Multi-Cloud Engineering Program

Parte 12 — Arquitectura cloud-native y sistemas distribuidos

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	12 de 23
Clases	145–156
Evaluaciones	12
Horas estimadas	52

Índice

Parte 12 — Arquitectura cloud-native y sistemas distribuidos

← Parte 11 · Índice completo · Parte 13 →

Descargar: Esta parte en PDF · Manual integral

Nivel: avanzado-experto · Clases: 12 · Duración sugerida: 6–8 semanas

Tomar decisiones de arquitectura con compensaciones explícitas y evidencia.

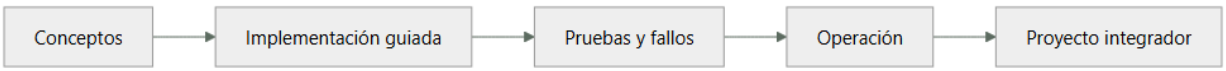
Resultados de la parte

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

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

Secuencia



Clases

ID	Clase	Laboratorio	Horas
145	Requisitos, restricciones y atributos de calidad	architecture	4
146	Twelve-Factor App y configuración cloud-native	architecture	4
147	DDD, bounded contexts y ownership de datos	architecture	4
148	Monolito modular, microservicios y función	decision	4
149	CAP, PACELC y consistencia por operación	distributed	4
150	Replicación, particionado y consenso	distributed	4
151	Fallos parciales y patrones de resiliencia	reliability	4
152	Service discovery, malla y comunicación	network	4
153	Contratos API, compatibilidad y evolución	api	4
154	Multi-tenancy, aislamiento y noisy neighbor	architecture	4
155	Rendimiento, costo, seguridad y operabilidad	decision	4
156	Proyecto: revisión de arquitectura 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 Native Patterns — Cornelia Davis.
- Building Microservices — Sam Newman.
- Release It! — Michael Nygard.
- Designing Data-Intensive Applications — Martin Kleppmann.

145 — Requisitos, restricciones y atributos de calidad

Parte: 12 — Arquitectura cloud-native y sistemas distribuidos Nivel: avanzado-experto · Horas estimadas: 4
Laboratorio: architecture · Estado: EXECUTABLECORE

Propósito

Abrir la parte de arquitectura por donde de verdad se decide: no por lo que el sistema hace, sino por cómo tiene que hacerlo y a qué precio. Dos sistemas con las mismas funciones y exigencias distintas de latencia, disponibilidad o consistencia no se parecen en nada por dentro. La clase enseña a escribir esas exigencias en forma comprobable en vez de adjetivos, a reconocer que se contradicen entre sí y por tanto se ordenan en vez de maximizarse, y a tratar las restricciones —presupuesto, plazos, normativa y tamaño del equipo— como lo que son: parte del diseño.

Resultados de aprendizaje

Al finalizar podrás:

1. Escribir exigencias como escenarios con números en vez de adjetivos.
2. Relacionar cada atributo con las decisiones de arquitectura que fuerza.
3. Ordenar los atributos cuando entran en conflicto, en vez de exigirlos todos.
4. Distinguir restricción de requisito y tratar el equipo como restricción.
5. Interrogar un requisito hasta saber qué pasa si no se cumple.

Conceptos centrales

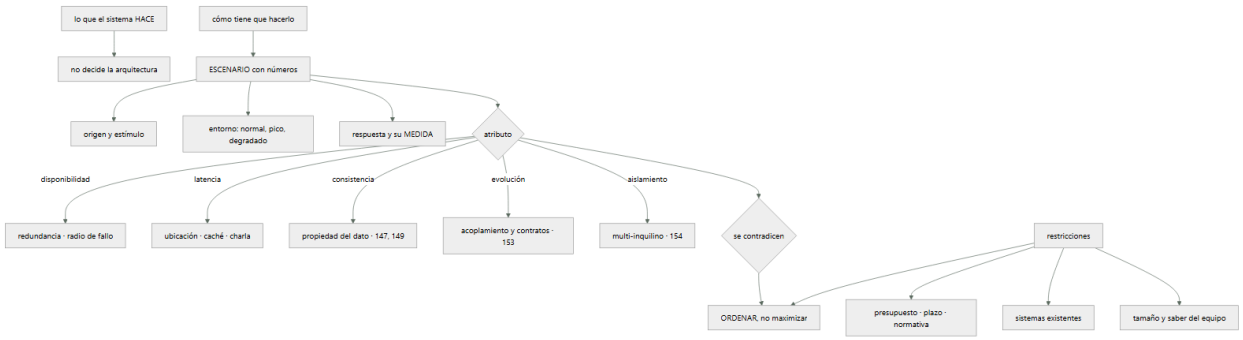
Concepto	Comprensión verificable
atributo de calidad	Propiedad del sistema que no es una función: latencia, disponibilidad, consistencia, evolución, coste, operabilidad.
escenario	Enunciado con origen, estímulo, entorno, respuesta y medida. Convierte un adjetivo en algo que se puede comprobar.
restricción	Condición dada que no se negocia: presupuesto, plazo, normativa, sistemas existentes, tamaño y conocimientos del equipo.
conflicto entre atributos	Mejorar uno empeora otro. Es la razón por la que se ordenan en vez de exigirlos todos.
orden de prioridad	Lista corta de atributos que ganan cuando hay que elegir. Una lista de doce igual de críticos no decide nada.
consecuencia del incumplimiento	Qué pasa de verdad si no se cumple un requisito. Es la pregunta que revela si el número es real.

Modelo mental

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

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 decide la arquitectura no es la función

Dos sistemas pueden tener la misma lista de funciones y no parecerse en nada:

```
una tienda con 200 pedidos al día
una tienda con 200 pedidos por segundo

mismas funciones
y ninguna decisión de arquitectura en común
```

Lo que las separa son los atributos de calidad, y su problema habitual es que se enuncian como adjetivos:

```
«tiene que ser escalable»          ¿cuánto, en cuánto tiempo, a qué coste?
«tiene que ser rápido»             ¿qué operación, con qué carga, qué percentil?
«tiene que ser seguro»             ¿frente a qué amenaza?          clase 136
«tiene que ser fiable»             ¿qué significa fallar?          clase 126
```

Ninguno de esos cuatro permite decidir nada. Y la corrección es escribirlos como escenarios, con cinco partes:

```
ORIGEN      quién o qué provoca la situación
ESTÍMULO    qué ocurre
ENTORNO     en qué condiciones: normal, pico, con una dependencia caída
RESPUESTA   qué debe hacer el sistema
MEDIDA      con qué número se comprueba
```

Y el mismo requisito, escrito bien:

```
mal   «el catálogo tiene que ser rápido»

bien  «un cliente que abre una ficha de producto (origen y estímulo)
       durante la campaña de noviembre, con 5.000 peticiones por
       segundo (entorno), recibe la página completa (respuesta)
       en menos de 500 ms en el percentil 99 (medida)»
```

Y la propiedad que lo hace útil: se puede comprobar. Con la clase 129 se mide, con la 126 se convierte en objetivo y con la 131 se ensaya el entorno degradado.

Y conviene escribir también los escenarios de cambio, que son los que deciden el acoplamiento:

```
«añadir un método de pago nuevo, sin tocar el servicio de pedidos,
 en menos de una semana y sin parada»

«cambiar el proveedor de envíos afectando a un solo servicio»

«dar de alta un cliente empresarial con sus propios datos aislados
 en menos de un día»
```

Y los de operación, que la parte 10 dejó claros y que casi nunca se escriben:

```
«cuando el servicio de pagos se degrada, el sistema sigue aceptando
 pedidos y alguien se entera en menos de dos minutos»
```

2. Qué decisión fuerza cada atributo

Los atributos que de verdad mueven la arquitectura, con lo que fuerzan y dónde se desarrolla en esta parte:

```
DISPONIBILIDAD
  fuerza redundancia, conmutación, radio de fallo, dependencias opcionales
  → y su techo lo fija la cadena de dependencias           clase 126
```

```
LATENCIA
  fuerza ubicación de los datos, caché, y sobre todo NÚMERO DE SALTOS
  → cada frontera de servicio añade red, serialización y cola

CAUDAL Y CRECIMIENTO
  fuerza particionado, ausencia de estado local y colas
                                                        clases 110, 150

CONSISTENCIA
  fuerza quién es dueño de cada dato y qué es síncrono
                                                        clases 147, 149

EVOLUCIÓN
  fuerza acoplamiento, contratos y fronteras de equipo
                                                        clases 148, 153

AISLAMIENTO
  fuerza el modelo multi-inquilino y el reparto de recursos
                                                        clase 154

OPERABILIDAD
  fuerza observabilidad, procedimientos y capacidad de intervenir
                                                        parte 10

COSTE
  atraviesa todos los anteriores                        clase 155
```

Y una observación que este programa ya ha demostrado tres veces y que conviene tener presente aquí: la latencia la deciden los saltos, no la velocidad de cada componente. La escalera de 340 consultas de la clase 124 y el abanico con rezagado son problemas de arquitectura, no de código.

Los conflictos, que son la parte que hay que aceptar:

```
consistencia fuerte      ←→  disponibilidad y latencia         clase 149
latencia baja            ←→  coste (réplicas, caché, cercanía)
aislamiento por inquilino ←→  coste y aprovechamiento           clase 154
flexibilidad             ←→  simplicidad y operabilidad
seguridad                ←→  facilidad de uso                  clase 133
evolución independiente  ←→  consistencia entre servicios
```

Y de ahí la regla central de la clase:

```
NO SE MAXIMIZAN: SE ORDENAN

una lista de tres atributos que ganan cuando hay conflicto
es una decisión
una lista de doce «todos críticos» es no haber decidido
```

Y el complemento honesto, que casi nunca se escribe y evita discusiones eternas:

```
lo que NO se está optimizando
  «no optimizamos para más de 10.000 clientes simultáneos»
  «no optimizamos para despliegues sin ninguna parada en la base»
  «no optimizamos coste por debajo de X mientras crezcamos»
```

Un diseño sin esa lista se defiende mal, porque cualquiera puede exigirle algo que nunca se propuso.

3. Restricciones, incluida la gente

Las restricciones no se negocian; se aceptan y se diseña con ellas.

```
PRESUPUESTO Y PLAZO       lo que hay y para cuándo
NORMATIVA                 residencia, retención, evidencia    clase 141
SISTEMAS EXISTENTES       lo que no se puede tocar o sustituir ahora
CONTRATOS                 proveedores con los que ya se trabaja
CONOCIMIENTOS DEL EQUIPO  lo que el equipo sabe operar
TAMAÑO DEL EQUIPO         cuánta gente hay para mantener esto
```

Y las dos últimas se ignoran sistemáticamente, aunque son de las más determinantes:

```
un sistema repartido en quince servicios con cuatro personas
→ nadie puede estar de guardia de quince cosas
→ y cada servicio necesita su canalización, su vigilancia y sus
  procedimientos: partes 08 y 10
```

Y la observación clásica que conviene enunciar sin misticismo:

```
la forma de comunicación de los equipos acaba reflejándose
en la forma del sistema
→ dos equipos que no hablan producirán una frontera dura entre
  sus partes, exista o no en el diseño
→ y un servicio compartido por cuatro equipos tendrá cuatro dueños,
  que es lo mismo que ninguno            ley 20
```

Y su uso práctico: si se quiere una frontera, hay que ponerle un equipo detrás; y si no hay equipos para tantas fronteras, hay que poner menos.

Interrogar un requisito es la técnica que más tiempo ahorra. Las preguntas, en orden:

```
1. ¿QUÉ PASA SI NO SE CUMPLE?
   si nadie sabe responder, el número no es real

2. ¿CUÁNDO HACE FALTA?
   «10 veces más carga» es distinto si es dentro de tres meses
   o dentro de tres años

3. ¿A QUÉ COSTE?
   cada nueve de disponibilidad tiene un precio     clase 126

4. ¿CON QUÉ FRECUENCIA OCURRE ESA SITUACIÓN?
   optimizar para un caso que se da una vez al año es caro

5. ¿QUÉ ESTAMOS DISPUESTOS A EMPEORAR PARA CONSEGUIRLO?
   si la respuesta es «nada», el requisito no se ha entendido
```

La primera y la quinta son las que separan un requisito de un deseo, y conviene hacerlas siempre, aunque incomoden.

4. Qué se entrega y para qué sirve

El resultado de esta clase es un documento corto que alimenta a todas las demás de la parte:

```
1. entre 5 y 12 escenarios, con números, priorizados
2. el orden de los atributos cuando hay conflicto: tres, no doce
3. lo que NO se optimiza, explícito
4. las restricciones, incluidas las de equipo
5. los supuestos, con quién los verifica              clase 140
```

Y cómo se usa después, que es lo que lo hace algo más que un trámite:

```
al elegir la división en servicios          clase 148
al decidir consistencia por operación       clase 149
al decidir replicación y particionado       clase 150
al fijar objetivos                          clase 126
al dimensionar y probar carga               clase 129
y al revisar una decisión, para saber si
  las premisas siguen siendo ciertas        clase 156
```

Y la propiedad que decide si sobrevive: los escenarios caducan.

```
el tráfico crece o decrece
aparece una normativa nueva
cambia el modelo de negocio
el equipo se dobla o se reduce a la mitad
→ y una arquitectura correcta para los escenarios de hace tres años
  puede ser incorrecta hoy sin que nada haya fallado
```

Por eso conviene revisarlos con la misma cadencia con la que se revisan las decisiones, y anotar en cada uno de qué fecha es y quién lo dio por bueno.

