid-connect-core-10.html>
- NIST (2020). Zero Trust Architecture, SP 800-207 — verificación por petición y credenciales de vida corta. <https://doi.org/10.6028/NIST.SP.800-207>
- Dotson, C. (2023). Practical Cloud Security, 2.ª ed., caps. 4-5 — gestión de identidad y privilegio mínimo iterativo.
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 01 en PDF · Manual integral

Evaluación — 018 Identidad, roles, políticas y federación

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-rbac 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.

019 — Modelo de responsabilidad compartida por servicio

Parte: 01 — Principios, estrategia y adopción cloud Nivel: inicial-intermedio · Horas estimadas: 4 Laboratorio: security · Estado: EXECUTABLECORE

Propósito

Convertir el modelo de responsabilidad compartida de un diagrama de márketing en una matriz operativa por servicio, con controles nombrados y un responsable por cada uno. La clase 010 estableció el principio; aquí se aplica servicio a servicio, que es donde aparecen las zonas grises que nadie reclama hasta que hay un incidente.

Resultados de aprendizaje

Al finalizar podrás:

1. Construir una matriz RACI de controles para un servicio concreto, distinguiendo quién ejecuta y quién responde.
2. Localizar las zonas grises —controles que ambas partes creen del otro— antes de que las encuentre un auditor.
3. Comparar el reparto real entre una base de datos autogestionada y una gestionada, con el detalle de qué se hereda.
4. Interpretar un informe de cumplimiento del proveedor sabiendo qué controles deja explícitamente al cliente.
5. Verificar un control heredado en vez de asumirlo, distinguiendo evidencia de declaración.

Conceptos centrales

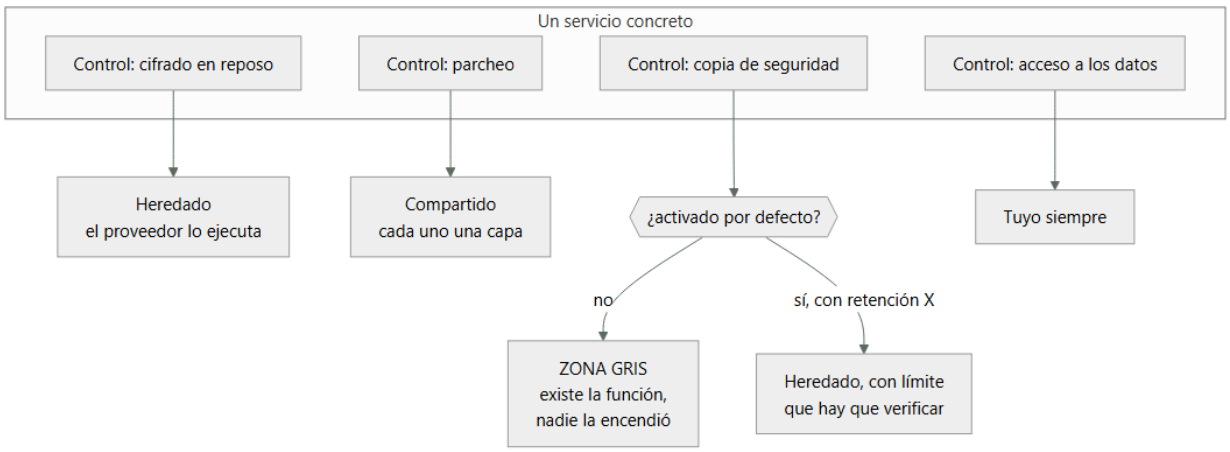
Concepto	Comprensión verificable
control heredado	Control que el proveedor ejecuta y del que obtienes el beneficio sin implementarlo. Se hereda el control, no la responsabilidad de comprobar que aplica a tu configuración.
control compartido	Control en el que ambas partes hacen algo distinto sobre la misma capa. El parcheo en IaaS es el ejemplo canónico: el proveedor parchea el hipervisor y tú el sistema operativo huésped.
zona gris	Control que ninguna de las dos partes ejecuta porque cada una supone que lo hace la otra. Es donde se concentran los hallazgos de auditoría y los incidentes evitables.
RACI	Reparto de un control en cuatro papeles: quién lo ejecuta, quién responde por él, a quién se consulta y a quién se informa. La distinción entre ejecutar y responder es la que evita las zonas grises.
informe de cumplimiento	Auditoría independiente sobre los controles del proveedor. Su sección de controles complementarios del usuario enumera lo que deja explícitamente a tu cargo, y es la parte que casi nadie lee.

Modelo mental

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

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. Tres clases de control, no dos

La presentación habitual reparte los controles en dos columnas y esa simplificación es la que crea las zonas grises. En la práctica hay tres clases:

Clase	Quién ejecuta	Tu obligación
Heredado	Solo el proveedor	Verificar que aplica a tu configuración y obtener la evidencia
Compartido	Ambos, en capas distintas	Ejecutar tu capa y saber dónde está la frontera
Propio	Solo tú	Todo

La columna derecha de la primera fila es la que se ignora. Heredar la seguridad física del centro de datos no exime de comprobar que la región que usas está incluida en el informe de cumplimiento; heredar el cifrado de disco no exime de comprobar que está activado en tu recurso.

El ejemplo canónico de control compartido es el parcheo en IaaS:

```
hipervisor y firmware  → proveedor, sin aviso ni ventana que tú controles
sistema operativo      → tú
bibliotecas del sistema→ tú
runtime del lenguaje   → tú
dependencias de la app → tú
```

Una vulnerabilidad en el hipervisor la resuelve el proveedor y puede implicar un reinicio de tu instancia con preaviso corto. Una en glibc es tuya, aunque el sistema operativo venga de una imagen del proveedor. La frontera es el hipervisor, no la imagen.

2. Autogestionado frente a gestionado: qué cambia de verdad

Comparar una base de datos instalada por ti con una gestionada, control a control, revela que el reparto no se mueve en bloque:

Control	En una VM	Servicio gestionado
Parche del motor	Tú	Proveedor, en ventana que tú eliges
Cifrado en reposo	Tú lo configuras	Heredado, pero tú lo activas
Copia de seguridad	Tú	Proveedor ejecuta, tú fijas retención
Prueba de restauración	Tú	Tú — nunca la hace el proveedor
Réplica entre zonas	Tú	Proveedor, si lo activas
Esquema y consultas	Tú	Tú
Permisos de acceso	Tú	Tú

Control	En una VM	Servicio gestionado
Clasificación del dato	Tú	Tú

Dos filas concentran casi todos los incidentes:

La prueba de restauración nunca es del proveedor. Que exista una copia no demuestra que se pueda restaurar, ni cuánto tarda. Una copia sin restauración probada es una suposición, y el momento de descubrirlo no puede ser el incidente. Es el control que más se hereda por error.

Casi todo lo gestionado depende de que lo actives. Cifrado, réplica, retención extendida, registro de auditoría: el proveedor los ofrece, no los impone. La diferencia entre «el servicio soporta X» y «mi instancia tiene X activado» es exactamente el espacio donde vive la zona gris.

Y hay un control que empeora al pasar a gestionado: el acceso del operador del proveedor a los datos. Es un riesgo aceptable y contractualmente acotado, pero existe y hay que nombrarlo en el análisis, no descubrirlo en una auditoría.

3. RACI: separar quién ejecuta de quién responde

La distinción entre ejecutar y responder es la que cierra las zonas grises. Un control puede ejecutarlo el proveedor y seguir siendo tu responsabilidad ante el regulador y ante tus clientes.

Para la copia de seguridad de una base de datos gestionada:

Actividad	Ejecuta	Responde	Evidencia
Tomar la copia	Proveedor	Cliente	Registro de instantáneas
Fijar la retención	Cliente	Cliente	Configuración versionada
Cifrar la copia	Proveedor	Cliente	Atributo del recurso
Probar la restauración	Cliente	Cliente	Informe con RTO medido
Verificar la integridad	Cliente	Cliente	Suma de comprobación tras restaurar

La columna «Responde» es siempre el cliente. Eso no es una carga retórica: significa que ante un fallo de copia no puedes trasladar la consecuencia al proveedor más allá del crédito del SLA, como se calculó en la clase 010.

La columna «Evidencia» es la que convierte la matriz en algo auditable. Un control sin evidencia es una intención: «el proveedor hace copias» no es evidencia; el identificador de la última instantánea y la fecha de la última restauración probada sí lo son.

Hacer esta tabla para los cinco o seis servicios principales de una plataforma lleva unas horas y localiza las zonas grises antes de que las encuentre un auditor o un incidente.

4. Leer un informe de cumplimiento por su última sección

Los informes de auditoría del proveedor —SOC 2 Tipo II, ISO 27001, y equivalentes— se usan mal: se archivan como prueba de que «el proveedor es seguro». Lo que contienen es más útil y más incómodo.

Qué mirar, en orden:

1. Alcance: qué servicios y qué regiones cubre. Un servicio nuevo o una región recién abierta pueden estar fuera, y entonces no hay nada heredado.
2. Periodo: los informes Tipo II cubren un intervalo pasado, típicamente 6 o 12 meses. Un informe cerrado hace ocho meses no dice nada de lo ocurrido después.
3. Excepciones: controles con desviaciones detectadas durante la auditoría, y qué se hizo.
4. Controles complementarios del usuario: la sección decisiva. Enumera lo que el proveedor deja explícitamente a tu cargo para que sus controles funcionen.

La cuarta sección suele contener afirmaciones muy concretas: que configures el cifrado, que gestiones el ciclo de vida de las credenciales, que revises los permisos, que actives el registro. Son exactamente los controles que después aparecen como hallazgos.

Un informe del proveedor no te certifica. Si tu organización necesita cumplir una norma, necesitas tu propia auditoría; el informe del proveedor cubre su parte del reparto y por eso incluye la lista de lo que espera de ti.

5. Verificar lo heredado en vez de asumirlo

Heredar un control no exime de comprobar que aplica. La diferencia entre declaración y evidencia:

```
# Declaración: la documentación dice que el servicio soporta cifrado
# Evidencia: mi recurso concreto lo tiene activado
$ aws rds describe-db-instances --db-instance-identifier cloudshop-prod \
    --query 'DBInstances[0].[StorageEncrypted,KmsKeyId,BackupRetentionPeriod,MultiAZ]'
[true, "arn:aws:kms:...:key/1a2b3c", 7, true]
```

Cuatro comprobaciones en una línea, y cada una podría haber sido false sin que nada fallara visiblemente. La retención de 7 días es un dato, no una validación: hay que contrastarla con el requisito. Si el requisito legal es 30 días, el control heredado no cumple aunque esté activado.

El patrón general, aplicable a cualquier control heredado:

```
1. ¿Qué afirma el proveedor?          documentación e informe de cumplimiento
2. ¿Aplica a mi recurso?              consulta al plano de control
3. ¿Cumple mi requisito concreto?     comparación con el número exigido
4. ¿Cómo lo demuestro dentro de un año? evidencia versionada y automatizada
```

El paso 4 es el que separa una comprobación puntual de un control: si la verificación no está automatizada, dentro de seis meses nadie sabrá si sigue siendo cierta. Convertir estas comprobaciones en pruebas que fallan el despliegue —lo que en la parte 11 será política como código— es lo que hace que un control heredado sea auditable de forma continua y no una foto de un día.

Ejemplo trabajado

Una auditoría de CloudShop devuelve un hallazgo: «no se evidencia la capacidad de restauración de la base de datos de pedidos». El equipo responde que el proveedor hace copias automáticas. Se construye la matriz para verlo.

Estado declarado frente a estado verificado:

```
$ aws rds describe-db-instances --db-instance-identifier cloudshop-prod \
    --query 'DBInstances[0].[BackupRetentionPeriod,PreferredBackupWindow,StorageEncrypted]'
[7, "07:00-07:30", true]
$ aws rds describe-db-snapshots --db-instance-identifier cloudshop-prod \
    --snapshot-type automated --query 'length(DBSnapshots)'
7
```

Las copias existen. El hallazgo no era ese:

```
tomar la copia                      proveedor    cliente    ✓ 7 instantáneas
cifrar la copia                     proveedor    cliente    ✓ StorageEncrypted
fijar retención                     cliente      cliente    ✓ 7 días
PROBAR LA RESTAURACIÓN              cliente      cliente    ✗ nunca se hizo
verificar integridad tras restaurar cliente      cliente    ✗
```

Dos zonas grises. El equipo heredó «copia de seguridad» completo cuando solo se heredaban dos de sus cinco actividades.

Se ejecuta la restauración por primera vez, en una cuenta de no producción:

```
$ aws rds restore-db-instance-from-db-snapshot \
    --db-instance-identifier cloudshop-restore-test \
    --db-snapshot-identifier rds:cloudshop-prod-2026-08-01-07-05
$ time aws rds wait db-instance-available --db-instance-identifier cloudshop-restore-test
real    38m12s
```

```
RTO comprometido en el plan de continuidad     60 min
RTO medido en la restauración                  38 min  ✓ cumple
RPO implícito (copia diaria a las 07:00)    hasta 24 h
RPO comprometido                               15 min  ✗ INCUMPLE
```

El segundo hallazgo aparece al medir, no al auditar. La retención de 7 días cumple el requisito de conservación, pero una copia diaria da un RPO de hasta 24 horas frente a los 15 minutos comprometidos. Nadie lo había notado porque la copia existía y se asumió que eso resolvía el requisito.

Se verifica la integridad, no solo la disponibilidad:

```
-- en el original
SELECT count(*), sum(total) FROM pedidos WHERE creado < '2026-08-01 07:00';
 1284471 | 48219338.55
-- en la restaurada
 1284471 | 48219338.55        ✓ coincide
```

Correcciones, separando qué es de cada quién:

```
RPO       activar recuperación a un instante (heredado, hay que ACTIVARLO)
          → RPO efectivo: 5 min                                  ✓
prueba    restauración trimestral automatizada en no producción
          con comparación de recuento y sumas                ✓ cliente
evidencia informe versionado con RTO y RPO medidos           ✓ cliente
control   regla que falla el despliegue si un recurso productivo
          no tiene recuperación a un instante activada       ✓ cliente
```

La última línea es la que convierte el arreglo en un control: sin ella, el próximo recurso que alguien cree volverá a nacer sin la función activada.