Y dos antipatrones frecuentes de esta materia:

```
DISEÑAR PARA UNA ESCALA QUE NO EXISTE
  «por si llegamos a un millón de usuarios»
  → se paga complejidad hoy por un escenario incierto
  → la pregunta 2 y la 4 lo resuelven

NO DISEÑAR PARA UNA ESCALA QUE SÍ LLEGA
  ignorar un crecimiento conocido porque «ya lo veremos»
  → y entonces se paga una migración: clases 110 y 114
```

Y el criterio entre los dos: diseñar para el escenario conocido y dejar abierta la puerta al siguiente, sabiendo cuáles son las decisiones que no se podrán deshacer —clave de partición, número de particiones, propiedad de los datos— y decidiendo esas con más cuidado que las demás.

Y la lista de comprobación de la clase:

- cada exigencia está escrita como escenario con medida
- hay escenarios de cambio y de operación, no solo de rendimiento
- los atributos están ordenados: tres ganan cuando hay conflicto
- está escrito lo que NO se optimiza
- las restricciones incluyen tamaño y conocimientos del equipo
- hay un equipo detrás de cada frontera que se quiera dura
- cada requisito ha pasado por las cinco preguntas
- los escenarios llevan fecha y quién los dio por bueno
- se sabe cuáles de las decisiones derivadas serán irreversibles

Y el cierre que enlaza con la clase siguiente: antes de decidir cómo se divide el sistema, conviene fijar cómo se comporta cada pieza para que pueda vivir en la nube —arrancar, configurarse, escalar y morir sin ceremonia—, que es la materia de la clase 146.

Ejemplo trabajado

CloudShop va a rediseñar su plataforma y empieza escribiendo los escenarios. El ejercicio dura dos sesiones y produce tres resultados: dos exigencias que se contradicen, un requisito que nadie pudo justificar y una restricción que nadie había puesto por escrito.

Los escenarios, tal como quedaron.

```
E1  RENDIMIENTO
    un cliente abre una ficha de producto durante la campaña de noviembre,
    con 5.000 peticiones/s
    → página completa en menos de 500 ms, percentil 99

E2  DISPONIBILIDAD
    el proveedor de pago deja de responder durante 30 minutos
    → el sistema sigue aceptando pedidos y los cobra después
    → ningún pedido perdido; el cliente ve un estado «en proceso»

E3  CONSISTENCIA
    dos clientes compran la última unidad con 50 ms de diferencia
    → uno la consigue y el otro recibe un mensaje claro
    → nunca se venden dos

E4  CAMBIO
    añadir un método de pago nuevo
    → sin tocar el servicio de pedidos, en menos de una semana,
      sin parada

E5  AISLAMIENTO
    dar de alta un cliente empresarial con datos separados
    → en menos de un día, sin que su carga afecte a los demás

E6  OPERACIÓN
    una dependencia se degrada
    → alguien se entera en menos de 2 minutos y el sistema sigue
      sirviendo lo que no depende de ella

E7  CRECIMIENTO
    el número de pedidos se multiplica por 4 en 18 meses
    → sin rediseño y con coste por pedido que no suba

E8  NORMATIVO
    un cliente europeo exige que sus datos no salgan del espacio europeo
    → incluidos registros, telemetría y copias           clase 141
```

El conflicto: E2 contra E3.

```
E2 pide seguir aceptando pedidos con el pago caído
E3 pide no vender nunca dos veces la última unidad

y si el pago está caído y se acepta el pedido:
  o se reserva la unidad sin cobrar         → se puede quedar reservada
```

```
  o no se reserva                          → se puede vender dos veces
```

No hay solución que cumpla los dos al 100 %. La discusión duró cuarenta minutos y terminó con una decisión explícita:

```
prioridad   E3 gana sobre E2
concreción  con el pago caído se acepta el pedido y SE RESERVA
            la unidad, con caducidad de 2 horas
            si el pago no se completa en 2 horas, se libera y se avisa
consecuencia aceptada
            unidades reservadas y no vendidas durante una caída larga
            → medido después: 41 unidades en 6 meses
```

Y esa decisión fija, sin haber hablado todavía de tecnología, que el inventario es dueño de la reserva y que el pago es asíncrono, que es lo que la clase 147 formalizará.

El orden de atributos.

La primera propuesta tenía nueve atributos «críticos». Al forzar el orden:

```
1. CORRECCIÓN DE LOS DATOS   no vender dos veces, no cobrar dos veces,
                             no perder pedidos
2. DISPONIBILIDAD del flujo de compra
3. EVOLUCIÓN                 poder añadir integraciones sin tocar el núcleo

lo que cede cuando hay conflicto
  latencia (hasta el límite de E1)
  coste (hasta el límite del presupuesto)
  aislamiento por inquilino más allá de lo que exige E5
```

Y lo que no se optimiza, escrito:

```
no se optimiza para más de 30.000 clientes simultáneos
no se optimiza para despliegues sin ninguna parada en migraciones
  de esquema mayores
no se optimiza para multi-proveedor activo-activo
no se optimiza para latencia por debajo de 200 ms fuera de Europa
```

La última evitó, meses después, una discusión sobre una red de distribución global que nadie necesitaba.

El requisito que nadie pudo justificar.

```
requisito propuesto   «disponibilidad del 99,99 %»

pregunta 1  ¿qué pasa si no se cumple?
            → «perderíamos ventas»
            → ¿cuántas? Nadie lo había calculado

cálculo hecho en la sesión
  99,9 %  → 43 min al mes de indisponibilidad
  99,99 % → 4 min al mes
  diferencia                                        39 min al mes
  ventas en 39 minutos, a la media                  ~2.100 €

pregunta 3  ¿a qué coste?
  estimación del salto: redundancia entre regiones,
  bases activas en dos sitios, ensayos              ~14.000 €/mes
```

Dos mil cien euros de pérdida evitada por catorce mil de coste. El requisito se cambió a 99,9 %, y la conversación duró veinte minutos porque había números.

Y el matiz que se añadió y que sí importaba:

```
lo que de verdad preocupaba no era el tiempo total,
sino perder pedidos durante la caída
→ y eso lo resuelve E2, que es mucho más barato
→ el requisito estaba mal expresado, no era exagerado
```

La restricción que nadie había escrito.

```
personas en el equipo de producto                          9
servicios que se proponía tener                           22
guardia                                          rotatoria, 4 personas

cuentas hechas en la sesión
  22 servicios × canalización, vigilancia, procedimientos,
  objetivos y guardia
```

```
  → 2,4 servicios por persona, contando solo el mantenimiento
```

Y la conclusión, que cambió el diseño antes de empezarlo:

```
restricción escrita   «el equipo puede operar entre 6 y 8 servicios»
consecuencia          la propuesta de 22 se descartó en esta sesión,
                      no seis meses después
```

Y se anotó la regla del apartado tercero: cada frontera dura necesita un equipo detrás; con dos equipos, dos o tres fronteras.

Lo que estos escenarios decidieron después.

```
E3 + prioridad 1   el inventario es dueño de la reserva      clase 147
E2                 el pago es asíncrono, con compensación    clase 116
E1                 caché del catálogo y pocos saltos         clases 111, 148
E4                 contrato estable de pagos                 clase 153
E5                 aislamiento por esquema, no por servicio  clase 154
E7                 clave de partición pensada para 4×        clases 110, 150
E8                 región europea y depuración de telemetría clase 141
restricción de equipo   6-8 servicios, no 22                 clase 148
```

Ocho escenarios y una restricción decidieron ocho cosas antes de escribir una línea de código.

A los dieciocho meses: la revisión.

```
escenarios que seguían siendo válidos                        6 de 8
E7 (×4 en 18 meses)     se cumplió: ×3,7. Sin rediseño.      ✓
E5                      cambió: hay 3 clientes empresariales
                        que exigen datos en su propia región
                        → escenario reescrito
E1                      cambió: la campaña de noviembre dejó
                        de celebrarse (clase 142)
                        → el pico de referencia bajó a 2.100/s
                        → y eso hizo revisable el compromiso

decisiones revisadas por el cambio de escenarios               2
```

Y el segundo caso es el que enseña la lección del apartado cuarto: una arquitectura correcta para los escenarios de hace dos años puede ser cara hoy sin que nada haya fallado, y solo se detecta si los escenarios llevan fecha y se revisan.

El resultado del ejercicio.

```
exigencias escritas como adjetivos             9             0
escenarios con medida                          0             8
atributos «críticos»                           9             3
lo que no se optimiza, escrito                 no            4 puntos
restricciones escritas                         2             6
requisitos retirados tras interrogarlos        —             1
conflictos entre atributos resueltos
explícitamente                                 0             1
decisiones de arquitectura derivadas           —             8
escenarios con fecha y responsable             no            sí
```

La lección que esta clase abre para la parte 12: la sesión decidió ocho cosas de arquitectura sin nombrar ni una tecnología, y las decidió porque había números. El requisito del 99,99 % se cayó al preguntar qué costaba y qué evitaba; la propuesta de veintidós servicios se cayó al escribir cuánta gente había; y la única discusión de verdad —qué hacer cuando dos exigencias se contradicen— terminó con una decisión explícita y una consecuencia aceptada por escrito, que es lo único que permite comprobar dieciocho meses después si seguía siendo la correcta.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-12-cloud-native-distributed-architecture/145-requisitos-restricciones-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 quality-attribute-scenarios 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 quality-attribute-scenarios 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
Las exigencias no permiten decidir nada	Están escritas como adjetivos: escalable, rápido, seguro, fiable	Escríbelas como escenarios con origen, estímulo, entorno, respuesta y medida.
Todos los atributos son críticos y cualquier decisión se discute eternamente	No se ha ordenado nada; se pretende maximizar todo	Fija tres atributos que ganan cuando hay conflicto y escribe explícitamente lo que no se optimiza.
Se diseña para una disponibilidad que cuesta más de lo que evita	El número se fijó por aspiración, sin calcular consecuencia ni coste	Interroga el requisito: qué pasa si no se cumple, cuándo hace falta, a qué coste y qué se está dispuesto a empeorar.
Se proponen veinte servicios para un equipo de nueve personas	El tamaño y los conocimientos del equipo no se trataron como restricción	Escríbelos como restricción y pon un equipo detrás de cada frontera que quieras dura.
Se paga complejidad por una escala que nunca llega	Se diseñó para un escenario sin fecha ni probabilidad	Diseña para el escenario conocido y decide con más cuidado solo lo que después será irreversible.
La arquitectura era correcta y hoy es cara sin que nada haya fallado	Los escenarios cambiaron y nadie los revisó	Ponles fecha y responsable, y revísalos con la misma cadencia que las decisiones.

■ 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 dos sistemas con las mismas funciones pueden no parecerse en nada?

2. ¿Qué cinco partes tiene un escenario y qué lo hace comprobable?
3. ¿Por qué los atributos se ordenan en vez de maximizarse, y cuántos deben ganar?
4. ¿Por qué el tamaño del equipo es una restricción de arquitectura?
5. ¿Qué cinco preguntas revelan si un requisito es real?

Referencias

- Bass, L., Clements, P. y Kazman, R. (2021). Software Architecture in Practice, caps. 3-4 — atributos de calidad y escenarios. <https://www.oreilly.com/library/view/software-architecture-in/9780136886051/>
- SEI (2025). Quality attribute workshop and utility tree — cómo obtener y priorizar escenarios. <https://insights.sei.cmu.edu/library/quality-attribute-workshops-qaws-third-edition/>
- Ford, N., Parsons, R. y Kua, P. (2017). Building Evolutionary Architectures — atributos como funciones de aptitud comprobables. <https://evolutionaryarchitecture.com/>
- Conway, M. (1968). How do committees invent? — relación entre comunicación de equipos y forma del sistema. <https://www.melconway.com/Home/CommitteesPaper.html>
- ISO/IEC (2023). 25010: modelo de calidad del producto software — taxonomía de atributos. <https://iso25000.com/index.php/normas-iso-25000/iso-25010>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 12 en PDF · Manual integral

Anterior	Índice	Siguiente
← 144 · Proyecto: landing zone con guardrails	Parte 12 · Programa	146 · Twelve-Factor App y configuración cloud-native →

Evaluación — 145 Requisitos, restricciones 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 quality-attribute-scenarios 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.

146 — Twelve-Factor App y configuración cloud-native

Parte: 12 — Arquitectura cloud-native y sistemas distribuidos Nivel: avanzado-experto · Horas estimadas: 4
Laboratorio: architecture · Estado: EXECUTABLECORE

Propósito

Fijar cómo debe comportarse cada pieza para que una plataforma que no sabe qué es pueda operarla: arrancar, configurarse, escalar, ser reemplazada y morir sin ceremonia. La clase toma la lista clásica de doce propiedades, la juzga por ese criterio, señala cuáles han envejecido mal o son directamente incorrectas hoy, y añade las siete que faltan porque se escribió antes de casi todo lo que este programa ha construido. Y termina con el mito más caro de esta materia: que un servicio pueda ser realmente sin estado.

Resultados de aprendizaje

Al finalizar podrás:

1. Juzgar cada propiedad por si permite que la plataforma opere la aplicación.
2. Corregir las tres que hoy se aplican mal, empezando por la configuración.
3. Añadir las propiedades que la lista original no contempla.
4. Separar qué configuración va en la imagen, en el despliegue y en ejecución.
5. Encontrar el estado local escondido, que siempre existe.

Conceptos centrales

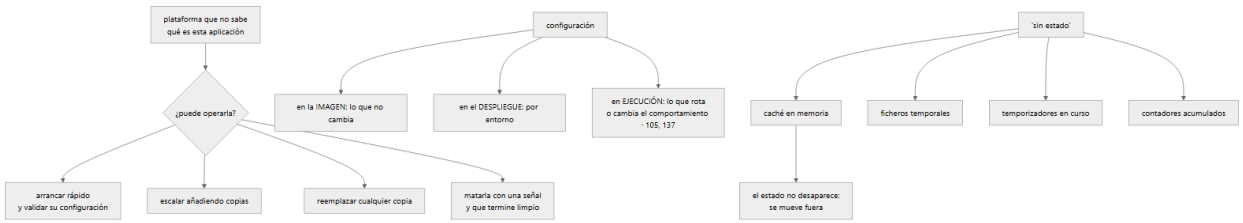
Concepto	Comprensión verificable
operable por la plataforma	Criterio único de esta clase: que algo que no conoce la aplicación pueda arrancarla, escalarla, reemplazarla y matarla sin dañarla.
configuración externalizada	Lo que cambia entre entornos vive fuera del artefacto. Externalizar es correcto; el mecanismo depende de si hay que rotarlo.
desechabilidad	Arrancar rápido y terminar limpiamente ante una señal. Es lo que permite escalar, desplegar y sobrevivir a la retirada de capacidad.
validación al arrancar	Comprobar la configuración completa al iniciar y fallar de inmediato con un mensaje claro, en vez de fallar tres horas después.
estado local escondido	Lo que la aplicación guarda en su proceso sin darse cuenta: cachés, ficheros temporales, temporizadores, contadores.
recurso adjunto	Toda dependencia externa se trata igual y se declara desde fuera, incluida su credencial, que además debe ser temporal.

Modelo mental

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

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 criterio, y lo que sigue siendo cierto

La lista clásica de doce propiedades se escribió para una plataforma concreta y se cita como dogma. Conviene juzgarla por lo que de verdad persigue:

```
que algo que NO sabe qué es esta aplicación pueda
  arrancarla, configurarla, escalarla, reemplazarla y matarla
  sin conocer sus detalles
```

Y con ese criterio, lo que sigue siendo cierto y central:

```
UN CÓDIGO, MUCHOS DESPLIEGUES
  el mismo artefacto en todos los entornos            clase 099

DEPENDENCIAS EXPLÍCITAS
  declaradas y fijadas con huella; nada implícito del sistema
                                                      clase 138

CONSTRUIR, PUBLICAR Y EJECUTAR SEPARADOS
  y publicar no vuelve a construir                    clase 099

PROCESOS SIN ESTADO LOCAL
  cualquier copia puede atender cualquier petición

RECURSOS ADJUNTOS
  toda dependencia externa se declara desde fuera y se puede
  sustituir sin tocar el código

ESCALAR AÑADIENDO COPIAS
  no haciendo la copia más grande

DESECHABILIDAD
  arranque rápido y terminación limpia ante una señal  clase 069

PARIDAD ENTRE ENTORNOS
  mismos servicios de apoyo, mismas versiones          clase 104

REGISTRO COMO FLUJO
  la aplicación no gestiona ficheros ni rotación       clase 122
```

Y de todas, la que más consecuencias tiene y menos se cumple es la desechabilidad, porque de ella dependen tres cosas a la vez:

```
escalar deprisa cuando llega carga                    clase 129
desplegar sin cortes                                  clase 102
sobrevivir a la retirada de capacidad interrumpible   clase 143
```

Y sus dos mitades se miden por separado:

```
ARRANQUE   desde que el proceso empieza hasta que sirve
  por debajo de 10 s        cómodo
  30 s a 2 min              el autoescalado llega tarde
  más de 2 min              no se puede escalar por demanda

TERMINACIÓN LIMPIA  al recibir la señal
  dejar de aceptar trabajo nuevo
  terminar lo que está en curso, con un plazo
  confirmar lo que corresponda                        clase 113
  cerrar conexiones y salir
```

Y el fallo típico de la segunda mitad: ignorar la señal y esperar a que la maten, lo que corta peticiones en curso y deja mensajes sin confirmar.

2. Las tres que hoy se aplican mal

1. «La configuración va en variables de entorno».

La idea correcta es externalizar; el mecanismo ha envejecido mal:

```
correcto     lo que cambia entre entornos no está dentro del artefacto
incorrecto   que CUALQUIER valor viaje en variables de entorno

porque una variable de entorno:
  no se puede cambiar sin reiniciar                    clase 137
  la heredan los procesos hijos
  aparece en volcados y en informes de error
```

Y de ahí la corrección de esta clase:

```
variables de entorno   valores no sensibles que no cambian en caliente
fichero montado        lo que hay que rotar, releído al cambiar
servicio de secretos   credenciales, con identidad de carga  clase 137
interruptor            lo que cambia el comportamiento       clase 105
```

Y una regla que evita sorpresas: si un valor cambia el comportamiento del sistema de forma apreciable, no es configuración: es un interruptor, y necesita registro de quién lo cambió, activación gradual y reversión.

2. «Los registros van a la salida estándar».

Cierto, y hoy insuficiente:

```
correcto     la aplicación no gestiona ficheros ni rotación
incompleto   emitir líneas de texto

lo que hace falta hoy
  eventos estructurados con mensaje constante          clase 122
  una línea ancha por unidad de trabajo
  y además métricas y trazas, que no son registro      clases 123, 124
```

3. «Los procesos administrativos se ejecutan como tareas puntuales».

Esta es la que peor ha envejecido:

```
entonces   entrar en una máquina y ejecutar un guion
hoy        un trabajo con la MISMA imagen y la misma configuración,
           lanzado por la plataforma, con su registro y su resultado
                                                        clase 079
```

Y sobre todo, la corrección importante:

```
las migraciones de esquema NO son tareas puntuales
son una secuencia de despliegues compatibles          clase 102
→ expandir, migrar, contraer
→ y el paso de contraer se planifica con fecha
```

Y una cuarta, más leve: «la concurrencia se consigue con el modelo de procesos». Hoy la unidad de escalado la decide la plataforma, y lo que queda de esa propiedad es lo que de verdad importaba: escalar horizontalmente y no guardar estado local.

3. Lo que falta en la lista

La lista original es anterior a casi todo lo que este programa ha construido. Lo que hay que añadirle:

```
1. SEÑALES DE ESTADO SEPARADAS                        clase 079
   viva     ¿hay que reiniciarme?
   lista    ¿puedo recibir tráfico?
   arranque ¿he terminado de iniciarme?
   → confundirlas provoca reinicios en cadena bajo carga

2. OBSERVABILIDAD COMO SALIDA DE PRIMERA CLASE        parte 10
   métricas con etiquetas acotadas, trazas con contexto propagado
   y el identificador de correlación en todo

3. IDENTIDAD OBTENIDA DEL ENTORNO                     clase 137
   la aplicación no guarda credenciales: las pide y las renueva
```

```
4. LÍMITES DECLARADOS Y RESPETADOS                    clase 078
   la aplicación declara cuánta memoria y procesador necesita
   y se comporta bien cuando se le acota
   → un tiempo de ejecución que no ve su límite se pasa y muere

5. IDEMPOTENCIA DE CUANTO LE LLEGA DE FUERA           clase 116
   la plataforma reintenta sola: mensajes, invocaciones, tareas

6. PLAZOS Y DEGRADACIÓN                               clase 130
   toda llamada saliente con plazo, y una respuesta razonable
   cuando la dependencia falla

7. COMPATIBILIDAD ENTRE VERSIONES                     clases 102, 115
   la versión nueva convive con la anterior: esquema, cola y contrato
```

Y la sexta y la séptima son las que convierten una aplicación correcta en una operable: sin plazos, una dependencia lenta la tumba; sin compatibilidad, no se puede desplegar de forma escalonada.

Y una propiedad transversal que merece nombre propio: fallar pronto y con un mensaje útil.

```
al arrancar, validar TODA la configuración
  ¿están todas las variables necesarias?
  ¿tienen valores del tipo y rango esperados?
  ¿se puede resolver y alcanzar cada dependencia declarada?

si falta algo → salir de inmediato con un mensaje que diga QUÉ falta
```

Y el motivo, con un ejemplo que ocurre siempre:

```
sin validación   la aplicación arranca, sirve durante tres horas
                 y falla al llegar la primera petición que usa
                 esa configuración
                 → y el error es un fallo de referencia nula,
                   no «falta la variable X»

con validación   no arranca, el despliegue no avanza y el mensaje
                 dice exactamente qué falta
```

Y su complemento: la configuración se valida también en la canalización, comparando la que se va a desplegar con el esquema declarado.

4. El estado que siempre está

«Sin estado» es una simplificación útil y falsa. El estado no desaparece: se mueve fuera del proceso. Y lo que queda dentro sin darse cuenta es la causa de una familia entera de errores raros.

La lista de estado local escondido, con lo que produce cada uno:

```
CACHÉ EN MEMORIA
  cada copia tiene el suyo → 40 copias, 40 versiones del dato
  → invalidar exige avisar a las 40                    clase 111
  → y si la clave no lleva el usuario, hay fuga entre usuarios

FICHEROS TEMPORALES
  se comparten entre peticiones de la misma copia      clase 117
  → nombres únicos y limpieza

SESIÓN PEGADA AL SERVIDOR
  obliga a mantener esa copia viva
  → impide desplegar y escalar con libertad

TEMPORIZADORES Y TAREAS DE FONDO
  se pierden al terminar el proceso
  → y con varias copias, se ejecutan N veces
  → esto es un trabajo programado, no una tarea dentro del servicio

CONTADORES Y ACUMULADORES
  «llevamos 412 pedidos hoy» dentro del proceso
  → cada copia cuenta lo suyo y ninguna cuenta bien

CONEXIONES Y AGRUPADORES
  son estado, y multiplican por el número de copias    clase 109

ESTADO DE INICIALIZACIÓN PEREZOSA
```

```
  la primera petición de cada copia es lenta
  → y con autoescalado, eso ocurre continuamente
```

Y las dos preguntas que lo detectan:

```
¿qué se pierde si mato este proceso ahora mismo?
¿qué pasa si hay veinte copias en vez de una?
```

La segunda encuentra los temporizadores y los contadores; la primera, las cachés y los ficheros.

Y sobre los servicios que sí tienen estado por naturaleza —bases de datos, colas, motores durables—, la conclusión honesta:

```
estas propiedades NO se les aplican igual
  no son desechables: retirarlos tiene consecuencias
  no escalan añadiendo copias sin más
  y su terminación limpia es mucho más delicada
→ por eso se usan como servicios gestionados siempre que se pueda
  y por eso la parte 09 existe
```

Y la lista de comprobación de la clase:

- el mismo artefacto va a todos los entornos
- las dependencias están declaradas y fijadas con huella
- arranque medido, por debajo del umbral que el escalado necesita
- la terminación atiende la señal: deja de aceptar, termina y confirma
- la configuración está externalizada, con el mecanismo adecuado a si hay que rotarla
- lo que cambia el comportamiento es un interruptor, no una variable
- la configuración se valida al arrancar y el proceso falla si falta algo
- hay señales separadas de vivo, listo y arrancando
- se emiten métricas y trazas, no solo registros
- la aplicación obtiene su identidad del entorno
- declara sus límites y se comporta bien cuando se le acotan
- todo lo que llega de fuera es idempotente
- toda llamada saliente tiene plazo y hay respuesta degradada
- la versión nueva convive con la anterior
- se ha buscado el estado local escondido con las dos preguntas

Y el cierre que enlaza con la clase siguiente: con las piezas preparadas para vivir en la plataforma, queda la decisión que determina todo lo demás: dónde se ponen las fronteras. Y esa no la decide la tecnología, sino quién es dueño de qué datos, que es la materia de la clase 147.

Ejemplo trabajado

CloudShop audita sus quince servicios contra la lista corregida. La auditoría cabe en una tabla y produce tres hallazgos que ya habían causado incidentes sin que nadie los relacionara.

La tabla.

```
mismo artefacto en todos los entornos              15 de 15
dependencias fijadas con huella                    15 de 15   (clase 138)
arranque por debajo de 30 s                         9 de 15
terminación limpia ante la señal                    6 de 15
configuración externalizada                        15 de 15
mecanismo adecuado para lo que rota                 2 de 15   (clase 137)
validación de configuración al arrancar             3 de 15
señales de vivo y listo separadas                   7 de 15
métricas y trazas además de registros              15 de 15   (parte 10)
identidad obtenida del entorno                     15 de 15   (clase 137)
límites declarados                                 15 de 15
se comporta bien al ser acotado                    11 de 15
idempotencia de lo que llega de fuera              13 de 15   (clase 116)
plazos en llamadas salientes                       15 de 15   (clase 130)
compatibilidad con la versión anterior             15 de 15   (clase 102)
sin estado local escondido                          4 de 15
```

Las cuatro filas peores son las que producen los hallazgos.