La lección: se heredaba el mecanismo y se creía heredar el resultado. El proveedor toma copias; que esas copias satisfagan tu RPO y sean restaurables es una afirmación tuya que exige evidencia tuya.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-01-cloud-principles-strategy-adoption/019-modelo-de-responsabilidad-compartida-por-servicio/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 raci-de-controles 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 raci-de-controles 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 auditoría rechaza la copia de seguridad pese a existir instantáneas	Nunca se probó la restauración: el proveedor toma la copia, el cliente demuestra que sirve	Automatiza una restauración periódica con verificación de integridad y RTO medido.
El RPO real es de 24 h aunque el plan comprometa 15 min	Se asumió que existir copia equivale a cumplir el RPO	Activa recuperación a un instante y mide el RPO efectivo, no la frecuencia de la copia.
Un recurso nuevo nace sin cifrado ni réplica	Las funciones gestionadas se ofrecen, no se imponen, y no había control que lo exigiera	Convierte la comprobación en política como código que falle el despliegue.
Se presenta el informe de cumplimiento del proveedor como prueba propia	El informe cubre la parte del proveedor y enumera lo que deja al cliente	Lee la sección de controles complementarios del usuario: es tu lista de tareas.
Se hereda un control de una región o servicio que el informe no cubre	No se revisó el alcance ni el periodo del informe	Comprueba servicios, regiones y ventana temporal antes de dar por heredado un control.

■ 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 diferencia hay entre control heredado, compartido y propio, y qué obligación te queda en el primero?
2. En una base de datos gestionada, ¿qué actividad de la copia de seguridad nunca ejecuta el proveedor?
3. ¿Por qué la columna «Responde» de la matriz RACI es siempre el cliente, y qué consecuencia práctica tiene?
4. ¿Cuál es la sección más útil de un informe de cumplimiento del proveedor y por qué?
5. Un servicio soporta cifrado en reposo. ¿Qué te falta comprobar antes de darlo por heredado?

Referencias

- AWS (2024). Shared Responsibility Model — controles heredados, compartidos y propios del cliente. <https://aws.amazon.com/compliance/shared-responsibility-model/>
- Microsoft (2024). Shared responsibility in the cloud — reparto por modelo de servicio con tabla de controles. <https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility>
- Cloud Security Alliance (2024). Cloud Controls Matrix — catálogo de controles con reparto proveedor/cliente. <https://cloudsecurityalliance.org/research/cloud-controls-matrix>
- AICPA (2017). SOC 2 Trust Services Criteria — estructura del informe y controles complementarios del usuario. <https://www.aicpa-cima.com/resources/landing/system-and-organization-controls-soc-suite-of-services>
- Dotson, C. (2023). Practical Cloud Security, 2.ª ed., cap. 1 — construir la matriz de responsabilidad servicio a servicio.
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 01 en PDF · Manual integral

Anterior	Índice	Siguiente
← 018 · Identidad, roles, políticas y federación	Parte 01 · Programa	020 · TCO, costos variables, unit economics y FinOps →

Evaluación — 019 Modelo de responsabilidad compartida por servicio

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 raci-de-controles 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.

020 — TCO, costos variables, unit economics y FinOps

Parte: 01 — Principios, estrategia y adopción cloud Nivel: inicial-intermedio · Horas estimadas: 4 Laboratorio: finops · Estado: EXECUTABLECORE

Propósito

Construir un caso de coste total defendible ante alguien que sabe de finanzas, no solo de infraestructura. La clase 011 dio la unidad de coste; aquí se añade lo que un TCO honesto incluye y casi ningún cálculo de migración recoge: personal, licencias, amortización pendiente, coste de salida y el valor temporal del dinero.

Resultados de aprendizaje

Al finalizar podrás:

1. Enumerar las partidas de un TCO que no aparecen en la factura y estimarlas con órdenes de magnitud.
2. Calcular el valor actual neto de una migración y explicar por qué comparar totales sin descontar engaña.
3. Distinguir coste hundido de coste evitable y por qué el hardware ya comprado no debe entrar en la decisión.
4. Construir un modelo de coste unitario que muestre si la escala mejora la economía o solo la traslada.
5. Presentar el caso con supuestos explícitos y análisis de sensibilidad sobre los dos que más pesan.

Conceptos centrales

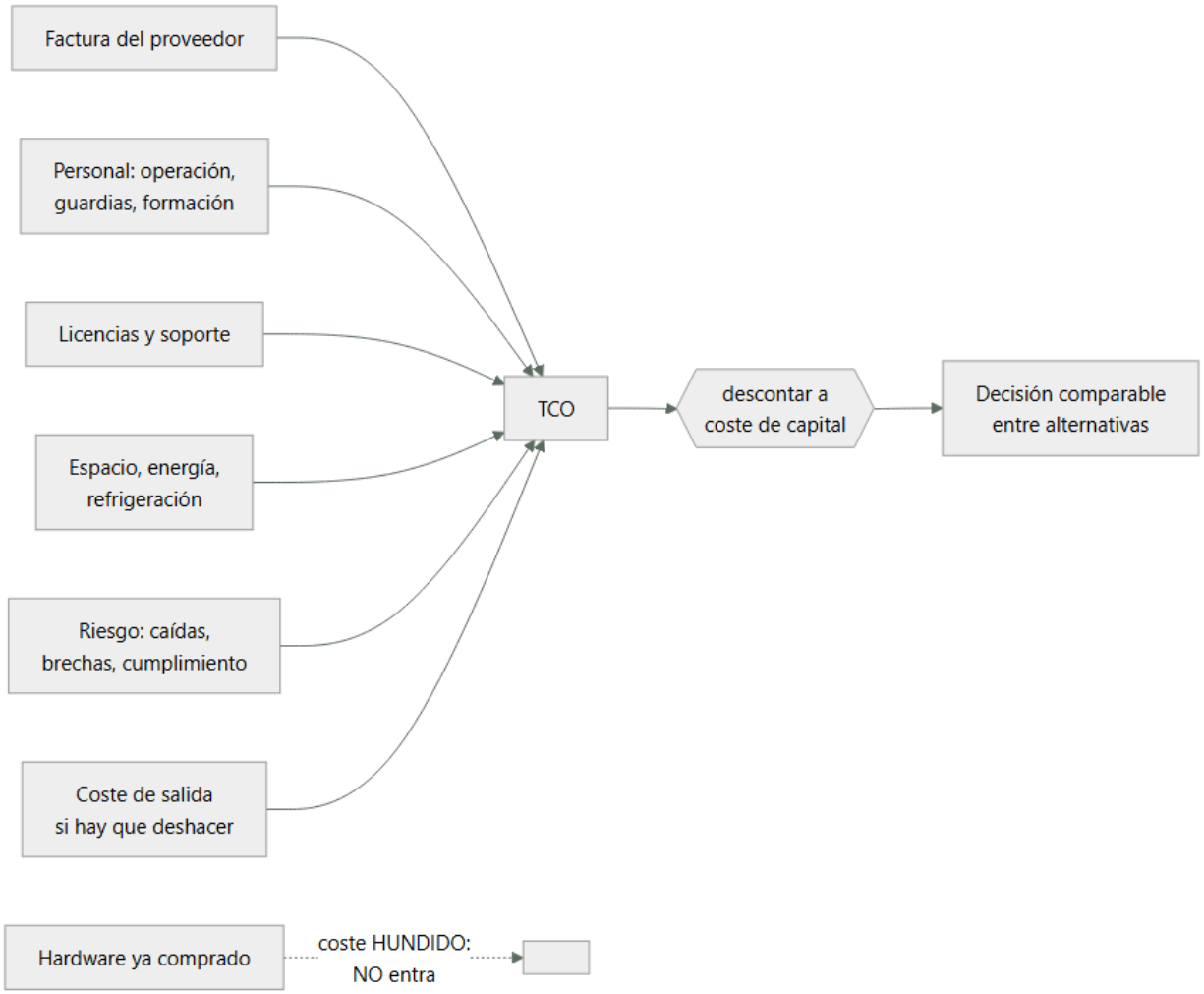
Concepto	Comprensión verificable
TCO	Coste total de propiedad: todo lo que cuesta operar una capacidad durante su vida útil, no solo lo que se factura. Incluye personal, licencias, espacio, energía, riesgo y coste de oportunidad.
coste hundido	Gasto ya realizado e irrecuperable. Es irrelevante para decidir el futuro: incluirlo en la comparación es la falacia del coste hundido, y produce decisiones de mantener sistemas solo porque ya se pagaron.
valor actual neto	Suma de flujos futuros descontados a una tasa que refleja el coste del capital. Permite comparar alternativas cuyos gastos ocurren en momentos distintos, que es siempre el caso en una migración.
unit economics	Coste e ingreso por unidad de negocio. Es lo que revela si crecer mejora o empeora el margen, algo que el total absoluto oculta por completo.
coste de salida	Lo que costaría deshacer la decisión: egreso de datos, reescritura, formación y tiempo. Rara vez se calcula y es lo que convierte una decisión reversible en una irreversible.

Modelo mental

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

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. Lo que la factura no incluye suele ser la mitad

Un TCO que solo suma la factura del proveedor subestima sistemáticamente ambos lados de la comparación. Las partidas ausentes, con órdenes de magnitud orientativos:

Partida	En centro de datos propio	En nube
Infraestructura	CAPEX amortizado a 3-5 años	Factura mensual
Personal de operación	1 persona por 100-200 servidores	Menor, pero no cero
Guardias	Rotación completa 24×7	Rotación completa 24×7
Licencias de virtualización	Significativas	Incluidas o distintas
Espacio, energía, refrigeración	Directo	Dentro del precio
Renovación de hardware	Ciclo de 4-5 años	No aplica
Sobreaprovisionamiento	Se compra para el pico	Se paga el consumo
Formación	Menor, tecnología estable	Alta y recurrente
Coste de salida	Bajo	Medio-alto

Dos filas merecen atención porque suelen decidir el resultado:

Personal. Es habitual que sea la mayor partida del centro de datos propio y la más ignorada. Una persona dedicada cuesta, con cargas, del orden de 40.000-90.000 USD anuales según el mercado; tres personas superan a mucha

infraestructura. La nube reduce esa partida pero no la elimina: cambia parcheo de sistemas operativos por gestión de identidades, políticas y costes.

Formación. Es la que se olvida en la dirección contraria. Un equipo que domina VMware necesita meses para operar con soltura en un hiperescalar, y el coste no es solo el curso: es la productividad reducida durante la transición y los errores del periodo de aprendizaje.

2. El hardware ya comprado no entra en la decisión

La objeción más frecuente contra una migración es «acabamos de invertir 400.000 USD en servidores». Es la falacia del coste hundido: ese dinero ya se gastó y no se recupera decidas lo que decidas.

Lo que sí entra es lo evitable:

```
NO entra (hundido):
  compra del hardware ya pagada           400.000 USD

SÍ entra (evitable si migras):
  soporte y mantenimiento restantes        38.000 USD/año
  energía y espacio                        22.000 USD/año
  renovación prevista en 2028             180.000 USD
  valor de reventa hoy                      60.000 USD (ingreso)
```

La pregunta correcta no es «¿amortizamos lo comprado?» sino «a partir de hoy, ¿qué alternativa cuesta menos?». Si operar lo existente cuesta 60.000 USD anuales evitables y la nube cuesta 45.000, migrar ahorra 15.000 al año por mucho que se hayan pagado 400.000 el año pasado.

Hay dos matices legítimos que no son la falacia:

1. El valor de reventa sí entra, como ingreso, y desaparece con el tiempo: un servidor de tres años vale bastante menos que uno de uno.
2. Un contrato con penalización por cancelación anticipada es coste evitable de la opción de migrar, no coste hundido. Hay que incluirlo.

La distinción es la que permite tener una conversación financiera en vez de una emocional.

3. Descontar: 100.000 hoy no son 100.000 dentro de tres años

Una migración concentra gastos al principio y ahorros después. Comparar los totales sin descontar favorece artificialmente a la alternativa que gasta tarde.

```
VAN = Σ  flujo_t / (1 + r)^t
```

Con una tasa de descuento del 10 % —el coste del capital de la organización— y un horizonte de 5 años:

```
año   flujo neto      factor      valor actual
 0    −250.000        1,000       −250.000     migración e implantación
 1     +85.000        0,909        +77.273
 2     +92.000        0,826        +76.033
 3     +99.000        0,751        +74.380
 4    +106.000        0,683        +72.395
 5    +113.000        0,621        +70.156
                                   ----------
                            VAN = +120.237 USD
```

Sin descontar, la suma sería +245.000. Con descuento, +120.237: el descuento se come la mitad del beneficio aparente, porque los ahorros llegan tarde y el gasto es inmediato.

Dos métricas complementarias que la dirección suele pedir:

```
periodo de recuperación simple: 250.000 / 85.000 ≈ 2,9 años
tasa interna de retorno (TIR):  ≈ 24 %
```

Si la TIR supera el coste de capital, el proyecto crea valor. Con 24 % frente a 10 %, lo crea con holgura — siempre que los ahorros proyectados se materialicen, que es donde entra el análisis de sensibilidad.

4. Unit economics: la escala puede empeorar el margen

El total absoluto oculta si el negocio mejora al crecer. La descomposición mínima:

```
margen unitario = ingreso por unidad − coste por unidad
```

Con los datos de una plataforma real:

```
pedidos                  620.000      980.000    1.740.000
coste de infraestructura  12.400       17.360       26.100
coste unitario             0,0200       0,0177       0,0150
ingreso por pedido         0,4200       0,4100       0,3900
margen unitario            0,4000       0,3923       0,3750
```

El coste unitario baja un 25 %: hay economía de escala. Pero el margen unitario también baja, porque el ingreso por pedido cae más rápido que el coste. La infraestructura mejora y el negocio empeora, y solo se ve con las dos series juntas.