Hallazgo 1: nueve servicios no terminaban limpiamente.

```
comportamiento observado al recibir la señal
  6 servicios    terminan correctamente
  7 servicios    la ignoran y esperan a que los maten (30 s después)
```

```
  2 servicios    cierran de inmediato, cortando peticiones en curso
```

Y el efecto, que llevaba meses atribuido a otra cosa:

```
peticiones cortadas por despliegue, medido              ~180 por despliegue
despliegues al día                                            40
peticiones cortadas al día                                ~7.200
proporción del total                                       0,06 %
se achacaba a                              «errores intermitentes de red»
```

```
termina limpiamente                          6 de 15        15 de 15
peticiones cortadas por despliegue              ~180              0
mensajes sin confirmar por terminación          ~40               0
```

Y la secuencia que se implantó en los quince, idéntica:

```
1. marcar «no listo» → el reparto deja de enviar tráfico
2. esperar a que el reparto se entere (unos segundos)
3. dejar de aceptar peticiones y de leer de las colas
4. terminar lo en curso, con plazo de 25 s
5. confirmar lo que corresponda
6. cerrar conexiones y salir
```

El paso 2 es el que casi nadie pone y sin él los pasos siguientes no sirven de nada: el reparto sigue enviando tráfico durante unos segundos después de que el proceso se marque como no listo.

Hallazgo 2: la configuración que falló tres horas después.

```
03:10  despliegue de una versión con una variable nueva mal escrita
03:10  el servicio arranca correctamente
03:10  las comprobaciones de salud pasan
06:40  llega la primera petición que usa esa ruta
06:40  fallo de referencia nula; 100 % de esa ruta cae
07:15  diagnosticado y corregido
```

Tres horas y media entre desplegar y fallar, y el error no decía nada útil.

```
validación al arrancar                        3 de 15        15 de 15
qué se valida                                     —      presencia, tipo,
                                                         rango y alcance
                                                         de cada dependencia
validación también en la canalización            no             sí
incidentes por configuración en 6 meses           4              0
despliegues bloqueados por configuración
inválida                                          —             11
```

Once despliegues bloqueados en seis meses antes de llegar a producción.

Hallazgo 3: el estado local escondido.

Las dos preguntas del apartado cuarto se aplicaron a los quince servicios:

```
¿qué se pierde si lo mato ahora?
  cachés en memoria sin invalidación coordinada              9 servicios
  ficheros temporales sin limpiar                            4
  trabajo en curso no confirmado                             7 (hallazgo 1)

¿qué pasa con veinte copias en vez de una?
  temporizadores dentro del servicio                         3
    → un envío de recordatorios se ejecutaba una vez por copia
    → con 12 copias, 12 correos por cliente
  contadores acumulados en memoria                           2
    → un panel de «pedidos de hoy» mostraba 1/12 del valor real
  inicialización perezosa cara                               5
    → la primera petición de cada copia tardaba 3,2 s
    → con autoescalado, ocurría constantemente
```

Y el de los recordatorios llevaba cinco meses enviando correos duplicados:

```
correos duplicados enviados, estimados                    ~31.000
reclamaciones recibidas                                        14
cómo se explicaba antes                      «un problema del proveedor»
corrección   el envío pasó a ser un trabajo programado, con bloqueo
             y con idempotencia                          clases 079, 116
```

Y las inicializaciones perezosas se movieron al arranque, con una consecuencia buscada:

```
tiempo de arranque                          4 s             9 s
```

```
primera petición de una copia nueva      3,2 s            41 ms
coste                          arranque más lento; escalado igual de rápido
                               porque el umbral eran 30 s
```

El arranque, medido y corregido.

```
servicios por encima de 30 s                                  6
causas
  consultar configuración remota al arrancar                  4
  cargar un modelo o un catálogo entero en memoria            3
  esperar a que una dependencia estuviera lista               2
  ejecutar migraciones al arrancar                            1   ← grave
```

El último merecía su propio incidente:

```
el servicio ejecutaba las migraciones de esquema al arrancar
con 12 copias arrancando a la vez tras un despliegue
→ 12 procesos intentaban migrar
→ bloqueos y arranques de 4 minutos
corrección   migración como trabajo previo al despliegue,
             con expandir y contraer                       clase 102
```

```
arranque, mediana                            38 s            11 s
arranque, peor caso                         4 min            19 s
servicios que escalan a tiempo              9 de 15        15 de 15
```

La configuración, reorganizada en tres capas.

```
EN LA IMAGEN            lo que no cambia entre entornos
                        rutas internas, valores por defecto, esquema
EN EL DESPLIEGUE        lo que cambia por entorno
                        puntos de acceso, tamaños, regiones
EN EJECUCIÓN            lo que rota o cambia el comportamiento
                        credenciales (fichero releído) e interruptores
```

```
valores que estaban en variables de entorno y cambiaron de sitio
  credenciales                                      18  → almacén
  valores que cambiaban el comportamiento            7  → interruptores
  el resto                                          61  → siguen igual
```

Y los siete que pasaron a interruptores tenían todos la misma característica: alguien los cambiaba de vez en cuando y nadie sabía quién ni cuándo.

A los cuatro meses.

```
terminación limpia                            6 de 15       15 de 15
peticiones cortadas por despliegue              ~180             0
validación al arrancar                        3 de 15       15 de 15
incidentes por configuración                 4 / 6 meses        0
arranque, peor caso                          4 min           19 s
estado local escondido, servicios afectados   11 de 15       0 de 15
correos duplicados                           ~31.000           0
inicialización perezosa en el camino crítico  5 de 15        0 de 15
valores sensibles en variables de entorno       18              0
valores que cambian comportamiento
como variable                                    7              0
```

La lección que esta clase traslada a la parte 12: la auditoría no encontró ningún problema nuevo. Encontró la explicación de tres cosas que ya estaban ocurriendo y que se atribuían a otra causa: siete mil doscientas peticiones cortadas al día que se llamaban «errores de red», treinta y un mil correos duplicados que se llamaban «un problema del proveedor», y un panel con la doceava parte de los pedidos que nadie había cuestionado. Las tres eran estado local escondido o terminación sucia, y las tres estaban en la lista desde 2011.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-12-cloud-native-distributed-architecture/146-twelve-factor-app-y-configuracion-cloud-native/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-twelve-factor 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-twelve-factor 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
Cada despliegue corta peticiones que se achacan a errores de red	El proceso ignora la señal de terminación, o cierra sin esperar a que el reparto deje de enviarle tráfico	Márcate no listo, espera a que el reparto se entere, deja de aceptar, termina lo en curso con plazo y confirma antes de salir.
Un despliegue arranca bien y falla horas después por configuración	No se valida la configuración al arrancar	Valida presencia, tipo, rango y alcance de las dependencias al iniciar, falla de inmediato con un mensaje claro y valida también en la canalización.
Una tarea periódica se ejecuta tantas veces como copias hay	El temporizador vive dentro del servicio	Conviértelo en un trabajo programado con bloqueo e idempotencia, fuera del proceso que sirve peticiones.
Un contador o panel muestra una fracción del valor real	Se acumula en memoria y cada copia cuenta lo suyo	Saca el estado del proceso; cuenta con métricas agregadas o en un almacén compartido.
El autoescalado llega tarde y las copias nuevas responden lentas al principio	Arranque largo e inicialización perezosa en el camino crítico	Mueve la inicialización al arranque, quita consultas remotas y saca las migraciones del arranque.
Un valor de configuración cambia el comportamiento y nadie sabe quién lo tocó	Se trató como configuración lo que es un interruptor	Si cambia el comportamiento de forma apreciable, hazlo interruptor: con registro, gradualidad y reversión.

■ 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 el criterio único para juzgar estas propiedades?
2. ¿Qué tiene de correcto y de incorrecto «la configuración va en variables de entorno»?
3. ¿Qué siete propiedades faltan en la lista original y por qué?
4. ¿Qué dos preguntas encuentran el estado local escondido?
5. ¿Por qué las migraciones de esquema no son una tarea puntual?

Referencias

• Wiggins, A. (2011). The Twelve-Factor App — la lista original y su contexto. <https://12factor.net/>

• Hoffman, K. (2016). Beyond the Twelve-Factor App — propiedades añadidas: telemetría, autenticación y seguridad. <https://www.oreilly.com/library/view/beyond-the-twelve-factor/9781492042631/>

• Kubernetes (2025). Pod lifecycle: termination and probes — señal de terminación, plazo y señales de estado. <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/>

• Google Cloud (2025). Best practices for building containers — arranque, configuración y desechabilidad. <https://cloud.google.com/architecture/best-practices-for-building-containers>

• OpenTelemetry (2025). Instrumentation as a first-class output — telemetría como salida de la aplicación. <https://opentelemetry.io/docs/concepts/instrumentation/>

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

Descargar: Parte 12 en PDF · Manual integral

Anterior	Índice	Siguiente
← 145 · Requisitos, restricciones y atributos de calidad	Parte 12 · Programa	147 · DDD, bounded contexts y ownership de datos →

Evaluación — 146 Twelve-Factor App y configuración cloud-native

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-twelve-factor 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.

147 — DDD, bounded contexts y ownership de datos

Parte: 12 — Arquitectura cloud-native y sistemas distribuidos Nivel: avanzado-experto · Horas estimadas: 4
Laboratorio: architecture · Estado: EXECUTABLECORE

Propósito

Decidir dónde van las fronteras, que es la decisión de la que dependen todas las demás de esta parte. La clase defiende que no las decide la tecnología sino dos cosas concretas: dónde cambia el significado de las palabras y quién puede modificar cada dato. Da una técnica para encontrarlas en horas en vez de en meses, establece la regla que hace que todo lo demás funcione —un solo escritor por dato— y desarrolla la pieza que más problemas evita en sistemas reales: la capa que traduce el modelo ajeno en la frontera para que no se filtre hacia dentro.

Resultados de aprendizaje

Al finalizar podrás:

1. Encontrar fronteras buscando dónde una misma palabra significa cosas distintas.
2. Asignar un único escritor a cada dato y derivar de ahí la separación.
3. Elegir la relación entre contextos y aislar los modelos ajenos.
4. Delimitar la unidad de consistencia y saber qué queda fuera de ella.
5. Decidir dónde invertir esfuerzo y qué adoptar tal cual.

Conceptos centrales

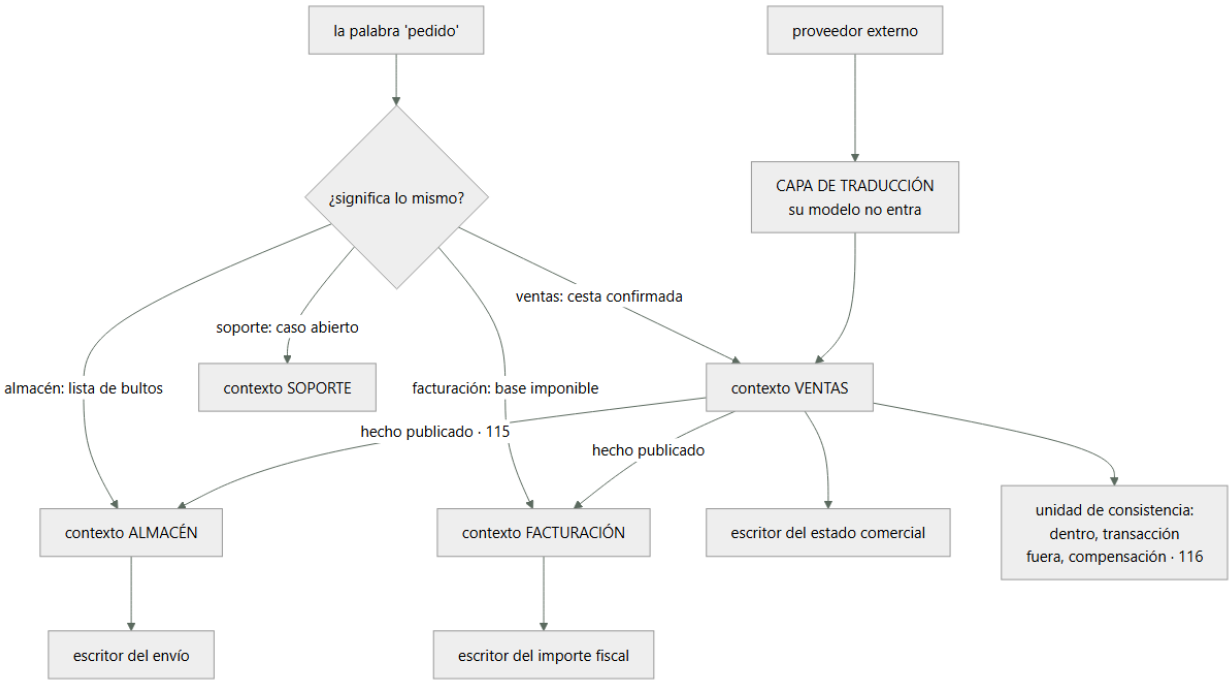
Concepto	Comprensión verificable
contexto acotado	Región donde cada palabra tiene un único significado y un modelo coherente. Fuera de ella, la misma palabra puede significar otra cosa.
un solo escritor	Cada dato tiene exactamente un contexto que puede modificarlo. Los demás tienen copia o preguntan.
base de datos de integración	Base compartida por varios contextos que escriben en las mismas tablas. Es lo que impide desplegar y evolucionar por separado.
capa de traducción	Código en la frontera que convierte el modelo ajeno al propio, para que el ajeno no se filtre hacia dentro.
unidad de consistencia	Conjunto de datos que cambia junto, en una transacción. Lo que la cruza es eventualmente consistente.
subdominio central	La parte que diferencia al negocio. Es donde se invierte; lo genérico se adopta tal cual.