El patrón contrario también existe y es más peligroso: un coste unitario que sube con el volumen indica que algo escala peor que linealmente —normalmente el término de coherencia de la clase 016, o transferencia entre zonas que crece con el cuadrado de los nodos—.

Tres preguntas que el modelo debe responder:

1. ¿El coste unitario baja, sube o se mantiene al crecer?
2. ¿Qué partida domina, y cambia esa dominancia con la escala?
3. ¿A qué volumen el coste unitario deja de bajar? Es el punto donde la arquitectura actual agota su economía.

5. Supuestos explícitos y sensibilidad sobre los dos que más pesan

Un TCO sin supuestos declarados no es defendible: cualquiera puede rehacerlo con otros números y llegar a la conclusión contraria. La disciplina mínima es listar los supuestos y probar cuánto aguanta la conclusión.

```
supuestos del modelo
  crecimiento de volumen         +18 % anual
  reducción de personal           1,5 personas
  utilización media               58 %
  tasa de descuento               10 %
  horizonte                       5 años
  inflación de precios cloud       0 % (los proveedores tienden a bajar)
```

El análisis de sensibilidad se hace sobre los dos supuestos con más peso, no sobre todos:

```
caso base                        +120.237
crecimiento +9 % en vez de +18 %  +71.400
sin reducción de personal          −18.900   ← cambia el signo
utilización 40 % en vez de 58 %   +148.000   (más ahorro por elasticidad)
descuento 15 %                     +82.100
```

El caso depende críticamente de un supuesto: la reducción de 1,5 personas. Si no ocurre, el proyecto destruye valor. Eso convierte una decisión técnica en una decisión organizativa, y es exactamente lo que la dirección necesita saber antes de aprobar.

Declararlo tiene una ventaja adicional: convierte el supuesto en un compromiso verificable. Si a los doce meses el personal no se ha reducido, no hay que esperar cinco años para saber que el caso no se cumple.

Ejemplo trabajado

CloudShop evalúa migrar su plataforma del centro de datos actual. El equipo de infraestructura presenta un ahorro del 60 % comparando la factura estimada con el gasto en hardware. Se rehace el cálculo completo.

Cálculo original presentado:

```
amortización de hardware      95.000 USD/año
factura cloud estimada        38.000 USD/año
"ahorro"                      57.000 USD/año (60 %)
```

Errores detectados: incluye coste hundido, omite personal en ambos lados, ignora licencias, no descuenta y no contempla el coste de salida.

TCO reconstruido, solo con costes evitables desde hoy:

```
soporte y mantenimiento de hardware       38.000          0
energía, espacio, refrigeración           22.000          0
licencias de virtualización               31.000          0
personal de operación (3,0 → 1,5 FTE)    186.000     93.000
```

```
factura del proveedor                          0     38.000
sobreaprovisionamiento (util. 30 %)   ya incluido    evitado
formación primer año                           0     24.000
                                          --------   --------
coste anual evitable                     277.000    155.000
ahorro anual                                        122.000
```

Más del doble del ahorro que se presentaba —y por razones distintas: no viene del hardware sino del personal y las licencias.

Costes de una sola vez:

```
migración e implantación (6 meses, 2 personas)   140.000
doble ejecución durante la transición             46.000
reescritura de dos componentes acoplados          64.000
valor de reventa del hardware                    −60.000
                                                 -------
inversión neta inicial                           190.000
```

Valor actual neto a 5 años, descontando al 10 %:

```
año 0   −190.000   × 1,000 = −190.000
año 1   +122.000   × 0,909 = +110.909
año 2   +122.000   × 0,826 = +100.826
año 3   +122.000   × 0,751 =  +91.660
año 4   +122.000   × 0,683 =  +83.327
año 5   +122.000   × 0,621 =  +75.752
                            --------
                     VAN = +272.474 USD     TIR ≈ 56 %
                     recuperación ≈ 1,6 años
```

Sensibilidad sobre los dos supuestos dominantes:

```
caso base                                      +272.474
personal NO baja de 3,0 a 1,5 FTE               −80.100   ← el caso se cae
migración cuesta el doble (280.000)            +182.474
factura cloud un 40 % mayor (53.200)           +214.800
ambos adversos: sin reducción y +40 % factura −137.700
```

El proyecto depende de una sola variable: la reducción de personal. No de la factura, no del coste de migración. Esa es la conclusión que debe llevarse la dirección, y no aparecía en ninguna parte del cálculo original.

Se declara además el coste de salida, que nadie había estimado:

```
egreso de 42 TB a 0,09 USD/GB                   3.780 USD
reescritura de adaptadores propietarios        ~35.000 USD
reformación del equipo                         ~20.000 USD
coste de deshacer la decisión                  ~59.000 USD
```

Recomendación registrada: migrar, condicionado a un plan explícito de recolocación de 1,5 personas, con revisión a los 12 meses. Si a esa fecha el personal no se ha reducido, el caso se reevalúa con el coste de salida ya calculado sobre la mesa.

La diferencia entre el primer cálculo y este no es la cifra: es que el segundo dice de qué depende.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-01-cloud-principles-strategy-adoption/020-tco-costos-variables-unit-economics-y-finops/lab.py
```

El laboratorio selecciona el motor de práctica finops 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 caso-tco 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 cálculo trazable con unidad, supuesto y sensibilidad. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye caso-tco 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 rechaza una migración porque «acabamos de comprar el hardware»	Falacia del coste hundido: el gasto ya realizado no se recupera con ninguna decisión	Compara solo costes evitables desde hoy; incluye la reventa como ingreso, no la compra como coste.
El ahorro proyectado nunca aparece en las cuentas	El caso dependía de una reducción de personal que no se ejecutó	Declara los supuestos organizativos como compromisos verificables con fecha de revisión.
Dos alternativas parecen equivalentes y una es claramente peor	Se compararon totales sin descontar, favoreciendo a la que gasta más tarde	Calcula el VAN con la tasa de coste de capital de la organización.
El volumen crece, la factura crece y nadie sabe si eso es bueno	No hay unit economics: falta el coste por unidad junto al ingreso por unidad	Sigue las dos series juntas; el coste unitario puede bajar mientras el margen empeora.
Deshacer una decisión resulta imposible por su coste	El coste de salida nunca se estimó, así que la decisión era irreversible sin saberlo	Calcula egreso, reescritura y reformación al decidir, no cuando quieras salir.

■ 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. Se compraron servidores por 400.000 USD el año pasado. ¿Qué parte de esa cifra entra en la decisión de migrar y cuál no?
2. ¿Por qué comparar el total sin descontar favorece a la alternativa que concentra el gasto al final?
3. El coste unitario baja un 25 % y el margen unitario también baja. ¿Qué está ocurriendo y por qué el total lo oculta?
4. ¿Sobre qué supuestos conviene hacer el análisis de sensibilidad y cómo se eligen?
5. Nombra tres partidas de un TCO que no aparecen en ninguna factura y afectan a ambos lados de la comparación.

Referencias

- Storment, J. R. y Fuller, M. (2023). Cloud FinOps, 2.ª ed., caps. 4-6 — modelo de coste unitario y previsión.
- FinOps Foundation (2024). Unit Economics — definición y construcción de métricas por unidad de negocio. <https://www.finops.org/framework/capabilities/unit-economics/>
- Brealey, R., Myers, S. y Allen, F. (2020). Principles of Corporate Finance, 13.ª ed., caps. 2-6 — VAN, TIR y costes hundidos.
- Hohpe, G. (2020). Cloud Strategy, cap. sobre economía — por qué el ahorro de la nube rara vez viene de donde se anuncia.
- Barroso, L., Hölzle, U. y Ranganathan, P. (2018). The Datacenter as a Computer, 3.ª ed., cap. 6 — TCO de un centro de datos. <https://doi.org/10.2200/S00874ED3V01Y201809CAC046>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 01 en PDF · Manual integral

Anterior	Índice	Siguiente
← 019 · Modelo de responsabilidad compartida por servicio	Parte 01 · Programa	021 · Well-Architected y atributos de calidad →

Evaluación — 020 TCO, costos variables, unit economics y FinOps

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 caso-tco 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.

021 — Well-Architected y atributos de calidad

Parte: 01 — Principios, estrategia y adopción cloud Nivel: inicial-intermedio · Horas estimadas: 4 Laboratorio: architecture · Estado: EXECUTABLECORE

Propósito

Usar los marcos Well-Architected como una lista de preguntas que fuerza a hacer explícitos los compromisos, no como una certificación que se aprueba. Su valor real está en los pilares que se contradicen entre sí: mejorar seguridad suele costar rendimiento, y mejorar coste suele costar fiabilidad. Esta clase enseña a nombrar ese intercambio en vez de fingir que no existe.

Resultados de aprendizaje

Al finalizar podrás:

1. Traducir un requisito de negocio a atributos de calidad medibles con escenario, estímulo y respuesta.
2. Identificar qué pares de pilares se oponen en una decisión concreta y cuantificar el intercambio.
3. Conducir una revisión que produzca riesgos priorizados y no una lista de buenas prácticas genéricas.
4. Distinguir un riesgo alto de uno medio por su impacto y su reversibilidad, no por su gravedad aparente.
5. Convertir los hallazgos en acciones con responsable, fecha y criterio de cierre verificable.

Conceptos centrales

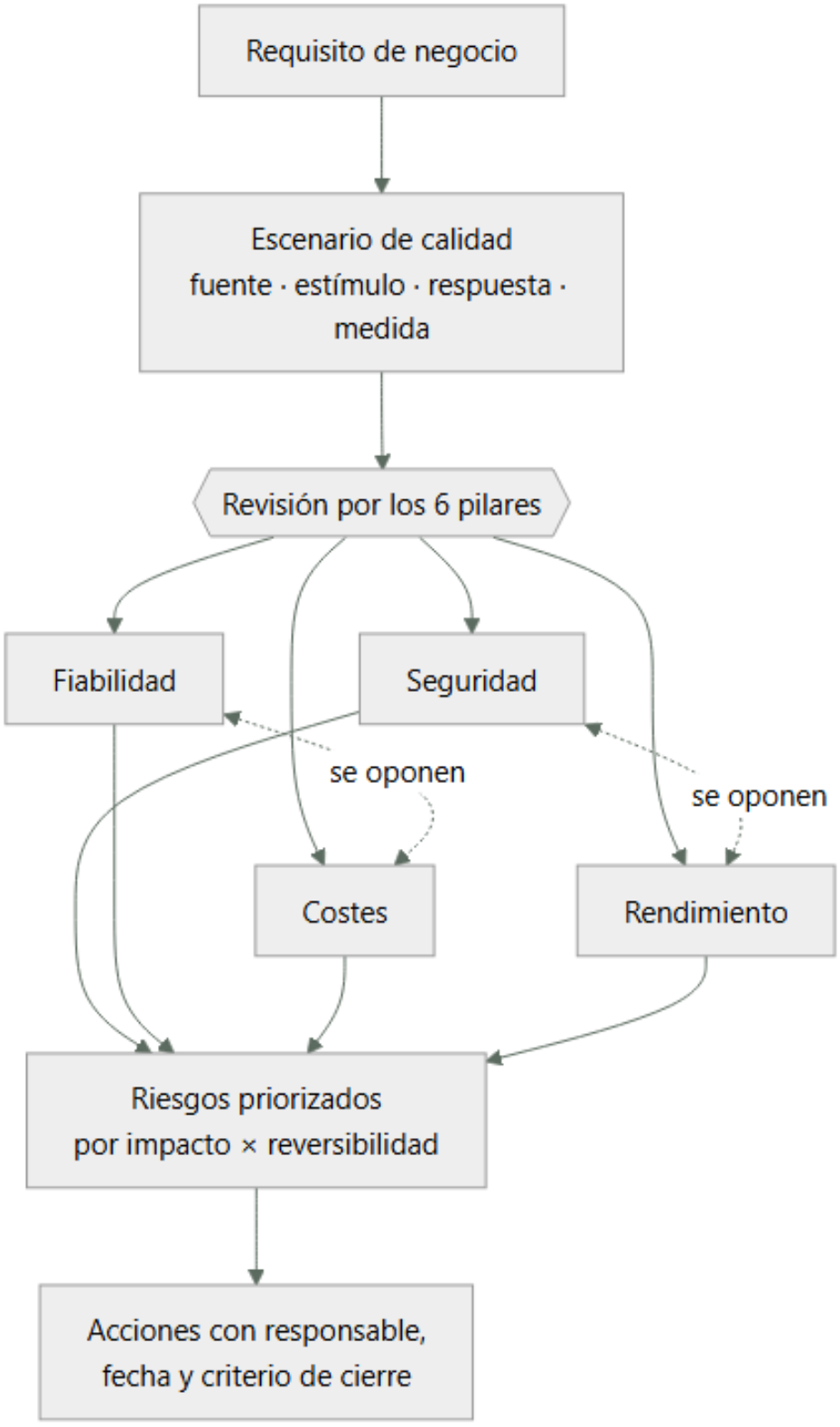
Concepto	Comprensión verificable
atributo de calidad	Propiedad no funcional medible: latencia, disponibilidad, coste unitario, tiempo de recuperación. Se especifica con un escenario completo, porque «rápido» y «seguro» no son requisitos verificables.
pilar	Dimensión de análisis del marco: excelencia operativa, seguridad, fiabilidad, eficiencia de rendimiento, optimización de costes y sostenibilidad. Su utilidad es forzar a mirar las seis, incluidas las incómodas.
escenario de calidad	Formulación de seis partes —fuente, estímulo, artefacto, entorno, respuesta y medida— que convierte una aspiración en algo comprobable.
compromiso	Reconocimiento explícito de que mejorar un atributo empeora otro. Un diseño sin compromisos declarados no es equilibrado: es un diseño cuyos costes aún no se han descubierto.
riesgo alto	Hallazgo cuyo impacto es grave y cuya corrección posterior es cara o destructiva. La reversibilidad, no solo la gravedad, es lo que separa el riesgo alto del medio.

Modelo mental

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

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 atributo sin escenario no es un requisito

«El sistema debe ser rápido y seguro» no se puede verificar, así que no se puede diseñar contra ello. La formulación de seis partes de Bass, Clements y Kazman lo convierte en algo comprobable:

Parte	Ejemplo
Fuente	Un usuario autenticado desde Chile
Estímulo	Solicita la página de un producto
Artefacto	El servicio de catálogo
Entorno	Operación normal, hora punta
Respuesta	Devuelve la página completa
Medida	p95 < 300 ms, medido en el cliente

La última fila es la que hace la diferencia. Y tiene dos detalles que se omiten sistemáticamente:

1. El percentil, porque como se vio en la clase 012 la media no la experimenta nadie.
2. El punto de medición. «300 ms en el servidor» y «300 ms en el cliente» difieren en todo el tramo de red, que en la clase 001 resultó ser el componente dominante. Un SLO medido en el sitio equivocado se cumple mientras los usuarios se quejan.

El mismo rigor aplica a los demás pilares:

```
disponibilidad  99,95 % mensual medido sobre peticiones con éxito, excluyendo
                mantenimiento anunciado con 72 h de antelación
coste           ≤ 0,018 USD por pedido a volumen de 1 M/mes
recuperación    RTO 60 min y RPO 15 min, verificados trimestralmente
seguridad       ninguna credencial de larga vida en el camino de despliegue
```

Cada uno tiene número, unidad y método de verificación. Sin los tres, un pilar es una intención.

2. Los pilares se contradicen, y ahí está el trabajo

El error más común al usar el marco es tratarlo como una lista donde todo se puede maximizar. Los pares que se oponen, con el mecanismo concreto:

Par	Por qué chocan	Ejemplo medible
Seguridad ↔ Rendimiento	Cifrado, inspección y verificación añaden latencia	mTLS entre servicios: +3-8 ms por salto
Fiabilidad ↔ Coste	La redundancia se paga aunque no se use	3 zonas: +50 % de cómputo sobre 2
Rendimiento ↔ Coste	La capacidad libre absorbe varianza	ρ 0,70 en vez de 0,95: +36 % de instancias
Operación ↔ Rapidez	Las puertas de calidad frenan la entrega	Revisión obligatoria: +4 h de plazo medio
Sostenibilidad ↔ Rendimiento	La región limpia puede estar lejos	+40 ms de latencia por 8× menos CO■e

Una revisión útil cuantifica estos intercambios en vez de declararlos. «Hay un trade-off entre seguridad y rendimiento» no ayuda a decidir; «mTLS cuesta 6 ms sobre un presupuesto de 300 y elimina la posibilidad de suplantación este-oeste» sí.

Y hay un pilar que se opone a casi todos: la simplicidad, que no figura en el marco pero debería. Cada mecanismo añadido —una caché, una cola, una región extra— mejora un atributo y empeora la capacidad del equipo de operar y diagnosticar el sistema. Ese coste no aparece en ninguna métrica hasta el primer incidente a las 3 de la madrugada.

3. Priorizar por impacto y reversibilidad

Los marcos clasifican hallazgos en alto, medio y bajo, y la clasificación suele hacerse solo por gravedad. Falta la segunda dimensión: cuánto cuesta arreglarlo después.

```
riesgo = impacto × probabilidad × (coste de corregir tarde / coste de corregir ahora)
```

El tercer factor es el que reordena la lista:

Hallazgo	Impacto	Corregir ahora	Corregir en 2 años	Prioridad
Sin cifrado en tránsito	Alto	2 días	1 semana	Media
Esquema de datos sin particionar	Medio	1 semana	6 meses + migración	Alta

Hallazgo	Impacto	Corregir ahora	Corregir en 2 años	Prioridad
Sin alertas de coste	Medio	1 día	1 día	Baja
Identificadores secuenciales expuestos	Alto	3 días	Reescritura de clientes	Alta

La segunda fila es el patrón clave: un hallazgo de impacto medio con corrección casi irreversible pesa más que uno de impacto alto y arreglo trivial. Las decisiones de datos y de contratos públicos son las que más envejecen: cambiar un esquema con 2 TB en producción o un formato de identificador que ya consumen terceros no es lo mismo que activar una opción.

La regla operativa: decide pronto lo irreversible y tarde lo reversible. Un hallazgo reversible puede esperar a tener más información; uno irreversible hay que resolverlo mientras el coste sigue siendo bajo.

4. Una revisión que produce decisiones, no una lista

Una revisión Well-Architected mal conducida genera 60 recomendaciones genéricas que nadie ejecuta. Bien conducida produce entre 5 y 10 riesgos con dueño.

El guion que funciona:

```
1. Escenarios primero (30 min)
   ¿Cuáles son los 5 atributos de calidad con número y método de medición?
   Sin esto, la revisión no tiene contra qué contrastar.

2. Recorrido por pilares (2-3 h)
   Para cada pregunta: ¿lo hacemos? ¿cómo lo demostramos? ¿qué evidencia hay?
   "Sí" sin evidencia se registra como "no verificado", que es distinto de "no".

3. Compromisos explícitos (45 min)
   ¿Qué decisión mejora un pilar a costa de otro? ¿Está declarada y aceptada?

4. Priorización (30 min)
   Impacto × reversibilidad. Máximo 10 riesgos; si salen 40, la revisión
   está describiendo el estado, no priorizando.

5. Acciones (30 min)
   Cada riesgo alto: responsable, fecha, criterio de cierre verificable.
```

El paso 2 tiene una regla que cambia el resultado: distinguir «no» de «no verificado». Un equipo suele responder «sí, tenemos copias de seguridad»; la pregunta siguiente —«¿cuándo se restauró por última vez?»— convierte muchos síes en no verificados, que es exactamente el hallazgo de la clase 019.

Y el paso 5 exige un criterio de cierre comprobable: «mejorar la seguridad de IAM» no se puede cerrar; «ninguna clave de acceso permanente activa, verificado por una consulta automática semanal» sí.

5. Los marcos coinciden más de lo que su vocabulario sugiere

Los tres grandes proveedores tienen su versión y las tres comparten estructura. Traducirlas evita aprender tres veces lo mismo y permite hacer revisiones multi-proveedor:

Pilar	AWS	Azure	Google Cloud
Operación	Operational Excellence	Operational Excellence	Operational excellence
Seguridad	Security	Security	Security, privacy, compliance
Fiabilidad	Reliability	Reliability	Reliability
Rendimiento	Performance Efficiency	Performance Efficiency	Performance optimization
Coste	Cost Optimization	Cost Optimization	Cost optimization
Sostenibilidad	Sustainability	—	Sustainability

Las diferencias reales son de énfasis, no de fondo: AWS incorporó sostenibilidad como pilar en 2021; Azure la trata dentro de otros; Google agrupa privacidad y cumplimiento con seguridad.

El límite honesto de los tres: están escritos por quien vende la plataforma. Ninguno pregunta si deberías estar en esa nube, si el bloqueo es aceptable o si la alternativa más barata es no migrar. Son excelentes para revisar cómo está construido algo y no sirven para decidir si construirlo.

Para esa segunda pregunta hacen falta los criterios de las clases 013, 015 y 020: definición, modelo de servicio y coste total. Un marco Well-Architected aplicado a una decisión que no debió tomarse produce un sistema impecablemente construido sobre una premisa equivocada.

Ejemplo trabajado

Revisión Well-Architected de la plataforma de pedidos de CloudShop antes de duplicar el volumen. Se sigue el guion y se cuantifican los compromisos.

Paso 1 — escenarios de calidad, con medida y punto de medición:

```
Q1 latencia      p95 < 300 ms medido en el cliente, hora punta
Q2 disponibilidad 99,95 % mensual sobre peticiones con éxito
Q3 coste          ≤ 0,018 USD/pedido a 1 M pedidos/mes
Q4 recuperación   RTO 60 min · RPO 15 min, verificado cada trimestre
Q5 seguridad      cero credenciales de larga vida en el despliegue
```

Paso 2 — recorrido, separando «no» de «no verificado»:

```
copias de seguridad automáticas          sí          ✓ 7 instantáneas
restauración probada                     "sí"        ✗ NO VERIFICADO
despliegue sin secretos                  sí          ✓ OIDC (clase 018)
límites de tasa por cliente              no          —
particionado del esquema de pedidos      no          —
alertas sobre consumo de cuota           no          —
```

Paso 3 — compromisos, cuantificados:

```
mTLS entre servicios
  seguridad: elimina suplantación este-oeste
  rendimiento: +6 ms medidos sobre 3 saltos → 18 ms del presupuesto de 300 (6 %)
  DECISIÓN: se adopta. 6 % del presupuesto por eliminar una clase entera de ataque.

tercera zona de disponibilidad
  fiabilidad: de 99,995 % a 99,999 % → de 1,3 a 0,3 min/mes
  coste: +50 % de cómputo = +58 USD/mes
  DECISIÓN: NO. El requisito es 21,6 min/mes; ya se cumple con holgura.

utilización objetivo 0,70 en vez de 0,90
  rendimiento: p95 de 91 ms en vez de ~300 ms (clase 011)
  coste: +36 % de instancias
  DECISIÓN: se adopta. Sin ella Q1 no se cumple.
```

Paso 4 — priorización por impacto × reversibilidad:

```
esquema de pedidos sin particionar        medio   1 sem   6 meses+migr   ALTA
restauración nunca probada                alto    2 días  2 días         ALTA
sin límites de tasa por cliente           alto    3 días  3 días         MEDIA
sin alerta de cuota                       medio   1 día   1 día          BAJA
```

El esquema sin particionar sube a alta pese a impacto medio: hoy la tabla tiene 180 GB y particionar cuesta una semana; con el volumen duplicado durante dos años serán 1,3 TB y la migración exigirá ventana de indisponibilidad. Es el único hallazgo cuya ventana de corrección barata se está cerrando.

La restauración no probada es alta por impacto puro: su coste de corrección no crece, pero el riesgo se materializa en cualquier momento.

Paso 5 — acciones con criterio de cierre:

```
R1  particionar pedidos por mes         Ana   15-09   pg_partman activo y
                                                      consulta p95 < 40 ms
R2  restauración automatizada           Luis  22-08   informe trimestral con
                                                      RTO medido y recuento verificado
R3  límite de tasa por cliente          Ana   30-09   429 con Retry-After bajo
                                                      prueba de carga
R4  alerta de cuota al 80 %             Luis  08-08   alarma disparada en simulacro
```

Resultado: 4 acciones con dueño y criterio comprobable, frente a las 34 recomendaciones genéricas que devolvió la herramienta automática. La diferencia la produjo el paso 1: sin los cinco escenarios con número, no había contra qué

contrastar y todo parecía igual de importante.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-01-cloud-principles-strategy-adoption/021-well-architected-y-atributos-de-calidad/lab.py
```

El laboratorio selecciona el motor de práctica architecture 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 revision-well-architected 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 diagrama acompañado por decisiones y trade-offs. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye revision-well-architected 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 revisión produce 40 recomendaciones y no se ejecuta ninguna	Se describió el estado en vez de priorizar; faltaban escenarios contra los que contrastar	Empieza por 5 atributos con número y método de medición; limita la salida a 10 riesgos.
Un SLO de latencia se cumple mientras los usuarios se quejan	Se mide en el servidor y el tramo dominante es la red	Declara el punto de medición en el escenario; medir en el cliente cambia el número por completo.
Un hallazgo de impacto medio se pospone y dos años después cuesta una migración	Se priorizó solo por gravedad, ignorando la reversibilidad	Prioriza por impacto × coste de corregir tarde; decide pronto lo irreversible.
El equipo declara tener un control que en realidad nunca se ejerció	No se distinguió «sí» de «sí, pero no verificado»	Pide la evidencia en la misma pregunta; un sí sin evidencia se registra como no verificado.
Se construye impecablemente un sistema que no debió existir	El marco revisa cómo está hecho algo, no si debía hacerse	Decide primero con definición, modelo de servicio y TCO; el marco viene después.

■ 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. Convierte «el sistema debe ser rápido» en un escenario de calidad completo, incluyendo el punto de medición.
2. Nombra dos pares de pilares que se opongan y cuantifica el intercambio con un número concreto.
3. ¿Por qué un hallazgo de impacto medio puede tener más prioridad que uno de impacto alto?
4. ¿Qué diferencia hay entre responder «no» y «no verificado» en una revisión, y cómo se distingue?
5. ¿Qué pregunta importante no responde ningún marco Well-Architected, y con qué criterios se responde?

Referencias

- AWS (2024). Well-Architected Framework — seis pilares, preguntas y proceso de revisión. <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html>
- Microsoft (2024). Azure Well-Architected Framework — pilares y compromisos entre ellos. <https://learn.microsoft.com/en-us/azure/well-architected/>
- Google Cloud (2024). Architecture Framework — pilares y guía de diseño. <https://cloud.google.com/architecture/framework>
- Bass, L., Clements, P. y Kazman, R. (2021). Software Architecture in Practice, 4.ª ed., caps. 3-4 — escenarios de atributos de calidad en seis partes.
- Richards, M. y Ford, N. (2020). Fundamentals of Software Architecture, cap. 2 — «todo en arquitectura es un compromiso».
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 01 en PDF · Manual integral

Anterior	Índice	Siguiente
← 020 · TCO, costos variables, unit economics y FinOps	Parte 01 · Programa	022 · Cloud Adoption Framework y modelo operativo →

Evaluación — 021 Well-Architected y atributos de calidad

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 revision-well-architected 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.

022 — Cloud Adoption Framework y modelo operativo

Parte: 01 — Principios, estrategia y adopción cloud Nivel: inicial-intermedio · Horas estimadas: 4 Laboratorio: governance · Estado: EXECUTABLECORE

Propósito

Entender por qué las migraciones fracasan por causas organizativas y no técnicas, y qué modelo operativo hay que construir para que la plataforma no se convierta en un cuello de botella. La ley de Conway no es una curiosidad: es la razón por la que una arquitectura de microservicios impuesta a una organización de silos produce un monolito distribuido.