Modelo mental

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

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 frontera está donde cambia el significado

La técnica más útil de esta materia cabe en una frase:

```
busca dónde la misma palabra significa cosas distintas
```

Y el ejemplo canónico de cualquier tienda:

```
«PEDIDO» para VENTAS
  una cesta confirmada, con precios, descuentos y método de pago
  se cierra cuando se cobra

«PEDIDO» para ALMACÉN
  una lista de artículos que hay que localizar y empaquetar
  puede partirse en tres envíos, y le da igual el precio

«PEDIDO» para FACTURACIÓN
  una base imponible con impuestos y un documento legal
  puede rectificarse años después

«PEDIDO» para SOPORTE
  un caso, con su historial de conversaciones
```

Cuatro cosas distintas con el mismo nombre. Y las consecuencias de no separarlas:

```
un modelo único intenta servir a las cuatro
→ acumula campos que solo usa uno
→ cada cambio afecta a los cuatro equipos
→ y las discusiones son sobre el significado, no sobre el código
```

Y dos señales más de que hay una frontera:

```
la misma palabra significa lo mismo pero con DETALLE distinto
  ventas necesita el cliente completo; almacén solo su dirección

los ciclos de vida no coinciden
  el pedido de ventas termina al cobrar
  el de facturación sigue vivo cuatro años
```

Y la técnica para encontrarlas en horas, sin un año de análisis:

```
1. escribir los HECHOS del negocio en orden temporal
   «cesta confirmada», «pago autorizado», «pedido preparado»,
   «envío entregado», «factura emitida», «devolución solicitada»

2. anotar quién reacciona a cada hecho y quién lo produce
```

```
3. agrupar por quién habla el mismo idioma

4. y marcar los puntos donde una palabra cambia de significado:
   ahí está la frontera
```

Y lo que no decide una frontera, aunque se use constantemente:

```
la tecnología      «el servicio de base de datos», «el servicio de API»
la capa            separar por capas produce fronteras que hay que
                   cruzar en cada operación
el organigrama     por accidente; aunque el organigrama SÍ importa
                   como restricción                        clase 145
el tamaño          «cada servicio no más de N líneas» no es un criterio
```

2. Un solo escritor por dato

Esta es la regla que convierte una idea de modelado en una decisión de arquitectura:

```
cada dato tiene EXACTAMENTE UN contexto que puede modificarlo
los demás tienen una copia, o preguntan
```

Y de ella se derivan casi todas las decisiones posteriores:

```
quién expone qué operación
qué se publica como hecho                              clase 115
qué se puede desplegar sin coordinar
y quién está de guardia de qué                         clase 095
```

El antipatrón correspondiente tiene nombre propio:

```
BASE DE DATOS DE INTEGRACIÓN
  varios contextos escriben en las mismas tablas
  → el esquema es un contrato implícito entre todos
  → nadie puede cambiarlo sin coordinar con todos
  → y una migración exige parar todo a la vez
```

Y conviene distinguirlo de algo parecido y aceptable:

```
compartir la INSTANCIA de base de datos     aceptable, es economía
compartir las TABLAS y escribirlas          esto es el antipatrón
```

Con esquemas separados y permisos separados, una sola instancia puede alojar varios contextos sin acoplarlos. Lo que acopla es quién escribe.

Y qué hacen los demás con un dato del que no son dueños:

```
COPIA LOCAL, actualizada por hechos                    clase 114
  + funciona con el dueño caído; rápido
  − eventualmente consistente, y hay que mantenerla

PREGUNTAR AL DUEÑO
  + siempre al día
  − acopla en tiempo de ejecución: si el dueño cae, tú caes
  − y añade latencia y saltos                          clase 145
```

Y el criterio, que es el de la clase 145: si el atributo prioritario es disponibilidad, copia; si es exactitud del momento, pregunta.

Y una tercera vía muy usada y honesta: copia local con caducidad y degradación, que sirve el valor copiado y avisa de su antigüedad.

La unidad de consistencia, que enlaza con la parte 09:

```
dentro de una unidad     una transacción; invariantes garantizadas
                         «las líneas de un pedido suman su total»
entre unidades           NO hay transacción
                         → compensaciones y estados intermedios  clase 116
```

Y el error habitual es hacerlas demasiado grandes:

```
«el cliente y todos sus pedidos» como una unidad
→ dos operaciones sobre pedidos distintos del mismo cliente compiten
→ y la unidad crece sin límite                          clase 110
```

La regla práctica: la unidad es lo mínimo que debe cambiar junto para que una regla de negocio se cumpla. Todo lo demás, fuera.

3. Relaciones entre contextos

Dos contextos que se hablan tienen una relación, y conviene nombrarla porque determina quién soporta el coste del cambio:

```
PROVEEDOR Y CLIENTE
  el proveedor publica un contrato y el cliente lo consume
  el proveedor tiene en cuenta las necesidades del cliente
  → es la relación normal, y necesita el contrato de la clase 153

CONFORMISTA
  el cliente acepta el modelo del proveedor tal cual
  → barato, y el modelo ajeno entra en tu casa

CAPA DE TRADUCCIÓN
  el cliente traduce el modelo ajeno al suyo en la frontera
  → cuesta más y protege el modelo propio

NÚCLEO COMPARTIDO
  dos contextos comparten modelo y código
  → acopla sus despliegues; usar solo con un equipo detrás de los dos

CAMINOS SEPARADOS
  no se integran; se duplica a propósito
  → a veces es la respuesta correcta
```

Y la capa de traducción es la que más problemas evita en sistemas reales:

```
el proveedor de pago llama «transaction» a algo que en tu modelo
es un intento de cobro
su estado «PENDING_CAPTURE» no significa nada en tu dominio
y su identificador tiene su formato

sin traducción
  esos nombres aparecen en tu base, en tus eventos y en tu interfaz
  → cambiar de proveedor obliga a tocar todo
  → y un cambio suyo se propaga hacia dentro

con traducción
  una capa convierte su modelo al tuyo, en un solo sitio
  → cambiar de proveedor toca esa capa
  → y lo de dentro no se entera
```

Y se aplica a más cosas de las que parece:

```
proveedores externos                       casi siempre
sistemas heredados que no se pueden tocar  siempre
otro contexto cuyo modelo no te convence   cuando puedas permitírtelo
```

Y la señal de que falta una: vocabulario ajeno dentro de tu código. Si en el servicio de pedidos aparecen palabras del proveedor de pagos, el modelo ya se filtró.

Y el lenguaje publicado, que es lo que la clase 115 llamó contrato de eventos: un vocabulario común y estable que varios contextos entienden, distinto del modelo interno de cualquiera de ellos.

4. Dónde invertir, y cómo se estropea

No todas las partes merecen el mismo esfuerzo:

```
CENTRAL       lo que diferencia al negocio
              → aquí se invierte: modelo propio, buen código,
                los mejores del equipo
              → CloudShop: el motor de precios y promociones

DE APOYO      necesario y no diferencia
              → se construye simple, sin sofisticación
              → gestión de pedidos estándar

GENÉRICO      resuelto por el mercado
              → se adopta tal cual, sin personalizar
              → identidad, pagos, correo, facturación fiscal
```

Y las dos reglas que se derivan:

```
no construir lo genérico
```

```
no dejar que lo genérico dicte el modelo de lo central
  → y para lo segundo hace falta la capa de traducción
```

Y el error clásico: personalizar hasta el infinito una herramienta genérica porque «casi encaja», hasta que mantenerla cuesta más que haberla construido.

Cómo se estropea una división, con las causas por frecuencia:

```
FRONTERAS POR TECNOLOGÍA
  «servicio de base de datos», «servicio de notificaciones»
  → cada operación de negocio cruza cuatro fronteras
  → y la latencia y los fallos parciales se multiplican

FRONTERAS POR CAPA
  interfaz, lógica y datos como servicios distintos
  → mismo problema, con un nombre más respetable

DATOS COMPARTIDOS ENTRE CONTEXTOS
  el antipatrón del apartado segundo

UN CONTEXTO SIN EQUIPO
  ley 20: acaba sin dueño                          clase 144

DEMASIADOS CONTEXTOS PARA EL EQUIPO
  restricción de la clase 145
```

Y dos comprobaciones que detectan una mala división antes de sufrirla:

```
1. ¿cuántas fronteras cruza una operación de negocio típica?
   una o dos      bien
   cuatro o más   la división está mal hecha

2. ¿cuántos equipos hay que coordinar para un cambio habitual?
   uno            bien
   tres           la frontera está en el sitio equivocado
```

La segunda es la definitiva: si un cambio frecuente exige coordinar a tres equipos, la frontera no está donde debería.

Y la lista de comprobación de la clase:

- [ ] están identificadas las palabras que significan cosas distintas
- [ ] cada contexto tiene un vocabulario propio, escrito
- [ ] cada dato tiene un único escritor, y está anotado quién
- [ ] ningún contexto escribe en las tablas de otro
- [ ] está decidido, por dato, si los demás copian o preguntan
- [ ] las unidades de consistencia son lo mínimo que debe cambiar junto
- [ ] lo que cruza unidades usa compensación, no transacción
- [ ] hay capa de traducción ante proveedores y sistemas heredados
- [ ] no aparece vocabulario ajeno dentro del modelo propio
- [ ] está clasificado qué es central, de apoyo y genérico
- [ ] cada contexto tiene un equipo detrás
- [ ] una operación típica cruza una o dos fronteras, no cuatro
- [ ] un cambio habitual no exige coordinar tres equipos

Y el cierre que enlaza con la clase siguiente: con las fronteras decididas por el significado y por la propiedad de los datos, queda una pregunta que suele hacerse primero y debería hacerse ahora: cuántas de esas fronteras se convierten en procesos separados, que es la materia de la clase 148.

Ejemplo trabajado

CloudShop tiene quince servicios divididos por criterios que nadie recuerda y una base de datos que once de ellos escriben. El ejercicio consiste en encontrar las fronteras reales y compararlas con las que hay.

Paso 1: los hechos del negocio, en orden.

Una sesión de dos horas con gente de ventas, almacén, facturación y soporte produjo treinta y un hechos. Los principales:

```
cesta confirmada → pago autorizado → pedido aceptado → artículos
reservados → pedido preparado → envío entregado → factura emitida
→ devolución solicitada → devolución recibida → abono emitido
```

Y al anotar quién produce y quién reacciona, apareció el agrupamiento:

```
ventas       cesta, precios, promociones, aceptación
almacén      reserva, preparación, envío
facturación  factura, abono, impuestos
soporte      casos, devoluciones desde el punto de vista del cliente
pagos        autorización, captura, devolución de dinero
```

Paso 2: las palabras que significaban cosas distintas.

```
PEDIDO       4 significados, como en el apartado primero
CLIENTE      ventas: quien compra, con su historial de compras
             facturación: un sujeto fiscal con NIF y dirección legal
             soporte: alguien con un correo y una conversación
DEVOLUCIÓN   soporte: una solicitud del cliente
             almacén: un paquete que va a llegar
             facturación: un abono con efectos fiscales
             pagos: una operación de devolución de dinero
PRECIO       ventas: lo que se muestra, con promociones
             facturación: base imponible sin impuestos
```

Cuatro palabras, catorce significados. Y en el modelo había una sola tabla para cada una.

Paso 3: comparar con la división existente.

```
servicios actuales                                          15
contextos identificados                                      5

servicios que no correspondían a ningún contexto
  «servicio de base de datos»              técnico           1
  «servicio de API»                        capa              1
  «servicio de notificaciones»             técnico           1
  «servicio de utilidades»                 cajón desastre    1
  «servicio de informes»                   técnico           1
```

Y la comprobación del apartado cuarto, medida sobre el flujo de compra:

```
fronteras que cruzaba «confirmar un pedido»                  7
equipos que había que coordinar para cambiar el cálculo
de un descuento                                              3
```

Siete fronteras y tres equipos para tocar un descuento. Las dos comprobaciones daban rojo.

Paso 4: los escritores.

```
tablas de la base compartida                                84
tablas con más de un escritor                               41
la peor: tabla «pedidos»                          escrita por 7 servicios
```

Y el efecto que llevaba dos años atribuido a la complejidad del negocio:

```
cambios de esquema en 12 meses                               9
de ellos, que exigieron coordinar 3 o más equipos            9
tiempo medio de un cambio de esquema                  5 semanas
incidentes por un cambio de esquema                          3
```

Y al asignar un escritor por dato:

```
dato                            escritor único
estado comercial del pedido     ventas
precio aplicado y promoción     ventas
reserva y estado de preparación almacén
dirección de envío              ventas escribe, almacén copia
importe fiscal y factura        facturación
estado del cobro                pagos
conversaciones y casos          soporte
```

Y dos decisiones de copiar frente a preguntar, tomadas con el criterio de la clase 145:

```
almacén necesita la dirección de envío
  atributo prioritario: disponibilidad (E2)
  → COPIA, actualizada por el hecho «pedido aceptado»
  → funciona con ventas caído

facturación necesita el importe final
  atributo prioritario: exactitud
  → COPIA TAMBIÉN, porque el importe se congela al emitir la factura
  → y una factura no cambia si ventas cambia el precio después
```

La segunda es interesante: facturación no quiere el dato actual, quiere el que había en ese momento, y eso es una copia por definición.