Resultados de aprendizaje

Al finalizar podrás:

1. Aplicar la ley de Conway en sentido inverso: diseñar equipos para obtener la arquitectura que se quiere.
2. Elegir entre los modelos operativos centralizado, descentralizado y de plataforma según tamaño y madurez.
3. Distinguir un equipo de plataforma que ofrece un producto de uno que se convierte en un servicio de tickets.
4. Secuenciar una adopción por olas con criterios de salida verificables en vez de por fechas.
5. Reconocer las señales tempranas de que el modelo operativo elegido está fallando.

Conceptos centrales

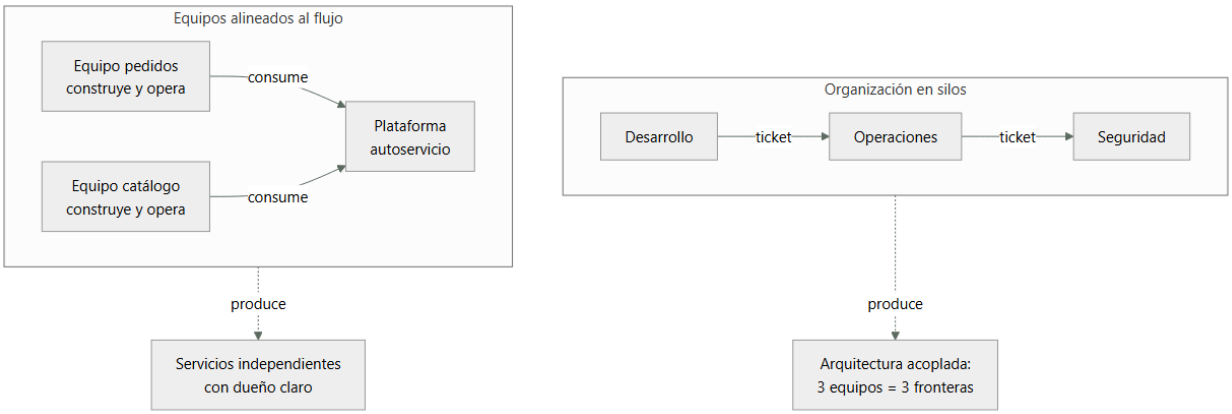
Concepto	Comprensión verificable
ley de Conway	Las organizaciones producen diseños que replican sus estructuras de comunicación. No es una tendencia evitable con disciplina: es una restricción que conviene usar deliberadamente.
carga cognitiva	Cantidad de conocimiento que un equipo debe sostener para operar lo suyo. Cuando supera su capacidad, la calidad cae sin que nadie sea negligente; es el límite real al tamaño del dominio de un equipo.
camino dorado	Trayecto recomendado y bien soportado para hacer algo común. No es obligatorio: compite por ser tan bueno que salirse resulte caro, y esa voluntariedad es lo que lo mantiene honesto.
equipo de plataforma	Equipo cuyo producto son las capacidades internas que otros consumen en autoservicio. Su métrica no es cuántos tickets cierra, sino cuántos deja de recibir.
ola de adopción	Grupo de cargas que se migran juntas con un criterio de salida definido. Avanzar por olas con criterio permite aprender; avanzar por fechas produce deuda que se descubre en la última ola.
equipo habilitador	Equipo que trabaja temporalmente con otros para elevar su capacidad y después se retira. Se distingue del de plataforma en que no deja un servicio permanente, sino conocimiento.

Modelo mental

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

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 ley de Conway, usada al revés

Conway observó en 1968 que las organizaciones producen diseños que copian sus estructuras de comunicación. La formulación es descriptiva, pero su uso práctico es prescriptivo: si quieres una arquitectura concreta, organiza los equipos con esa forma.

El caso negativo se repite en todas las migraciones grandes:

```
organización:  desarrollo → operaciones → seguridad   (tres silos, tickets entre ellos)
se pide:       microservicios independientes
se obtiene:    servicios que no se pueden desplegar sin coordinar tres equipos
               = un monolito distribuido, con lo peor de ambos modelos
```

El resultado no es un fallo de ejecución: es exactamente lo que la estructura de comunicación permite construir. Ningún equipo puede desplegar sin los otros dos, así que la frontera real del sistema son los tres equipos, no los servicios.

La maniobra de Conway inversa consiste en cambiar primero la organización:

```
antes:  3 equipos por función  →  fronteras por capa técnica
después: N equipos por dominio →  fronteras por dominio de negocio
```

Un equipo que construye y opera pedidos —incluyendo su base de datos, su despliegue y su guardia— puede producir un servicio independiente porque no necesita hablar con nadie para cambiarlo. Esa autonomía es la condición de la arquitectura, no su consecuencia.

Y el límite es honesto: reorganizar equipos es caro y lento. Por eso conviene hacerlo antes de comprometerse con una arquitectura, no después de descubrir que no encaja.

2. Tres modelos operativos y cuándo falla cada uno

Modelo	Cómo funciona	Escala hasta	Falla cuando
Centralizado	Un equipo aprovisiona todo por petición	5 equipos consumidores	Se convierte en cola: el tiempo de espera domina la entrega
Descentralizado	Cada equipo hace lo suyo sin capa común	10 equipos	Divergencia: 10 formas de desplegar, 10 posturas de seguridad
Plataforma	Capacidades en autoservicio con guardarraíles	Decenas	El equipo de plataforma se convierte en soporte de tickets

Los síntomas de que un modelo agotó su rango son medibles:

```
centralizado agotado:  tiempo de espera para un entorno nuevo > 3 días
                       cola de peticiones creciente semana a semana

descentralizado agotado: n.º de formas distintas de desplegar ≈ n.º de equipos
                         un hallazgo de seguridad exige N correcciones distintas

plataforma degradada:   > 40 % del tiempo del equipo en peticiones puntuales
                        el autoservicio existe y nadie lo usa
```

La tercera fila es la más frecuente y la más silenciosa: un equipo de plataforma que dedica la mayoría de su tiempo a resolver casos particulares ha vuelto al modelo centralizado con otro nombre. La causa suele ser que el camino dorado no cubre los casos reales, así que todos se salen de él y piden ayuda.

No hay un modelo correcto: hay un modelo adecuado al tamaño. Imponer plataforma a una organización de tres equipos crea más trabajo del que ahorra.

3. Carga cognitiva: el límite que no se negocia

Skelton y Pais sitúan la carga cognitiva como la restricción central del diseño de equipos. Un equipo tiene una capacidad finita de conocimiento sostenido, y cuando el dominio la supera la calidad cae sin que nadie sea negligente.

Los tres tipos, y qué hacer con cada uno:

Tipo	Qué es	Acción
Intrínseca	Complejidad del dominio: reglas de negocio	Formación; no se puede eliminar
Extrínseca	Cómo desplegar, configurar redes, obtener permisos	Eliminar con plataforma
Pertinente	Diseñar bien lo propio	Proteger: es donde está el valor

El objetivo de una plataforma es reducir la extrínseca para liberar capacidad para la pertinente. Medirlo es posible:

```
antes de la plataforma
  crear un entorno nuevo        14 pasos, 3 equipos, 4 días
  conocimiento necesario        redes, IAM, DNS, certificados, CI

después
  crear un entorno nuevo        1 comando, 12 minutos
  conocimiento necesario        el nombre del entorno
```

Si un equipo de producto necesita entender redes virtuales, políticas de identidad y emisión de certificados para publicar un servicio, esa carga extrínseca está consumiendo la capacidad que debería ir al dominio. No es que sean malos ingenieros: es que el sistema les pide sostener demasiado.

Y el criterio para dimensionar un dominio: si el equipo no puede explicar todo lo que opera en una pizarra durante 15 minutos, el dominio es demasiado grande.

4. Camino dorado: voluntario y bueno, o no funciona

Un camino dorado es el trayecto recomendado para hacer algo común. Su propiedad definitoria es que es voluntario, y esa voluntariedad es lo que lo mantiene honesto: si nadie lo usa, es que no es bueno, y esa señal es valiosa.

Qué lo hace funcionar:

```
1. Resuelve el 80 % de los casos, no el 100 %.
   Perseguir el 100 % produce una plataforma que tarda años y no llega.

2. Trae seguridad y cumplimiento incorporados.
   Usarlo debe ser más fácil QUE saltárselo, y además deja el sistema conforme.

3. Permite salirse, con coste explícito.
   Quien se sale asume la operación completa: su propia guardia, su propio parcheo.

4. Se mide por adopción, no por mandato.
   Si hay que obligar, el camino no es lo bastante bueno.
```

El punto 3 es el que evita el fracaso más común. Prohibir salirse convierte la plataforma en un obstáculo y genera infraestructura en la sombra —que es peor, porque nadie la ve—. Permitir salirse con el coste operativo explícito hace que la mayoría elija el camino por interés propio.

Métrica de salud de una plataforma, en una línea:

```
adopción = equipos que usan el camino dorado / equipos totales
```

Por debajo del 60 % hay que preguntar a los que no lo usan por qué, no insistir. La respuesta suele ser concreta: no soporta un tipo de carga, la latencia de aprovisionamiento es alta, o el equipo perdió algo que antes controlaba.

5. Olas con criterio de salida, no con fechas

Una adopción por fechas produce que la última ola herede toda la deuda de las anteriores. Por olas con criterio de salida verificable, cada una corrige lo aprendido:

```
Ola 0 · Fundación          cuentas, identidad federada, red, registro de auditoría
  salida: una carga trivial desplegada extremo a extremo sin secretos permanentes

Ola 1 · Piloto (1-2 cargas, sin criticidad)
  salida: RTO y RPO medidos; coste unitario conocido; runbook probado por alguien
          ajeno al equipo que lo escribió

Ola 2 · Camino dorado (3-5 cargas similares)
  salida: la tercera carga se despliega sin intervención de la plataforma
          y en menos de la mitad de tiempo que la primera

Ola 3 · Escala
  salida: adopción > 60 %; ningún equipo bloqueado esperando a la plataforma

Ola 4 · Cargas difíciles
  las que exigen reescritura o tienen dependencias legadas
```

El criterio de la ola 2 es el más revelador: si la tercera carga sigue necesitando ayuda manual, el camino dorado no existe todavía y escalar solo multiplicará el trabajo manual.

Y la ola 4 va al final por una razón deliberada: las cargas difíciles consumen mucho tiempo y enseñan poco reutilizable. Empezar por ellas —tentador, porque suelen ser las más visibles— agota el presupuesto político antes de tener nada que mostrar.

La señal de que hay que parar y corregir, en cualquier ola: el tiempo por carga no baja. Si migrar la carga número 8 cuesta lo mismo que la número 2, no se está construyendo capacidad reutilizable; se está haciendo el mismo trabajo ocho veces.

Ejemplo trabajado

CloudShop lleva ocho meses migrando. Han pasado 6 de 40 cargas y el equipo de plataforma está saturado. Se diagnostica el modelo operativo antes de pedir más gente.

Mediciones del estado actual:

```
cargas migradas                          6 de 40
tiempo por carga: nº 1                   6 semanas
tiempo por carga: nº 6                   5,5 semanas     ← no baja
tickets al equipo de plataforma/semana   47
tiempo del equipo en tickets             68 %
formas distintas de desplegar             6              ← una por carga
adopción del camino dorado               0 % (no existe)
```

Dos señales de agotamiento a la vez. El tiempo por carga no baja: no se está construyendo capacidad reutilizable. Y el 68 % del tiempo en tickets significa que la «plataforma» es un equipo centralizado con otro nombre.

Se revisa la estructura organizativa:

```
Desarrollo (4 equipos) ■■ticket■■■ Plataforma (5 personas) ■■ticket■■■ Seguridad (2)
```

Tres silos con tickets entre ellos. Por la ley de Conway, la arquitectura resultante replicará esa estructura: ningún servicio podrá desplegarse sin coordinar tres equipos, que es exactamente lo que se observa en las 6 semanas por carga.

Desglose de los 47 tickets semanales:

```
crear entorno o cuenta                    14        sí
permisos de IAM                           12        sí, con plantillas
reglas de red y DNS                        9        sí
certificados                               6        sí
casos genuinamente nuevos                  6        no
```

41 de 47 son repetición. No hace falta más gente: hace falta convertir esos cuatro tipos en autoservicio.

Intervención, en dos frentes a la vez:

```
ORGANIZATIVO (Conway inverso)
```

```
  seguridad deja de ser puerta y pasa a equipo habilitador:
  escribe las políticas como código, no revisa cada cambio
  los 4 equipos de desarrollo asumen la guardia de lo suyo

PLATAFORMA (camino dorado del 80 %)
  1 comando crea entorno + cuenta + red + rol federado + certificado
  cubre servicios HTTP sin estado: 31 de las 34 cargas restantes
  las 3 que no encajan se salen, con su operación explícitamente propia
```

Resultado seis meses después:

```
tiempo por carga (nº 20)              5,5 sem     4 días
tickets/semana                         47          9
tiempo del equipo en tickets           68 %        18 %
formas de desplegar                     6          1 + 3 excepciones
adopción del camino dorado              0 %        84 %
cargas migradas                         6          27
```

Se migraron 21 cargas en seis meses frente a 6 en ocho, con el mismo número de personas. El cuello no era capacidad: era que cada carga se resolvía a mano porque la estructura organizativa lo exigía.

Límite declarado: las 3 cargas fuera del camino dorado —dos con dependencias legadas y una con requisitos de latencia especiales— siguen consumiendo tiempo desproporcionado. Se dejan deliberadamente para el final, y sus equipos asumen su operación completa. Intentar que el camino dorado las cubriera habría retrasado el 84 % restante.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-01-cloud-principles-strategy-adoption/022-cloud-adoption-framework-y-modelo-operativo/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 backlog-de-adopcion 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 backlog-de-adopcion 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 adoptan microservicios y ninguno se puede desplegar de forma independiente	La organización sigue en silos funcionales: la arquitectura replica la comunicación	Reorganiza por dominio antes de comprometerte con la arquitectura; Conway no se evita con disciplina.
El equipo de plataforma dedica más de la mitad del tiempo a tickets	El camino dorado no cubre los casos reales, así que todos se salen y piden ayuda	Clasifica los tickets; si más del 80 % es repetición, conviértelos en autoservicio antes de contratar.
Migrar la carga número 8 cuesta lo mismo que la número 2	No se está construyendo capacidad reutilizable: se repite el mismo trabajo	Fija criterios de salida por ola; si el tiempo por carga no baja, para y construye antes de seguir.
Existe una plataforma en autoservicio y los equipos no la usan	No resuelve sus casos o les quitó control sin compensación	Pregunta a quienes no la usan; por debajo del 60 % de adopción el problema es el producto, no la disciplina.
Se prohíbe salirse del camino dorado y aparece infraestructura en la sombra	La obligatoriedad empuja fuera del radar en vez de fuera del camino	Permite salirse con el coste operativo explícito: guardia y parcheo propios.

■ 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 arquitectura produce una organización con tres silos funcionales que pide microservicios, y por qué?
2. ¿Qué tres señales medibles indican que un modelo operativo agotó su rango de tamaño?
3. ¿Cuál de los tres tipos de carga cognitiva debe eliminar una plataforma y cuál debe proteger?
4. ¿Por qué el camino dorado debe ser voluntario, y qué pasa si se hace obligatorio?
5. ¿Por qué las cargas más difíciles se dejan para la última ola en vez de atacarlas primero?

Referencias

- Conway, M. (1968). How Do Committees Invent?. Datamation 14(5), 28-31. <https://www.melconway.com/Home/CommitteesPaper.html>
- Skelton, M. y Pais, M. (2019). Team Topologies — cuatro tipos de equipo, carga cognitiva y maniobra de Conway inversa.
- Forsgren, N., Humble, J. y Kim, G. (2018). Accelerate — evidencia sobre estructura organizativa y rendimiento de entrega.
- Microsoft (2024). Cloud Adoption Framework — fases de estrategia, planificación, preparación y adopción. <https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/>
- AWS (2024). Cloud Adoption Framework — seis perspectivas, incluidas las de personas y gobierno. <https://aws.amazon.com/cloud-adoption-framework/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 01 en PDF · Manual integral

Anterior	Índice	Siguiente
← 021 · Well-Architected y atributos de calidad	Parte 01 · Programa	023 · Descubrimiento y clasificación de workloads →

Evaluación — 022 Cloud Adoption Framework y modelo operativo

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 backlog-de-adopcion 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.

023 — Descubrimiento y clasificación de workloads

Parte: 01 — Principios, estrategia y adopción cloud Nivel: inicial-intermedio · Horas estimadas: 4 Laboratorio: migration · Estado: EXECUTABLECORE

Propósito

Levantar un inventario de cargas que sirva para decidir, no para archivar. La mayoría de migraciones empieza con una hoja de cálculo de servidores y fracasa porque los servidores no son la unidad de decisión: lo son las cargas, con sus dependencias, sus datos y sus restricciones. Esta clase produce el insumo del ADR de la clase 024.

Resultados de aprendizaje

Al finalizar podrás:

1. Descubrir dependencias reales por observación de tráfico en vez de por documentación o entrevistas.
2. Clasificar cada carga en una de las siete estrategias de migración con un criterio explícito.
3. Detectar los grupos de cargas que deben moverse juntas y por qué partirlos rompe el sistema.
4. Priorizar por valor y dificultad, situando cada carga en un cuadrante con consecuencia clara.
5. Reconocer los datos que hacen inviable una estrategia antes de comprometerse con ella.

Conceptos centrales

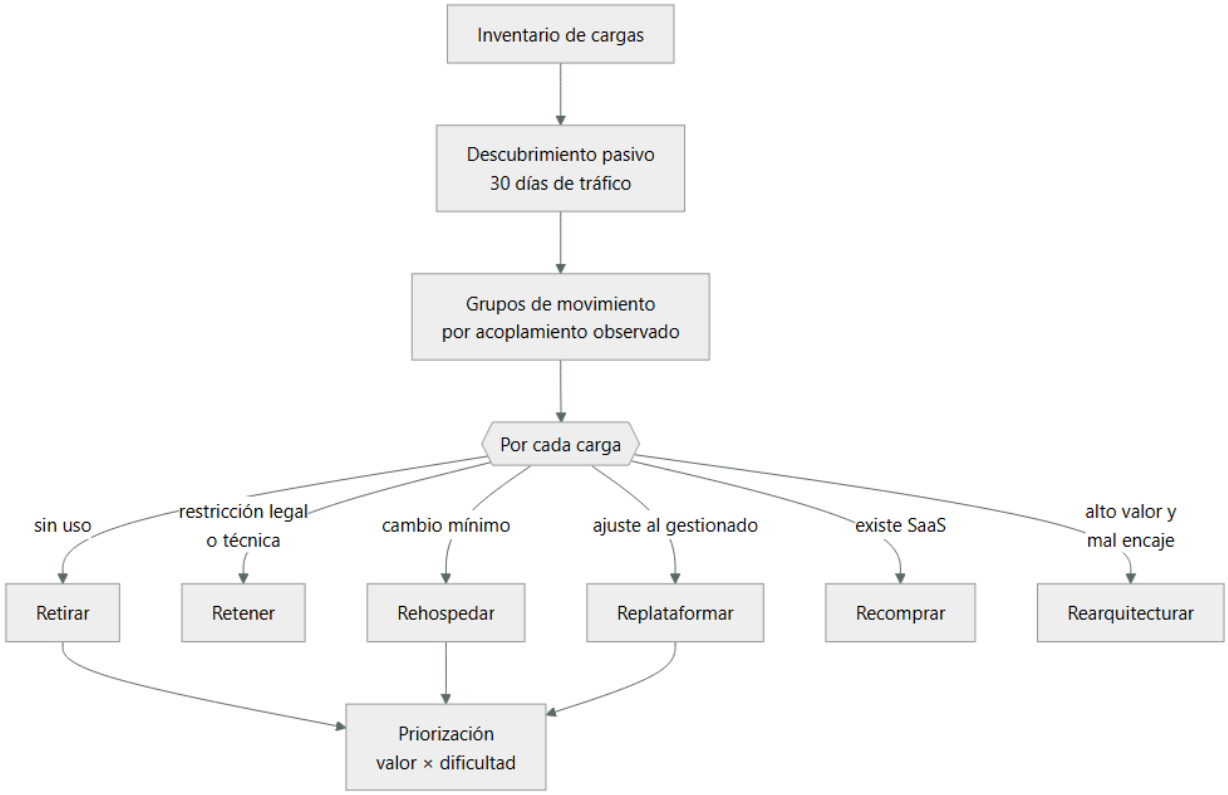
Concepto	Comprensión verificable
carga de trabajo	Conjunto de recursos y código que entrega una capacidad de negocio identificable. Es la unidad de decisión de una migración: los servidores son un detalle de implementación.
grupo de movimiento	Conjunto de cargas tan acopladas que deben migrarse a la vez. Determinarlo mal produce llamadas que cruzan la frontera con latencia y coste inesperados.
descubrimiento pasivo	Identificación de dependencias observando el tráfico real durante un periodo representativo, en lugar de preguntar. Encuentra lo que nadie recuerda y lo que nadie quiere admitir.
gravedad de los datos	Tendencia de las aplicaciones a permanecer cerca de los datos que consumen. Cuanto mayor es el volumen, más caro es moverlo y más atrae a todo lo demás.
las 7 R	Estrategias de migración: retirar, retener, reubicar, rehospedar, replataformar, recomprar y rearquitecturar. Ordenadas de menor a mayor coste y beneficio potencial.

Modelo mental

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

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 unidad de decisión es la carga, no el servidor

Un inventario de servidores responde «qué tenemos» y no responde ninguna de las preguntas que importan: qué depende de qué, qué se puede mover solo, qué se rompe si se parte.

El registro mínimo por carga, con lo que cada campo decide:

Campo	Decide
Capacidad de negocio que entrega	Si merece migrarse o retirarse
Dependencias entrantes y salientes	Con qué debe moverse
Volumen y clasificación del dato	Coste de traslado y restricciones legales
Perfil de tráfico	Modelo de servicio adecuado (clase 015)
Criticidad y RTO/RPO exigidos	Orden en las olas
Dueño técnico y de negocio	Quién decide y quién acepta el riesgo
Licencias y su portabilidad	Coste oculto que puede invertir el caso

La penúltima fila es la que más se olvida y la que más incidentes causa: una carga sin dueño de negocio identificable no se puede migrar, porque nadie puede aprobar la ventana de indisponibilidad ni aceptar el riesgo. Encontrar esas cargas huérfanas es uno de los resultados más valiosos del inventario, y a menudo revela candidatas a retirada.

La última fila puede invertir un caso de negocio completo: una licencia por núcleo físico que no se puede trasladar, o que en la nube se cuenta de otra forma, convierte una migración rentable en ruinosa. Hay que comprobarlo antes, no después.

2. Descubrir dependencias observando, no preguntando

Preguntar al equipo produce un mapa incompleto por tres razones que no son mala fe: nadie conoce el sistema entero, las integraciones antiguas se olvidan, y algunas conexiones las creó alguien que ya no está.

El descubrimiento pasivo observa el tráfico real:

```
# Conexiones establecidas con su proceso, agrupadas por destino
$ ss -tnp state established \
  | awk 'NR>1 {split($4,l,":"); split($5,r,":"); print r[1]":"r[2], $6}' \
  | sort | uniq -c | sort -rn | head -20
```

Un periodo de al menos 30 días es imprescindible, y la razón es concreta: los procesos de cierre mensual, las conciliaciones y los informes periódicos solo aparecen una vez al mes. Un descubrimiento de una semana pierde exactamente las dependencias que más duelen al fallar, porque son las que nadie ejercita a diario.

Lo que el descubrimiento encuentra y las entrevistas no:

```
- servicios que nadie sabía que seguían activos
- clientes externos consumiendo una API que se creía interna
- una tarea programada que escribe en una base de datos de otro sistema
- conexiones salientes a proveedores olvidados
- tráfico hacia una IP sin dueño identificable
```

El último caso es habitual y hay que resolverlo antes de migrar: una dependencia sin dueño es una llamada que dejará de funcionar sin que nadie sepa qué se rompió.

Y una advertencia sobre el método: observar tráfico en producción puede capturar datos sensibles. Hay que limitar la captura a metadatos de conexión —origen, destino, puerto, volumen— y nunca a contenido.

3. Las siete estrategias, con su criterio de elección

Estrategia	Qué es	Cuándo	Coste	Beneficio
Retirar	Apagar	Sin uso medido en 90 días	Mínimo	Alto: elimina coste y riesgo
Retener	Dejar donde está	Restricción legal, técnica o retirada próxima	Nulo	Ninguno, y es correcto
Reubicar	Mover la VM tal cual al hipervisor gestionado	Se necesita salir del centro de datos ya	Bajo	Bajo
Rehospedar	Mover sin cambios	Plazo corto, carga estándar	Bajo	Bajo-medio
Replataformar	Ajustes para usar servicios gestionados	Base de datos o cola sustituibles	Medio	Medio-alto
Recomprar	Sustituir por SaaS	Existe producto y no es diferencial	Medio	Alto si encaja
Rearquitecturar	Rediseñar	Alto valor y mal encaje con el modelo actual	Alto	Alto, con riesgo

Dos errores de selección que se repiten:

Rearquitecturar todo. Es la estrategia más cara y más lenta, y solo se justifica cuando la carga es de alto valor y su arquitectura actual impide obtener el beneficio. Aplicarla a una aplicación interna que usan 40 personas consume el presupuesto que necesitaban veinte cargas más rentables.

No retirar nada. Es la estrategia con mejor relación coste-beneficio y la que menos se usa, porque exige admitir que algo ya no sirve. Un inventario típico encuentra entre un 10 % y un 30 % de cargas sin uso real; migrarlas cuesta dinero y no aporta nada.

La comprobación de retirada debe ser por medición, no por opinión: sin conexiones entrantes durante 90 días, incluido un cierre mensual completo.

4. Grupos de movimiento: qué se rompe al partir

Dos cargas que intercambian mucho tráfico con baja tolerancia a latencia forman un grupo y deben moverse juntas. Separarlas produce un fallo silencioso: todo funciona en las pruebas y se degrada en producción.

La aritmética de partir un grupo mal:

```
carga A llama a carga B: 45 veces por petición de usuario
latencia en el mismo centro de datos:  0,4 ms → 45 × 0,4 =  18 ms
latencia si B se queda y A migra:       28 ms → 45 × 28  = 1.260 ms
```

De 18 ms a 1,26 segundos. El patrón de 45 llamadas por petición —una consulta dentro de un bucle— es invisible mientras la latencia es de 0,4 ms y catastrófico en cuanto sube. La migración no causó el problema de diseño: lo reveló.

Criterio para formar grupos, a partir del descubrimiento:

```
forman grupo si:
  llamadas por petición > 10, o
  volumen entre ellas > 1 GB/día, o
  comparten transacciones con consistencia fuerte, o
  comparten la misma base de datos

pueden separarse si:
  la comunicación es asíncrona y tolera segundos
  el volumen es bajo
  hay contrato estable entre ambas
```

La tercera condición es la más rígida: una transacción que abarca dos sistemas no se puede partir entre nubes sin cambiar el modelo de consistencia, que es rearquitecturar y no rehospedar. Descubrirlo durante la migración es descubrirlo tarde.

Y existe una salida intermedia legítima: partir el grupo y aceptar la degradación temporal durante una ventana corta, con las dos partes reunidas al final de la ola. Lo que no funciona es partirlo indefinidamente.

5. La gravedad de los datos decide más que la aplicación

Cuanto mayor es el volumen de datos, más caro es moverlo y más atrae hacia sí a todo lo que lo consume. Esa es la razón por la que las migraciones se planifican desde los datos hacia arriba y no al revés.