Paso 5: la capa de traducción para pagos.

```
palabras del proveedor encontradas dentro del código propio
  «transaction_id» en la tabla de pedidos                    sí
  estado «PENDING_CAPTURE» en el modelo de dominio           sí
  códigos de error del proveedor en la interfaz de usuario   sí
  el formato de importe del proveedor (céntimos como texto)  sí
servicios que conocían el vocabulario del proveedor          6
```

Y el coste medido cuando se integró el segundo proveedor:

```
servicios que hubo que tocar                    6               1
tiempo de la integración                  7 semanas         9 días
cambios en el modelo de dominio                 4               0
cambios en la interfaz de usuario               2               0
```

Y el modelo propio quedó así:

```
intento de cobro   { id propio, importe, moneda, estado propio,
                     referencia externa opaca }
estados propios    solicitado · autorizado · cobrado · rechazado ·
                   devuelto
```

Con cinco estados propios y una referencia opaca, los dos proveedores encajan y ninguno dicta nada.

Paso 6: dónde invertir.

```
CENTRAL     motor de precios y promociones
            → equipo dedicado, modelo propio, buena cobertura
DE APOYO    gestión de pedidos, preparación, casos
            → simple, sin sofisticación
GENÉRICO    identidad, pagos, correo, facturación fiscal, búsqueda
            → adoptado tal cual, con capa de traducción
```

Y una decisión que se revirtió con esto delante:

```
había un proyecto para construir un motor de facturación fiscal propio
motivo    «el producto de mercado no encaja al 100 %»
revisión  es genérico; no diferencia el negocio
decisión  se compró y se adaptó el proceso, no el producto
ahorro estimado                          5 meses de dos personas
```

El resultado.

```
servicios                                    15             8
contextos                                 no definidos      5
servicios que no eran de ningún contexto      5             0
tablas con más de un escritor                41             0
esquemas separados por contexto               no            sí
fronteras que cruza «confirmar pedido»         7             2
equipos a coordinar para cambiar un descuento  3             1
tiempo medio de un cambio de esquema       5 semanas         3 días
servicios que conocen el vocabulario de pagos  6             1
tiempo de integrar un proveedor nuevo      7 semanas         9 días
```

Y los ocho servicios encajan con la restricción de la clase 145: el equipo puede operar entre seis y ocho.

La lección que esta clase traslada a la parte 12: la división existente tenía quince servicios y cinco de ellos no correspondían a ninguna frontera del negocio: eran capas y tecnologías con nombre de servicio. Y el problema que más costaba —cinco semanas para un cambio de esquema y tres equipos para tocar un descuento— no venía de tener demasiados servicios, sino de que cuarenta y una tablas tenían más de un escritor. La frontera no estaba en el despliegue: estaba en los datos, y ahí no había ninguna.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-12-cloud-native-distributed-architecture/147-ddd-bounded-contexts-y-ownership-de-datos/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 context-map 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 context-map 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
Cada cambio de esquema exige coordinar a varios equipos	Varios contextos escriben en las mismas tablas: base de datos de integración	Asigna un único escritor a cada dato, separa esquemas y permisos, y que los demás copien o pregunten.
Discusiones interminables sobre qué campos lleva una entidad	Un solo modelo intenta servir a contextos donde la palabra significa cosas distintas	Busca dónde cambia el significado y separa; cada contexto tiene su propio modelo de la misma palabra.
Una operación de negocio cruza cuatro o más servicios	Las fronteras se trazaron por tecnología o por capa	Traza por contexto y propiedad de datos; mide cuántas fronteras cruza una operación típica.
Cambiar de proveedor obliga a tocar medio sistema	Su vocabulario y sus estados entraron en el modelo propio	Pon una capa de traducción en la frontera y guarda solo una referencia opaca.
Se construye una pieza que el mercado ya resuelve	No se clasificó qué es central, de apoyo y genérico	Invierte en lo central, simplifica lo de apoyo y adopta lo genérico sin personalizarlo.
Una unidad de consistencia crece sin límite y produce conflictos	Se agrupó de más: el cliente con todos sus pedidos	La unidad es lo mínimo que debe cambiar junto para que se cumpla una regla; lo demás va fuera y se coordina con compensación.

■ 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 señal indica que hay una frontera entre contextos?
2. ¿Qué diferencia hay entre compartir la instancia de base de datos y compartir las tablas?
3. ¿Cuándo conviene copiar un dato ajeno y cuándo preguntar por él?
4. ¿Qué protege una capa de traducción y cómo se detecta que falta?
5. ¿Qué dos comprobaciones revelan que una división está mal hecha?

Referencias

- Evans, E. (2003). Domain-Driven Design, parte IV — contextos acotados, mapas de contexto y capa de traducción. <https://www.domainlanguage.com/ddd/>
- Vernon, V. (2013). Implementing Domain-Driven Design, caps. 2-3 — contextos, relaciones y lenguaje. <https://www.informit.com/store/implementing-domain-driven-design-9780321834577>
- Brandolini, A. (2025). Event storming — encontrar fronteras a partir de los hechos del negocio. <https://www.eventstorming.com/>
- Fowler, M. (2025). Bounded context and integration database — el antipatrón de la base compartida. <https://martinfowler.com/bliki/BoundedContext.html>
- Skelton, M. y Pais, M. (2019). Team Topologies, cap. 6 — una frontera necesita un equipo detrás. <https://teamtopologies.com/book>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 12 en PDF · Manual integral

Anterior	Índice	Siguiente
← 146 · Twelve-Factor App y configuración cloud-native	Parte 12 · Programa	148 · Monolito modular, microservicios y función →

Evaluación — 147 DDD, bounded contexts y ownership de datos

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 context-map 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.

148 — Monolito modular, microservicios y función

Parte: 12 — Arquitectura cloud-native y sistemas distribuidos Nivel: avanzado-experto · Horas estimadas: 4
Laboratorio: decision · Estado: EXECUTABLECORE

Propósito

Decidir cuántas unidades desplegables independientes tiene el sistema, sabiendo que la pregunta difícil —dónde van las fronteras— ya la respondió la clase 147. La clase enumera con precisión lo que cuesta convertir una frontera interna en una de red —siete costes concretos por cada una—, distingue los motivos que justifican pagarlos de los que no, y defiende que el monolito modular bien hecho es la forma más barata de conservar la opción de dividir. Y termina con el peor resultado posible, que no es ninguno de los tres: servicios separados que hay que desplegar juntos.

Resultados de aprendizaje

Al finalizar podrás:

1. Enumerar lo que cuesta cada frontera que pasa de proceso a red.
2. Distinguir los motivos que justifican separar de los que no.
3. Construir un monolito modular que permita dividir después sin reescribir.
4. Extraer un servicio por el orden correcto, que no es el del código.
5. Reconocer un monolito repartido y saber qué lo produce.

Conceptos centrales

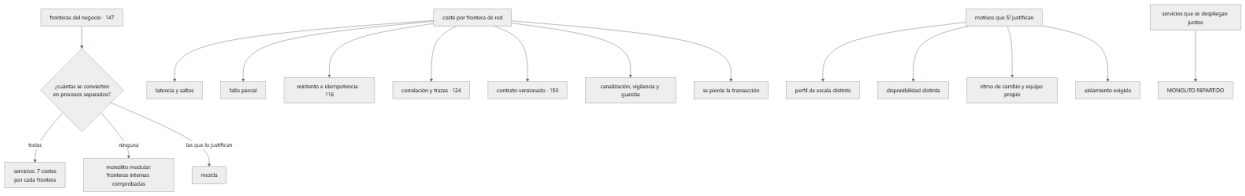
Concepto	Comprensión verificable
unidad desplegable	Lo que se construye, versiona y despliega por separado. Es la granularidad que decide esta clase.
monolito modular	Una unidad desplegable con fronteras internas explícitas y comprobadas. Conserva las llamadas en proceso y la opción de dividir.
coste de frontera	Lo que se paga al convertir una llamada en proceso en una llamada de red: siete cosas concretas.
monolito repartido	Servicios separados que hay que desplegar juntos. Paga todos los costes de dividir y no obtiene ninguno de los beneficios.
extracción progresiva	Sacar un contexto del monolito desviando tráfico poco a poco, sin reescribir todo de golpe.
prueba de arquitectura	Comprobación automática de que ningún módulo llama a lo que no debe. Es lo que impide que las fronteras internas se disuelvan.
perfil	Conjunto de exigencias de una parte: escala, disponibilidad, ritmo de cambio, tecnología, aislamiento. Perfiles distintos justifican separar.

Modelo mental

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

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 cuesta cada frontera de red

Convertir una llamada dentro del proceso en una llamada por red no es un detalle de despliegue: añade siete problemas que antes no existían.

```
1. LATENCIA Y SALTOS
   microsegundos pasan a milisegundos, y se acumulan

2. FALLO PARCIAL
   antes o funcionaba o no; ahora puede no responder, responder tarde
   o responder a medias                                    clase 151

3. REINTENTO E IDEMPOTENCIA
   toda operación que se pueda reintentar debe poder repetirse
   sin efecto adicional                                    clase 116

4. CORRELACIÓN
   para saber qué pasó hay que propagar contexto y tener trazas
                                                           clase 124

5. CONTRATO VERSIONADO
   las dos partes se despliegan por separado, así que tienen que
   convivir versiones                                      clases 102, 153

6. OPERACIÓN MULTIPLICADA
   canalización, objetivos, alertas, procedimientos y guardia
                                                           partes 08 y 10

7. SE PIERDE LA TRANSACCIÓN
   lo que antes era atómico pasa a necesitar compensación  clase 116
```

Y la conclusión que ordena la clase:

```
separar tiene un precio fijo y conocido
→ el beneficio de cada frontera tiene que superar esos siete
→ y si no se sabe cuál es el beneficio, no se separa
```

Y conviene subrayar el séptimo, porque es el que más se subestima: una transacción que abarcaba tres tablas del mismo esquema pasa a ser una secuencia con compensaciones, con estados intermedios visibles y casos que resolver a mano.

Y el sexto, que es el que agota equipos:

```
cada servicio necesita
  su canalización con puertas                              clase 100
  sus objetivos e indicadores                              clase 126
  sus alertas y procedimientos                             clases 125, 128
  su capacidad medida y su codo                            clase 129
  y alguien de guardia que lo entienda
```

Y de ahí la restricción de la clase 145: el número de servicios lo limita el número de personas, y no al revés.

2. Cuándo sí y cuándo no

Los motivos que sí justifican pagar los siete costes:

```
PERFIL DE ESCALA DISTINTO
  una parte necesita 40 copias y otra 2
  → juntas, se paga la de 40 para todo
  → es el motivo más frecuente y el más fácil de medir

DISPONIBILIDAD DISTINTA
```

```
  una parte tiene que seguir viva cuando la otra cae
  → y esto solo funciona si además se diseñan las dependencias
    como opcionales                                        clase 130

RITMO DE CAMBIO DISTINTO CON EQUIPO PROPIO
  una parte cambia a diario y otra dos veces al año
  → y hay dos equipos que quieren desplegar sin coordinarse
  → sin dos equipos, este motivo NO aplica

TECNOLOGÍA DISTINTA NECESARIA
  un modelo de aprendizaje, un motor de cálculo, una biblioteca
  que no existe en el lenguaje principal

AISLAMIENTO EXIGIDO
  normativa, residencia o un inquilino que exige separación
                                                  clases 141, 154

CICLO DE VIDA DE DATOS DISTINTO
  datos que se conservan cuatro años frente a datos efímeros
```

Y los que no lo justifican, aunque se usen constantemente:

```
«queda más limpio»                  la limpieza se consigue con módulos
«el código es muy grande»           el tamaño no es un criterio
«así cada uno elige su lenguaje»    multiplica el coste de operar
«es lo que hace todo el mundo»
«para poder reutilizarlo»           la reutilización se consigue con
                                    una biblioteca, sin red
«el equipo quiere aprenderlo»       motivo legítimo para un experimento,
                                    no para la arquitectura del producto
```

Y la comprobación que ordena la decisión, aplicable frontera por frontera:

```
para esta frontera concreta:
  ¿cuál de los seis motivos aplica?
  ¿está medido? (copias, disponibilidad exigida, cambios al mes)
  ¿hay un equipo detrás?

ninguno aplica → módulo, no servicio
```

Y una asimetría importante que conviene saber antes de decidir:

```
de módulo a servicio     se puede hacer después, y cuesta
de servicio a módulo     también se puede, y casi nadie lo hace
                         porque nadie propone «fusionar servicios»
→ por eso conviene empezar por menos unidades y dividir con motivo,
  no al revés
```

3. El monolito modular, hecho bien

El monolito modular no es «el código sin dividir»: es una unidad desplegable con las mismas fronteras que tendrían los servicios, comprobadas automáticamente.

```
un módulo por contexto                                     clase 147
cada uno con su interfaz pública explícita
nadie llama al interior de otro módulo
y nadie accede a las tablas de otro módulo
```

La última es la clave y la que casi nunca se aplica: la regla del escritor único vale también dentro del proceso. Un monolito con esquemas separados por módulo y sin consultas cruzadas es una división real; uno con doscientas tablas comunes no lo es, aunque tenga carpetas bonitas.