El tiempo de traslado por red tiene un suelo aritmético:

```
tiempo = volumen / ancho_de_banda_efectivo

40 TB por un enlace de 1 Gbit/s al 70 % de eficiencia:
  40.000 GB × 8 bit / (1 Gbit/s × 0,70) = 457.143 s ≈ 5,3 días
```

Cinco días de transferencia continua, durante los cuales los datos siguen cambiando. Por eso existen dos mecanismos que hay que considerar desde el inventario:

- Transferencia física: dispositivos que el proveedor envía, se cargan y se devuelven. Para decenas de TB suele ser más rápido, y el cálculo es sencillo: si el envío tarda menos que la red, gana.
- Replicación continua con corte final: se replica en caliente durante días o semanas y se hace un corte corto al final. Es lo que permite ventanas de indisponibilidad de minutos en lugar de días.

Y una restricción que no es técnica: los datos con residencia obligatoria no se mueven, por mucho ancho de banda que haya. Detectarlo en el inventario es lo que evita diseñar una arquitectura completa que después resulta ilegal.

La consecuencia práctica sobre el orden de las olas: una carga cuyos datos no se pueden mover fija la posición de todo su grupo. No es la aplicación la que decide dónde va: es el dato.

Ejemplo trabajado

CloudShop inventaría sus 40 cargas antes de decidir el plan de migración. El resultado reordena por completo el plan inicial, que era «mover todo tal cual en seis meses».

Descubrimiento pasivo, 34 días para cubrir un cierre mensual completo:

```
cargas declaradas por los equipos          40
cargas detectadas con tráfico              43
  → 3 servicios activos que nadie declaró
cargas declaradas SIN tráfico entrante      7
  → candidatas a retirada
dependencias declaradas                    58
dependencias observadas                    91
  → 33 dependencias que nadie recordaba (36 %)
```

Las 33 dependencias no documentadas son el hallazgo que justifica el método: una de cada tres conexiones reales no estaba en ningún diagrama.

Clasificación de las 43 cargas:

```
Retirar                        7   sin tráfico entrante en 34 días, incluido cierre
Retener                        3   1 por residencia de datos, 2 por retirada en 2027
Rehospedar                    18   estándar, plazo corto
Replataformar                  9   base de datos a gestionada, cola a gestionada
Recomprar                      4   correo interno, gestor documental, wiki, tickets
Rearquitecturar                2   catálogo y pedidos: alto valor y mal encaje
```

Las 7 retiradas ahorran 41.000 USD anuales sin migrar nada. Es el mejor retorno del proyecto entero y estuvo disponible desde el primer día.

Grupos de movimiento detectados:

```
G1  pedidos + inventario + pagos     transacciones compartidas → INSEPARABLE
G2  catálogo + búsqueda              820 MB/día entre ellos    → juntos
G3  informes + almacén de datos      lote nocturno             → separables
G4  15 cargas independientes         sin acoplamiento          → una a una
```

Se detecta a tiempo un error del plan original, que ponía inventario en la ola 1 y pedidos en la ola 3:

```
llamadas de pedidos a inventario por transacción:     45
latencia actual (mismo centro de datos):             0,4 ms → 18 ms totales
latencia si se separan (nube ↔ centro de datos):      28 ms → 1.260 ms
presupuesto de la petición:                          300 ms
```

Habrían roto el SLO por un factor de 4 durante los dos meses entre olas. El plan se corrige: G1 se mueve completo en una sola ventana.

Gravedad de los datos sobre las dos cargas mayores:

```
almacén de datos históricos      38 TB   residencia obligatoria en Chile
  → RETENER; fija la posición de los informes que lo consumen
base de datos de pedidos         2,4 TB  sin restricción
  → por red al 70 % de 1 Gbit/s: 7,6 h
  → con replicación continua + corte: ventana de 12 min
```

Priorización final por valor y dificultad:

```
7 retiradas                        alto     mínima     0   ← inmediato
4 recompras SaaS                   alto     baja       1
15 cargas independientes           medio    baja       1-2
G2 catálogo + búsqueda             alto     media      2
G1 pedidos + inventario + pagos    alto     ALTA       3   ← una ventana
2 rearquitecturas                  alto     ALTA       4
G3 informes                        bajo     alta       4   ← atado al dato retenido
```

Cambio respecto al plan inicial: de «40 cargas en 6 meses» a «7 retiradas esta semana, 19 cargas fáciles en 3 meses, y las 3 difíciles con su ventana propia». El inventario no retrasó el proyecto: evitó que la ola 3 rompiera producción y encontró 41.000 USD anuales de ahorro inmediato.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-01-cloud-principles-strategy-adoption/023-descubrimiento-y-clasificacion-de-workloads/lab.py
```

El laboratorio selecciona el motor de práctica migration 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 inventario-de-workloads 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 inventario, dependencias, riesgo y oleadas. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye inventario-de-workloads 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
Aparecen dependencias desconocidas durante la migración	El mapa se construyó por entrevistas; nadie conoce el sistema completo	Descubrimiento pasivo de al menos 30 días para capturar los procesos mensuales.
Una carga migrada funciona en pruebas y se degrada en producción	Se partió un grupo de movimiento con muchas llamadas por petición	Agrupa por llamadas por petición y volumen; una transacción compartida es inseparable.
Se migran cargas que nadie usa	No se midió el uso real antes de decidir la estrategia	Retira lo que no tenga tráfico entrante en 90 días; es el mayor retorno del proyecto.
El caso de negocio se invierte al llegar la factura de licencias	No se comprobó la portabilidad de las licencias ni cómo se cuentan en la nube	Incluye licencias en el inventario y verifica su modelo de conteo antes de decidir.
Una arquitectura completa resulta inviable por residencia de datos	La restricción legal se descubrió después de diseñar	Clasifica los datos en el inventario; un dato que no se mueve fija la posición de todo su grupo.

■ Seguridad, ética y costo

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

■ Preguntas de comprobación

1. ¿Por qué el descubrimiento pasivo necesita al menos 30 días, y qué dependencias se pierden con una semana?
2. Dos cargas hacen 45 llamadas por petición a 0,4 ms. ¿Qué latencia total tendrán si se separan a 28 ms y qué implica para un SLO de 300 ms?
3. ¿Cuál de las siete estrategias tiene mejor relación coste-beneficio y por qué casi nunca se usa?
4. ¿Qué condición hace que dos cargas sean inseparables incluso con poco tráfico entre ellas?
5. 40 TB por un enlace de 1 Gbit/s al 70 % de eficiencia: ¿cuánto tarda y qué dos alternativas existen?

Referencias

- AWS (2024). Migration strategies: the 7 Rs — criterios de elección por estrategia. <https://docs.aws.amazon.com/prescriptive-guidance/latest/large-migration-guide/migration-strategies.html>
- Microsoft (2024). Cloud Adoption Framework: assess workloads — inventario, dependencias y clasificación. <https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/plan/discover>
- McCrory, D. (2010). Data Gravity in the Clouds — por qué los datos atraen a las aplicaciones. <https://datagravitas.com/2010/12/07/data-gravity-in-the-clouds/>
- Google Cloud (2024). Migration to Google Cloud: assess and discover workloads — descubrimiento y agrupación. <https://cloud.google.com/architecture/migration-to-gcp-getting-started>
- Newman, S. (2019). Monolith to Microservices, caps. 2-3 — identificar fronteras y grupos acoplados antes de partir.
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 01 en PDF · Manual integral

Anterior	Índice	Siguiente
← 022 · Cloud Adoption Framework y modelo operativo	Parte 01 · Programa	024 · Proyecto: decisión de migración sustentada con ADR →

Evaluación — 023 Descubrimiento y clasificación de workloads

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 inventario-de-workloads 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.

024 — Proyecto: decisión de migración sustentada con ADR

Parte: 01 — Principios, estrategia y adopción cloud Nivel: inicial-intermedio · Horas estimadas: 8 Laboratorio: capstone · Estado: EXECUTABLECORE

Propósito

Cerrar la parte produciendo el artefacto que la sostiene: un registro de decisión de arquitectura que integra todo lo anterior —definición, topología, modelo de servicio, identidad, responsabilidad, coste y capacidad— en un documento que alguien pueda cuestionar dentro de tres años. Un ADR no documenta lo decidido: documenta por qué, y sobre todo qué se descartó y bajo qué condición habría que revisarlo.

Resultados de aprendizaje

Al finalizar podrás:

1. Escribir un ADR con contexto, opciones evaluadas, decisión, consecuencias y criterio de revisión.
2. Justificar el descarte de al menos una alternativa con datos y no con preferencia.
3. Declarar las consecuencias negativas de la opción elegida con la misma claridad que las positivas.
4. Definir la condición observable que obligaría a reabrir la decisión.
5. Distinguir una decisión reversible de una que no lo es, y ajustar el rigor del proceso a esa diferencia.

Conceptos centrales

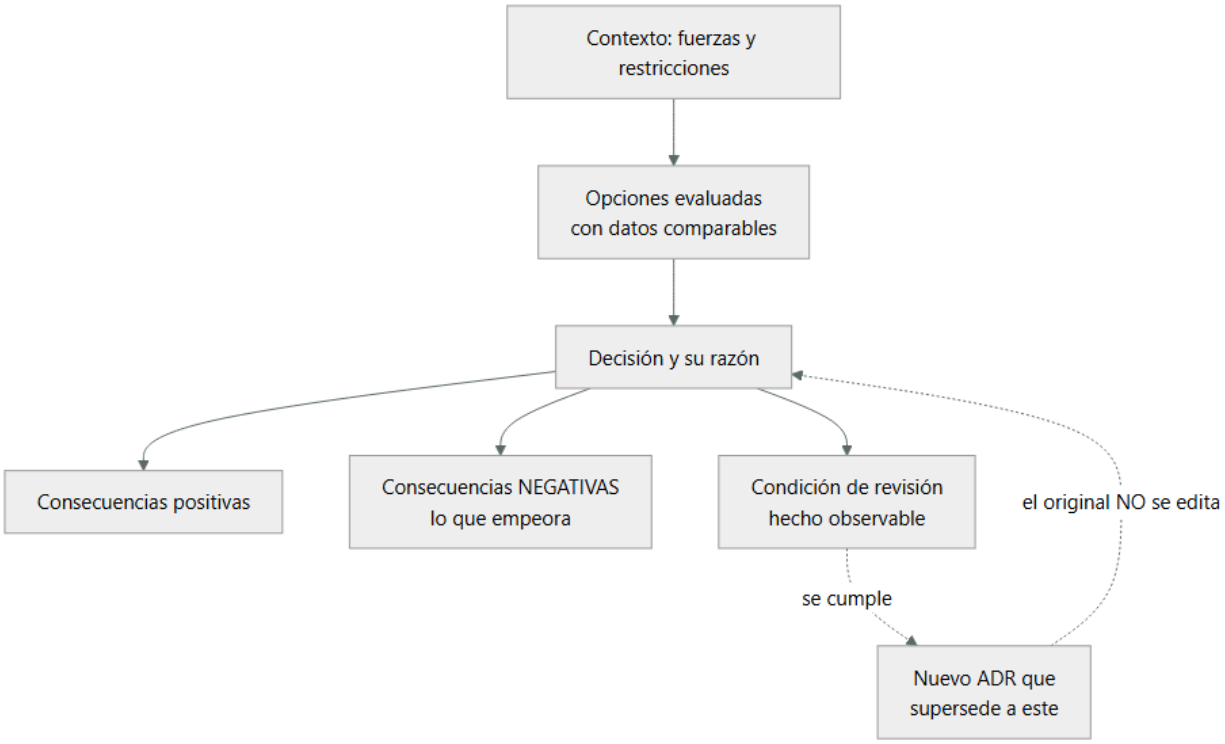
Concepto	Comprensión verificable
ADR	Documento breve e inmutable que registra una decisión de arquitectura con su contexto y consecuencias. Se numera, se versiona junto al código y no se edita: se sustituye por otro que lo supersede.
estado del ADR	Propuesto, aceptado, rechazado, obsoleto o superseded. La cadena de estados es lo que permite reconstruir la evolución del razonamiento, no solo el resultado final.
decisión irreversible	Aquella cuyo coste de deshacer es un múltiplo del de tomarla. Merece más análisis, más consenso y una condición de revisión explícita; las reversibles merecen lo contrario.
consecuencia	Lo que la decisión hace cierto a partir de ahora, tanto favorable como desfavorable. Un ADR sin consecuencias negativas no fue escrito con honestidad.
condición de revisión	Hecho observable y medible que obligaría a reabrir la decisión. Convierte un documento estático en un compromiso vivo.

Modelo mental

La nube es un modelo operativo de acceso bajo demanda, no un lugar mágico ni el centro de datos de otra persona.

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 ADR registra el razonamiento, no el resultado

Dentro de tres años nadie recordará por qué se eligió aquello, y la persona que lo decidió probablemente ya no esté. Lo que se pierde no es la decisión —esa está en el código— sino las restricciones que la hicieron razonable.

Sin ese registro ocurren dos patologías simétricas:

1. Se revierte una buena decisión porque su motivo ya no es visible. Alguien ve una elección que hoy parece subóptima y la cambia, sin saber que respondía a una restricción que sigue vigente.
2. Se mantiene una mala decisión porque nadie sabe si aún aplica. La restricción desapareció hace dos años y el diseño sigue pagándola.

La estructura mínima, en el formato de Nygard:

```
# ADR-0012: Base de datos gestionada multi-zona para pedidos

**Estado:** Aceptado · 2026-08-01
**Decide:** equipo de plataforma · **Acepta el riesgo:** dirección de tecnología

## Contexto
[Las fuerzas en juego, con números. No la solución.]

## Opciones consideradas
[Al menos dos alternativas reales, comparadas con los mismos criterios.]

## Decisión
[Qué se elige y por qué, ligado a los datos del contexto.]

## Consecuencias
[Positivas y NEGATIVAS. Sin las segundas no es honesto.]

## Condición de revisión
[Hecho observable que obligaría a reabrir esto.]
```

Dos reglas que lo mantienen útil: vive junto al código, no en una wiki que se abandona; y no se edita. Si la decisión cambia, se escribe otro ADR que la supersede y se enlazan. El historial de razonamiento es el valor, y editar lo destruye.

2. El contexto va con números y sin solución dentro

El error más común es escribir la solución en el contexto: «Necesitamos una base de datos gestionada multi-zona porque…». Eso no es contexto, es la decisión disfrazada, y elimina la posibilidad de evaluar alternativas.

El contexto debe enumerar fuerzas medibles:

```
## Contexto

- La plataforma sirve 980.000 pedidos/mes con crecimiento del 18 % anual.
- Requisito de disponibilidad: 99,95 % mensual → 21,6 min de caída permitida.
- Requisito de recuperación: RTO 60 min, RPO 15 min.
- Presupuesto de infraestructura: ≤ 0,018 USD por pedido.
- El equipo son 4 personas sin especialista en bases de datos.
- Restricción legal: los datos personales no salen del territorio nacional.
- La región disponible tiene 3 zonas.
```

Siete fuerzas, todas verificables, ninguna sugiere una solución. Con este contexto otra persona podría llegar a una decisión distinta y defenderla, que es exactamente la prueba de que el contexto está bien escrito.

La quinta línea merece comentario: las restricciones de equipo son tan legítimas como las técnicas y casi nunca se escriben. «No tenemos especialista en bases de datos» descarta razonablemente opciones que exigirían operarla, y omitirlo hace que la decisión parezca arbitraria a quien lea el documento después.

Y conviene distinguir en el contexto lo que es dato de lo que es supuesto. El crecimiento del 18 % anual es una proyección, no un hecho; marcarlo como supuesto permite que la condición de revisión lo vigile.

3. Las opciones se comparan con los mismos criterios

Un ADR con una sola opción no es una decisión: es un anuncio. Y con opciones evaluadas por criterios distintos, la comparación no significa nada.

La tabla que hace verificable la elección:

```
## Opciones consideradas

| Criterio | A: gestionada multi-zona | B: autogestionada en VM | C: gestionada una zona |
|---|---|---|---|
| Disponibilidad | 99,995 % | 99,5 % | 99,9 % |
| ¿Cumple 99,95 %? | Sí | No | No |
| RPO | 5 min | según operación | 5 min |
| Coste mensual | 4.900 USD | 3.100 USD | 3.400 USD |
| Coste por pedido | 0,0050 | 0,0032 | 0,0035 |
| Carga operativa | Baja | **Alta: parcheo y réplica** | Baja |
| Personal necesario | 0,2 FTE | **1,0 FTE** | 0,2 FTE |
| Coste real con personal | 4.900 | **3.100 + 5.200 = 8.300** | 3.400 |
```

Las dos últimas filas son las que cambian el resultado y las que casi nunca se incluyen. La opción B parece la más barata por infraestructura y es la más cara en total, porque exige una persona dedicada. Sin esa fila, el ADR habría elegido mal y con apariencia de rigor.

El criterio de descarte debe ser explícito y ligado al contexto:

```
- **B se descarta** por dos motivos independientes: no alcanza el requisito de
  disponibilidad (99,5 % frente a 99,95 %) y exige 1,0 FTE que el equipo de
  4 personas no tiene.
- **C se descarta** por disponibilidad (99,9 % → 43 min/mes frente a 21,6 permitidos),
  pese a ser 1.500 USD/mes más barata.
```

Cada descarte cita el número del contexto que lo justifica. Eso es lo que permite que dentro de tres años alguien compruebe si el motivo sigue vigente.

4. Las consecuencias negativas son la parte honesta

Un ADR que solo enumera ventajas describe una compra, no una decisión. Toda elección de arquitectura empeora algo, y escribirlo tiene dos efectos prácticos: obliga a mirarlo antes de comprometerse, y evita que en el futuro se presente como un fallo lo que fue una elección consciente.

```
## Consecuencias

### Positivas
- Disponibilidad de 99,995 %, con margen sobre el requisito.
```

```
- RPO de 5 min frente a los 15 exigidos.
- El parcheo del motor deja de consumir tiempo del equipo.

### Negativas
- **Coste por pedido de 0,0050 USD frente a 0,0035 de la opción C**: 43 % más
  caro por un requisito de disponibilidad que ninguna medición de negocio ha
  validado todavía.
- **Bloqueo de proveedor medio**: la conmutación automática y el punto de
  recuperación son propietarios. Migrar exigiría rediseñar la continuidad.
- **Pérdida de control sobre la ventana de parcheo**: se elige el intervalo,
  no la versión ni la fecha exacta.
- **El operador del proveedor tiene acceso técnico a los datos**, acotado por
  contrato pero real. Aceptado por dirección de tecnología.
```

La primera consecuencia negativa es la más valiosa del documento: admite que el requisito que justifica el sobrecoste no está validado. Eso convierte una suposición invisible en algo que alguien puede refutar con datos, y es justo lo que la condición de revisión debe vigilar.

La cuarta muestra otro patrón útil: nombrar un riesgo residual y quién lo acepta. Sin la segunda parte, el riesgo queda huérfano, que es como acaban la mayoría.

5. Ajustar el rigor a la reversibilidad

No todas las decisiones merecen el mismo proceso. Bezos las clasificó en dos tipos y la distinción es operativamente útil:

Tipo	Coste de deshacer	Proceso adecuado
Reversible	Bajo, días	Decide rápido, prueba, corrige
Irreversible	Alto o destructivo	ADR completo, consenso, condición de revisión

Ejemplos del programa, con su coste real de reversión:

```
reversible      elegir una biblioteca de registro          horas
reversible      umbrales del autoescalado                  minutos
semi            modelo de servicio: contenedor o función   semanas
IRREVERSIBLE    esquema y particionado de datos            meses + ventana
IRREVERSIBLE    formato de identificadores públicos        rompe a terceros
IRREVERSIBLE    región donde residen los datos             coste de egreso + legal
```

Aplicar el proceso pesado a una decisión reversible es la forma más común de que un equipo se vuelva lento sin ganar nada. Aplicar el ligero a una irreversible es cómo se acumulan las restricciones que dentro de dos años nadie puede quitar.

La condición de revisión es lo que mantiene vivo el ADR, y debe ser observable:

```
## Condición de revisión

Esta decisión se reabre si ocurre cualquiera de:

1. El coste de la base de datos supera el 35 % de la factura total.
2. Se mide durante dos trimestres que la disponibilidad real del negocio no
   se ve afectada por caídas menores de 40 min/mes (invalidaría el requisito
   que justifica el sobrecoste).
3. El volumen supera 5 M de pedidos/mes, donde el particionado obliga a
   revisar el modelo de datos completo.
```

Cada condición es un número comprobable, no una impresión. La segunda es la más valiosa: describe cómo se demostraría que la propia decisión fue innecesariamente cara, que es la prueba definitiva de que el ADR se escribió con honestidad.

Ejemplo trabajado

ADR-0012 de CloudShop, escrito con todo lo acumulado en la parte 01.

```
# ADR-0012: Base de datos gestionada multi-zona para pedidos

**Estado:** Aceptado · 2026-08-01
**Supersede:** ninguno · **Superseded por:** —
**Decide:** equipo de plataforma · **Acepta el riesgo:** dirección de tecnología
```

```
## Contexto

- 980.000 pedidos/mes, crecimiento proyectado del 18 % anual [SUPUESTO].
- Disponibilidad exigida 99,95 % mensual → 21,6 min de caída permitida [DATO].
- RTO 60 min, RPO 15 min, del plan de continuidad vigente [DATO].
- Presupuesto ≤ 0,018 USD/pedido; hoy el total es 0,0177 [DATO].
- Equipo de 4 personas, sin especialista en bases de datos [DATO].
- Datos personales con residencia nacional obligatoria [DATO, legal].
- La región disponible tiene 3 zonas [DATO].

## Opciones consideradas

| Criterio | A: gestionada multi-zona | B: autogestionada | C: gestionada 1 zona |
|---|---|---|---|
| Disponibilidad | 99,995 % | 99,5 % | 99,9 % |
| Caída mensual | 1,3 min | 3,6 h | 43 min |
| ¿Cumple 21,6 min? | Sí | No | **No** |
| RPO | 5 min | variable | 5 min |
| Infraestructura | 4.900 USD | 3.100 USD | 3.400 USD |
| Personal | 0,2 FTE | 1,0 FTE (5.200 USD) | 0,2 FTE |
| **Coste total** | **4.900** | **8.300** | 3.400 |

**B se descarta** por dos motivos independientes: incumple la disponibilidad
(3,6 h frente a 21,6 min) y exige 1,0 FTE inexistente. Es además la más cara
en total pese a ser la más barata en infraestructura.

**C se descarta** solo por disponibilidad: 43 min/mes frente a 21,6 permitidos.
Es 1.500 USD/mes más barata y sería la elección si el requisito cambiara.

## Decisión

Se adopta **A**: base de datos gestionada con réplica síncrona entre dos zonas
y recuperación a un instante activada.

Coste unitario resultante: 4.900 / 980.000 = 0,0050 USD/pedido; el total sube
de 0,0177 a 0,0192 USD/pedido, **por encima del presupuesto de 0,018**. Se
compensa con la optimización de transferencia identificada en la clase 011
(−2.300 USD/mes), que deja el unitario en 0,0169. Ambas acciones van juntas.

## Consecuencias

### Positivas
- 1,3 min/mes de caída frente a 21,6 permitidos.
- RPO de 5 min frente a 15 exigidos.
- El parcheo del motor sale del trabajo del equipo (≈ 0,8 FTE liberados).

### Negativas
- **43 % más cara que C** por un requisito de disponibilidad que ninguna
  medición de impacto de negocio ha validado.
- **Bloqueo medio**: conmutación y punto de recuperación son propietarios.
  Coste de salida estimado: 3.780 USD de egreso + ~35.000 de rediseño.
- Se elige la ventana de parcheo, no la versión ni la fecha.
- El operador del proveedor tiene acceso técnico a los datos, acotado por
  contrato. Riesgo aceptado por dirección de tecnología.
- **No sobrevive a un fallo regional completo.** Aceptado: la continuidad
  regional no es requisito hoy.

## Condición de revisión

1. El coste de la base supera el 35 % de la factura total.
2. Dos trimestres consecutivos sin impacto de negocio atribuible a caídas
   menores de 40 min/mes → invalidaría el requisito y haría preferible C.
3. El volumen supera 5 M pedidos/mes → obliga a revisar el particionado.
4. Aparece requisito de continuidad regional → reabre con opción multi-región.

## Verificación

- [ ] Restauración probada con RTO medido, trimestral (clase 019).
- [ ] Alerta si la retención baja de 7 días o se desactiva el punto de recuperación.
- [ ] Revisión de las condiciones 1 y 2 en el comité trimestral de arquitectura.
```

Lo que hace útil a este ADR no es la decisión —A era razonablemente evidente— sino tres cosas que un documento típico omite: la fila de personal que invierte el orden de coste de B, la admisión de que el requisito que justifica el sobrecoste no está validado, y una condición de revisión que describe cómo se demostraría que la decisión fue innecesariamente cara.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-01-cloud-principles-strategy-adoption/024-proyecto-decision-de-migracion-sustentada-con-adr/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 adr-de-migracion 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 adr-de-migracion 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 revierte una decisión y se rompe algo que nadie sabía que dependía de ella	El ADR no registró la restricción que la hacía razonable	Escribe el contexto con fuerzas medibles; el código dice qué se hizo, el ADR por qué.
El ADR presenta una sola opción y parece una decisión	Es un anuncio: sin alternativas comparadas no hay razonamiento verificable	Evalúa al menos dos opciones reales con los mismos criterios y cita el número que descarta cada una.
La opción más barata en infraestructura resulta la más cara en total	No se incluyó el coste de personal en la comparación	Añade carga operativa y FTE necesarios como criterio; suele invertir el orden.
Años después nadie sabe si la decisión sigue siendo válida	No hay condición de revisión observable	Define hechos medibles que obliguen a reabrir, incluido uno que demostraría que fue innecesaria.
Un equipo se vuelve lento decidiendo cosas triviales	Se aplica el proceso de decisión irreversible a decisiones reversibles	Clasifica por coste de deshacer: reversible se decide rápido y se corrige; irreversible merece el ADR completo.

■ Seguridad, ética y costo

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

■ Preguntas de comprobación

1. ¿Por qué un ADR no se edita cuando la decisión cambia, y qué se hace en su lugar?
2. ¿Cómo distingues un contexto bien escrito de uno que ya contiene la solución?
3. ¿Qué criterio incluido en la comparación invirtió el orden de coste entre las opciones A y B?
4. ¿Qué le falta a un ADR que solo enumera consecuencias positivas?
5. Escribe una condición de revisión que demostraría que la decisión tomada fue innecesariamente cara.

Referencias

- Nygard, M. (2011). Documenting Architecture Decisions — formato original de los ADR. <https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions>
- Keeling, M. y Runde, J. (2017). Share the Load: Distribute Design Authority with Architecture Decision Records. Agile Alliance. <https://www.agilealliance.org/resources/experience-reports/distribute-design-authority-with-architecture-decision-records/>
- Bezos, J. (2016). Letter to Shareholders — decisiones de tipo 1 y tipo 2 según su reversibilidad. <https://www.sec.gov/Archives/edgar/data/1018724/000119312516530910/d168744dex991.htm>
- Ford, N., Parsons, R. y Kua, P. (2017). Building Evolutionary Architectures, cap. 2 — funciones de aptitud y decisiones con condición de revisión.
- Richards, M. y Ford, N. (2020). Fundamentals of Software Architecture, cap. 19 — registro y comunicación de decisiones.
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 01 en PDF · Manual integral

Anterior	Índice	Siguiente
← 023 · Descubrimiento y clasificación de workloads	Parte 01 · Programa	025 · Organizations, cuentas, OU, SCP y landing zone →

Evaluación — 024 Proyecto: decisión de migración sustentada con ADR

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 adr-de-migracion 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.