Y lo que impide que las fronteras se disuelvan con el tiempo:

```
PRUEBAS DE ARQUITECTURA en la canalización
  «ningún módulo importa el interior de otro»
  «ningún módulo consulta tablas de otro esquema»
  «las dependencias entre módulos no tienen ciclos»
→ es una puerta como las de la clase 100, y falla la construcción
```

Sin eso, la primera urgencia crea el primer atajo y en un año no queda ninguna frontera.

Y lo que se gana frente a los servicios, que es mucho:

```
las llamadas entre módulos no fallan por red
las transacciones funcionan dentro del proceso
```

```
un cambio que cruza dos módulos es un solo despliegue
la depuración es local
y hay UNA canalización, UN objetivo, UNA guardia
```

Y lo que se paga:

```
toda la unidad se despliega junta
toda la unidad escala junta
un fallo de memoria en un módulo afecta a los demás
y un cambio arriesgado arriesga todo el conjunto
```

Y la propiedad que lo hace la opción por defecto razonable:

```
extraer un módulo bien aislado es un trabajo de días
extraer un módulo mal aislado es un proyecto de meses
→ el monolito modular es la forma más barata de CONSERVAR la opción
```

Las funciones encajan como tercera forma para casos concretos, con la aritmética de la clase 117:

```
sí   procesamiento por eventos irregular, integraciones, tareas
     programadas, adaptadores
no   el camino principal de una API con tráfico constante
```

Y conviene no mezclarlas con la decisión anterior: una función es una unidad desplegable más, con sus siete costes de frontera y con los límites propios de su modelo.

4. Extraer, y el peor resultado posible

El orden correcto para extraer un contexto no es el del código:

```
1. AISLAR EL MÓDULO dentro del monolito
   interfaz explícita, sin accesos cruzados, con prueba de arquitectura
   → y esto ya da la mayor parte del beneficio de claridad

2. SEPARAR LOS DATOS
   esquema propio, sin consultas cruzadas, un solo escritor
   → ESTE es el paso difícil, y el que decide el plazo

3. PUBLICAR EL CONTRATO
   la interfaz pasa a ser un contrato versionado              clase 153

4. DESVIAR EL TRÁFICO progresivamente
   una fachada dirige un porcentaje al servicio nuevo
   → con reversión inmediata                                  clase 102

5. RETIRAR el código del monolito
   → y este paso se olvida, exactamente como el «contraer» de la 102
```

Y el aviso que ahorra meses: lo difícil no es el código, son los datos. Separar una tabla que once sitios consultan exige encontrarlos todos, decidir quién copia y quién pregunta, y convivir un tiempo con las dos formas.

El monolito repartido es el peor resultado posible, y es más común que cualquiera de los tres:

```
servicios separados que hay que desplegar JUNTOS
→ paga los siete costes de dividir
→ y no obtiene ninguno de los beneficios
```

Sus señales, que se pueden comprobar:

```
un cambio habitual toca dos o más repositorios
hay que desplegar en un orden concreto
dos servicios comparten tablas                              clase 147
un servicio no arranca si otro no está listo
las versiones tienen que coincidir
las pruebas de un servicio necesitan otro servicio real
y hay una reunión de coordinación de despliegues
```

Y sus causas, por frecuencia:

```
fronteras trazadas por capa o por tecnología                clase 147
datos compartidos, que no se separaron
contratos que cambian a la vez que la implementación
y división hecha antes de entender el dominio
```

La última es la más cara y la más frecuente: dividir al principio, cuando menos se sabe, y descubrir a los seis meses que la frontera está en el sitio equivocado. Mover una frontera dentro de un monolito es refactorizar; moverla entre

servicios es un proyecto.

Y las dos medidas que dicen si la división funciona:

```
proporción de cambios que tocan un solo repositorio
  > 80 %   la división está bien
  < 50 %   es un monolito repartido

proporción de despliegues que hay que coordinar
  ~ 0 %    bien
```

Y la lista de comprobación de la clase:

- las fronteras salen del dominio, no de la tecnología
- para cada frontera de red está escrito cuál de los seis motivos aplica
- ese motivo está medido, no supuesto
- hay un equipo detrás de cada servicio
- el número de servicios cabe en el equipo clase 145
- los módulos internos tienen interfaz explícita y pruebas de arquitectura
- ningún módulo o servicio accede a datos de otro
- las extracciones separan los datos antes de separar el despliegue
- el paso de retirar el código antiguo tiene fecha
- más del 80 % de los cambios tocan un solo repositorio
- no hay despliegues que haya que coordinar

Y el cierre que enlaza con la clase siguiente: al separar, lo que era una transacción pasa a ser una decisión sobre cuánta consistencia necesita cada operación. Y esa decisión no es global ni se toma una vez: es la materia de la clase 149.

Ejemplo trabajado

CloudShop salió de la clase 147 con cinco contextos y ocho servicios. Aquí decide cuántos de esos ocho deben ser procesos separados de verdad, y descubre que tenía un monolito repartido sin saberlo.

El diagnóstico previo: los quince servicios originales.

```
proporción de cambios que tocaban un solo repositorio          38 %
despliegues que había que coordinar                            41 %
servicios que compartían tablas                             11 de 15
servicios que no arrancaban sin otro                         6 de 15
reunión semanal de coordinación de despliegues                  sí
```

Cinco de las siete señales del apartado cuarto. Era un monolito repartido, y llevaba dos años produciendo la sensación de que «los microservicios son complicados».

La decisión, frontera por frontera.

Para cada uno de los cinco contextos se aplicó la comprobación del apartado segundo:

```
VENTAS (precios y promociones)
  perfil de escala          40 copias en campaña, 4 en valle
  disponibilidad            es el flujo principal
  ritmo de cambio           diario; equipo propio
  → SEPARAR: tres motivos, todos medidos

ALMACÉN
  perfil de escala          2-3 copias, estable
  disponibilidad            puede degradarse sin cerrar la tienda
  ritmo de cambio           semanal; equipo propio
  → SEPARAR: ritmo y equipo, y disponibilidad distinta

FACTURACIÓN
  perfil de escala          1-2 copias
  ritmo de cambio           mensual
  equipo propio             NO: lo mantiene el equipo de ventas
  ciclo de vida de datos    4 años frente a meses
  aislamiento               requisitos fiscales
  → SEPARAR: ciclo de vida y aislamiento; sin equipo propio, se
    asigna guardia compartida y se acepta el coste

SOPORTE
  perfil de escala          1 copia
  ritmo de cambio           quincenal
```

```
  equipo propio             no
  ningún otro motivo
  → NO SEPARAR: queda como módulo dentro de ventas

PAGOS
  aislamiento               toca dinero; superficie de ataque acotada
  tecnología                capa de traducción por proveedor  clase 147
  disponibilidad            debe poder caer sin cerrar la tienda
  → SEPARAR

resultado                    4 servicios + 1 módulo
contra los 8 anteriores      y contra los 15 originales
```

Y los tres que desaparecieron:

```
«servicio de notificaciones»   → módulo dentro de cada contexto
                                 que notifica
«servicio de informes»         → consultas sobre el lago    clase 112
«servicio de utilidades»       → biblioteca compartida, sin red
```

La última es la corrección más instructiva: se había convertido en servicio algo que solo se quería reutilizar, y reutilizar no necesita red.

```
llamadas de red al día                     4,1 millones          0
latencia añadida al flujo de compra             38 ms            0
incidentes por su indisponibilidad         3 en 12 meses         0
coste operativo                            canalización,      ninguno
                                           guardia, alertas
```

El monolito modular, para lo que queda junto.

Ventas y soporte quedaron en una unidad desplegable con dos módulos:

```
reglas impuestas por prueba de arquitectura
  soporte no importa el interior de ventas ni al revés
  cada módulo tiene su esquema; sin consultas cruzadas
  las dependencias entre módulos no tienen ciclos

violaciones detectadas al implantar la prueba                  31
  de ellas, consultas cruzadas a tablas                        19
  de ellas, importaciones del interior                         12
tiempo en corregirlas                                  3 semanas
violaciones nuevas bloqueadas en 6 meses                       14
```

Catorce intentos en seis meses de saltarse la frontera, todos bloqueados en la canalización. Sin la prueba, serían catorce atajos permanentes.

La extracción de pagos, por el orden correcto.

```
semana 1-2   aislar el módulo dentro del monolito
             interfaz explícita, prueba de arquitectura
             → ya aquí desapareció el vocabulario del proveedor
               de otros módulos                          clase 147

semana 3-7   separar los datos    ← el paso largo
             tablas de pagos consultadas desde otros sitios       9
             decisiones tomadas: 6 copian por evento, 3 preguntan
             convivencia de las dos formas                 3 semanas

semana 8     publicar el contrato, con versión              clase 153

semana 9-11  desviar tráfico: 1 %, 10 %, 50 %, 100 %        clase 102
             reversiones necesarias                                1
               → un caso de idempotencia que faltaba       clase 116

semana 12    retirar el código antiguo
             → con fecha y en el tablero, para que no se olvidara

tiempo total                                          12 semanas
tiempo dedicado al código                              4 semanas
tiempo dedicado a los DATOS                            5 semanas
```

Los datos costaron más que el código, exactamente como avisa el apartado cuarto.

El resultado, medido con los dos indicadores.

```
cambios que tocan un solo repositorio         38 %              91 %
despliegues que hay que coordinar             41 %               0 %
servicios que comparten tablas             11 de 15           0 de 4
servicios que no arrancan sin otro          6 de 15           0 de 4
reunión de coordinación                        sí                no
fronteras que cruza «confirmar pedido»          7                 2
latencia p99 del flujo de compra             840 ms            210 ms
canalizaciones que mantener                    15                 5
objetivos e indicadores que vigilar            15                 5
servicios por persona de guardia             3,75              1,25
```

Y la latencia bajó a la cuarta parte sin optimizar ningún código: solo por dejar de cruzar cinco fronteras de red.

Lo que se decidió no hacer, y por qué.

```
separar el motor de precios de ventas
  motivo alegado    «es lo más complejo y merece su servicio»
  motivos reales    ninguno de los seis: mismo equipo, mismo ritmo,
                    misma escala, y comparte datos con ventas
  decisión          módulo con interfaz estricta
  revisión          cuando tenga equipo propio o perfil distinto

convertir el adaptador de correo en función
  motivo            tráfico irregular
  cuenta            clase 117: 40 envíos/hora → sí encaja
  decisión          SÍ, es la única función del sistema
```

A los doce meses.

```
unidades desplegables                          15              5
de ellas, funciones                             0              1
módulos con frontera comprobada                 0              4
cambios en un solo repositorio                 38 %           91 %
despliegues coordinados                        41 %            0 %
violaciones de frontera bloqueadas              —             14
latencia p99 del flujo de compra             840 ms          210 ms
incidentes por fallo parcial entre servicios  9 / año         2 / año
tiempo de un cambio que cruza dos contextos  5 semanas       4 días
```

La lección que esta clase traslada a la parte 12: el sistema pasó de quince servicios a cinco unidades y mejoró en todo lo medible: menos latencia, menos incidentes, menos coordinación y cambios cuatro veces más rápidos. Lo que estaba mal no era tener microservicios ni no tenerlos: era que once de quince compartían tablas y el 41 % de los despliegues había que coordinarlos, es decir, se pagaban los siete costes de dividir sin obtener ninguno de los beneficios. Y de las cinco unidades finales, cada una tiene escrito cuál de los seis motivos la justifica, medido.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-12-cloud-native-distributed-architecture/148-monolito-modular-microservicios-y-funcion/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 adr-descomposicion 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 adr-descomposicion 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 cambio habitual obliga a tocar varios repositorios y a desplegar en orden	Monolito repartido: se dividió sin separar datos ni contratos	Mide cambios en un solo repositorio y despliegues coordinados; separa los datos o vuelve a unir lo que no debía separarse.
La latencia del flujo principal es alta sin que ningún componente sea lento	La operación cruza demasiadas fronteras de red	Reduce el número de fronteras que cruza una operación típica; cada una añade latencia, fallo parcial y correlación.
Se separó algo solo para poder reutilizarlo	La reutilización no necesita red	Usa una biblioteca compartida; reserva el servicio para cuando aplique uno de los seis motivos.
Las fronteras internas del monolito se disuelven con el tiempo	Nada impide los atajos cuando hay prisa	Pruebas de arquitectura en la canalización que bloqueen importaciones y consultas cruzadas.
Una extracción que parecía de semanas lleva meses	Se separó el código antes que los datos	Aísla el módulo, separa los datos, publica el contrato, desvía el tráfico y retira el código antiguo, en ese orden.
Hay más servicios que personas para operarlos	El número de unidades no se trató como restricción	Limita las unidades a lo que el equipo puede operar y exige un equipo detrás de cada frontera dura.

■ Seguridad, ética y costo

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

■ Preguntas de comprobación

1. ¿Cuáles son los siete costes de convertir una frontera interna en una de red?
2. ¿Qué seis motivos justifican separar y cuáles no lo justifican nunca?
3. ¿Qué distingue un monolito modular de un monolito sin dividir?
4. ¿Por qué los datos cuestan más que el código en una extracción?
5. ¿Qué dos medidas revelan un monolito repartido?

Referencias

- Fowler, M. (2015). Monolith first — por qué conviene empezar por menos unidades y dividir con motivo. <https://martinfowler.com/bliki/MonolithFirst.html>
- Newman, S. (2019). Monolith to Microservices — extracción progresiva y separación de datos. <https://samnewman.io/books/monolith-to-microservices/>
- Richardson, C. (2025). Microservice architecture: the pattern and its drawbacks — beneficios y costes explícitos. <https://microservices.io/patterns/microservices.html>
- Tornhill, A. (2025). Architectural fitness functions and dependency rules — pruebas de arquitectura automatizadas. <https://www.adamtornhill.com/>
- Ford, N. y otros (2021). Software Architecture: The Hard Parts — criterios de granularidad y desintegración. <https://www.oreilly.com/library/view/software-architecture-the/9781492086888/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 12 en PDF · Manual integral

Anterior	Índice	Siguiente
← 147 · DDD, bounded contexts y ownership de datos	Parte 12 · Programa	149 · CAP, PACELC y consistencia por operación →

Evaluación — 148 Monolito modular, microservicios y función

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-descomposicion 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.

149 — CAP, PACELC y consistencia por operación

Parte: 12 — Arquitectura cloud-native y sistemas distribuidos Nivel: avanzado-experto · Horas estimadas: 4
Laboratorio: distributed · Estado: EXECUTABLECORE

Propósito

Decidir cuánta consistencia necesita cada operación, que es una decisión por operación y no por sistema. La clase enuncia con precisión el teorema que todo el mundo cita mal —habla solo de lo que ocurre durante una partición de red, y no dice nada del 99,99 % del tiempo restante—, presenta la extensión que sí describe el día a día, y desmonta la palabra «eventual», que esconde comportamientos muy distintos. Y termina con la idea que ahorra más consistencia de la que cualquier mecanismo puede dar: si las operaciones conmutan, el orden deja de importar.

Resultados de aprendizaje

Al finalizar podrás:

1. Enunciar correctamente qué dice y qué no dice el teorema.
2. Usar la extensión que describe el comportamiento sin partición.
3. Distinguir los niveles que se esconden bajo la palabra «eventual».
4. Clasificar cada operación por lo que cuesta una lectura desfasada.
5. Diseñar operaciones que conmuten para necesitar menos consistencia.

Conceptos centrales

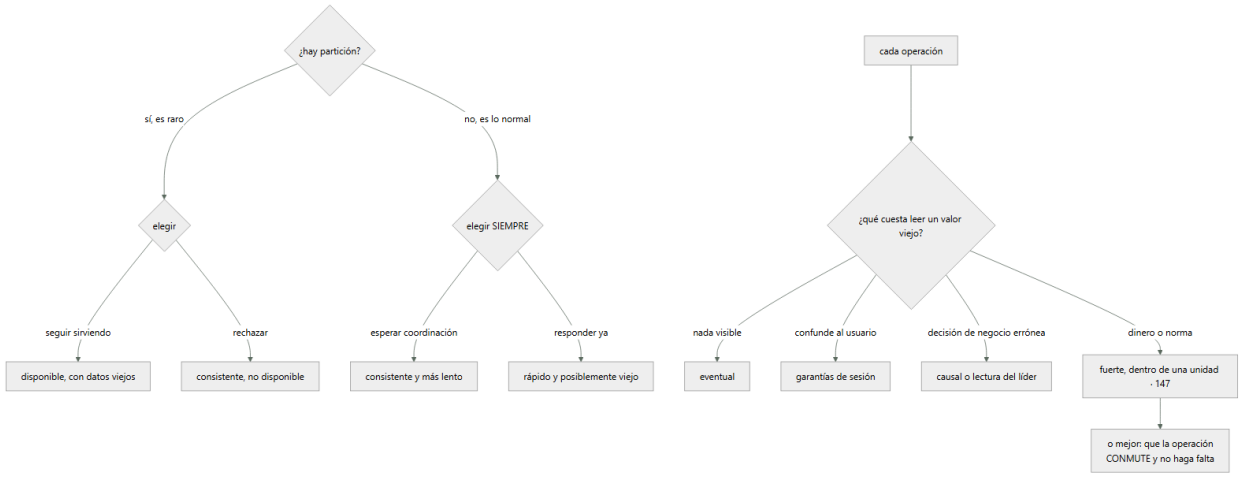
Concepto	Comprensión verificable
partición de red	Situación en la que dos partes del sistema no se ven entre sí pero ambas siguen vivas. No se elige: ocurre.
garantías de sesión	Leer lo que uno mismo escribió, no retroceder en el tiempo y ver las escrituras propias en orden. Son baratas y resuelven casi todo lo visible.
consistencia causal	Si un cambio depende de otro, todos lo ven en ese orden. No exige orden global.
escritura condicional	Aplicar el cambio solo si el dato sigue en la versión esperada. Detecta el conflicto en vez de ignorarlo.
conmutatividad	Que el resultado no dependa del orden de las operaciones. Elimina la necesidad de coordinar.
gana la última escritura	Resolver conflictos por marca de tiempo. Con relojes distintos, pierde silenciosamente datos correctos.

Modelo mental

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

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 el teorema dice de verdad

La formulación popular —«elige dos de tres»— es incorrecta y ha causado mucho daño. El enunciado real es más estrecho y más útil:

```
CUANDO hay una partición de red, un sistema distribuido
no puede a la vez responder a todas las peticiones
y garantizar que todas ven el último valor
```

Y de ahí las dos precisiones que cambian su uso:

```
1. LA PARTICIÓN NO SE ELIGE
   ocurre: un enlace se cae, una zona se aísla, un cortafuegos falla
   → así que no hay tres opciones: hay una decisión sobre qué hacer
     cuando ocurra

2. NO DICE NADA DEL RESTO DEL TIEMPO
   y el resto del tiempo es el 99,99 %
```

La segunda es la importante, y por eso hace falta la extensión que sí describe el día a día:

```
SI hay partición  → elegir entre seguir sirviendo o ser exacto
SI NO la hay      → elegir entre LATENCIA y CONSISTENCIA
```

Y esa segunda mitad es la que se paga todos los días:

```
leer del líder                  exacto, y más lento y menos disponible
leer de una réplica cercana     rápido, y puede estar desfasada
esperar confirmación de N       exacto, y añade la latencia de coordinar
confirmar y propagar después    rápido, y hay una ventana de pérdida
```

Y dos ejemplos concretos de esa elección, en decisiones que este programa ya tomó:

```
réplicas de lectura                                        clase 109
  leer de la réplica es rápido y puede devolver algo viejo

caché con clave versionada                                 clase 111
  servir del caché es rápido; el desfase es la ventana acumulada
```

Y la conclusión que ordena la clase:

```
la consistencia no es una propiedad del sistema
es una decisión POR OPERACIÓN
```

Un mismo sistema tiene operaciones que no pueden equivocarse nunca —descontar la última unidad— y operaciones que toleran minutos de desfase —una recomendación—. Tratarlas igual significa pagar la más cara para todas o aceptar el riesgo de la más barata en todas.

2. Lo que esconde la palabra «eventual»

«Eventualmente consistente» solo promete que, si dejan de llegar escrituras, al final todos verán lo mismo. No promete nada sobre lo que ocurre mientras tanto, y ahí caben comportamientos muy distintos:

```
FUERTE (linealizable)
  toda lectura ve la última escritura confirmada
  como si hubiera una sola copia
  → caro: coordinación en cada operación

CAUSAL
  si un cambio depende de otro, todos los ven en ese orden
  no exige orden global entre cambios independientes
  → mucho más barato y suficiente para casi todo lo que confunde

GARANTÍAS DE SESIÓN     ← las prácticas
  leer lo propio     ves lo que TÚ acabas de escribir
  no retroceder      una lectura posterior no devuelve algo más viejo
  escrituras en orden tus escrituras se aplican en el orden que las hiciste

EVENTUAL A SECAS
  cualquier cosa mientras tanto: puedes ver un valor, luego uno
  más antiguo, y luego el nuevo
```

Y el hallazgo práctico de esta clase:

```
casi todos los problemas VISIBLES para un usuario los resuelven
las garantías de sesión, que son baratas

«guardo mi perfil y sale el viejo»            leer lo propio
«actualizo y al recargar desaparece»          no retroceder
«mis dos cambios se aplicaron al revés»       escrituras en orden
```

Y cómo se consiguen, sin cambiar de base de datos:

```
rutar al líder durante N segundos tras escribir      clase 109
o llevar la versión leída y exigir al menos esa       token de sesión
o fijar la sesión a una réplica concreta
```

La clasificación por operación, que es la técnica central. La pregunta:

```
¿qué ocurre si esta lectura devuelve un valor de hace 5 segundos?
```

Y la escala de respuestas decide el nivel:

```
NADA VISIBLE                    → eventual
  recomendaciones, contador de visitas, listados de catálogo

CONFUNDE AL USUARIO             → garantías de sesión
  su perfil, su cesta, su último pedido

DECISIÓN DE NEGOCIO ERRÓNEA     → causal o lectura del líder
  mostrar disponible algo agotado, aplicar una promoción retirada

SE PIERDE DINERO O SE INCUMPLE  → fuerte, y dentro de una unidad
  descontar la última unidad, cobrar, aplicar un límite de crédito
```

Y la observación de la clase 109, que aquí se generaliza:

```
toda lectura que DECIDE una escritura necesita el nivel de esa escritura
→ comprobar existencia antes de insertar
→ leer el saldo antes de descontarlo
```

3. Mecanismos, y el que evita el problema

Los mecanismos disponibles, del más caro al más barato:

```
TRANSACCIÓN                exacta, y solo dentro de una unidad de
                           consistencia                        clase 147

LECTURA DEL LÍDER          exacta para esa lectura; concentra carga

QUÓRUM                     leer y escribir en la mayoría
                           → si lectura + escritura > total, se ve
                             lo último
                           → y cuesta latencia y disponibilidad

ESCRITURA CONDICIONAL      aplicar solo si la versión es la esperada
                           → no evita el conflicto: lo DETECTA
                           → y detectar es mucho mejor que ignorar

GARANTÍAS DE SESIÓN        baratas, resuelven lo visible
```

```
EVENTUAL                    gratis
```

Y la escritura condicional merece detalle porque resuelve la mayoría de los casos intermedios con muy poco coste:

```
UPDATE producto SET stock = 4, version = 8
WHERE id = 'X' AND version = 7

si afecta a 0 filas → alguien cambió el dato entre medias
→ y entonces se relee y se reintenta, o se informa al usuario
```

Y el mismo mecanismo sirve para el desorden de la clase 114: si llega la versión 3 después de la 4, se descarta.

Y la idea que evita el problema en vez de resolverlo:

```
SI LAS OPERACIONES CONMUTAN, EL ORDEN NO IMPORTA

no conmuta        stock = 4       dos escrituras concurrentes:
                                  una se pierde
conmuta           stock -= 1      da igual el orden; el resultado
                                  es el mismo
```

Y de ahí una regla de diseño con más alcance del que parece:

```
expresar el cambio como INCREMENTO o como HECHO añadido,
no como asignación del resultado
→ «reservada 1 unidad» en vez de «stock = 4»
→ y el valor actual se calcula, o se mantiene como proyección
```

Y su límite honesto: conmutar no garantiza la invariante. Dos decrementos conmutan y pueden llevar el stock a −1; para impedirlo hace falta o coordinación, o aceptar el sobreventa y compensar —que es lo que hacen muchos negocios a propósito—.

```
coordinar      no vender nunca de más; se pierde disponibilidad
compensar      vender de más raramente y resolverlo después
               → decisión de negocio, no técnica
```

Y estructuras de datos que conmutan por construcción —contadores, conjuntos, mapas de última escritura por campo— resuelven casos concretos como contadores compartidos, listas de favoritos o presencia, y no resuelven una invariante de negocio.

4. El tiempo, que no es de fiar

Muchos sistemas resuelven conflictos con la marca de tiempo: gana la última escritura. Y es peligroso por una razón sencilla:

```
los relojes de dos máquinas no coinciden
desviación habitual con sincronización        1-50 ms
desviación cuando la sincronización falla     segundos o más
```

Y con eso:

```
la escritura A ocurre después de la B en la realidad
pero la máquina de B tiene el reloj 200 ms adelantado
→ gana B
→ y A se pierde EN SILENCIO: no hay error, no hay conflicto,
  simplemente el dato correcto desaparece
```

Las alternativas, en orden de preferencia:

```
VERSIÓN POR ELEMENTO        un número que crece con cada escritura
                            → la escritura condicional del apartado anterior
RELOJ LÓGICO                cuenta causalidad, no tiempo real
CONFLICTO EXPLÍCITO         guardar las dos versiones y decidir arriba
                            → lo hacen algunos almacenes, y obliga a
                              escribir la lógica de resolución
RELOJ CON INCERTIDUMBRE     el sistema espera a que el intervalo pase
                            → es lo que permite orden global en algunas
                              bases distribuidas, y cuesta latencia
```

Y dos avisos prácticos sobre el tiempo:

```
nunca uses la marca de tiempo del cliente para decidir orden
  → el reloj del navegador o del móvil puede estar en otro año
y distingue el momento del HECHO del momento de la publicación
  → ya lo exigía la envoltura de la clase 115
```

Y una comprobación que conviene ensayar, porque es un experimento de la clase 131 que casi nadie hace:

```
desajustar el reloj de una máquina a propósito
y ver qué se rompe
```

Y la lista de comprobación de la clase:

- está escrito qué hace el sistema durante una partición
- para cada operación está escrito qué cuesta una lectura desfasada
- el nivel de consistencia se decide por operación, no por sistema
- hay garantías de sesión donde el desfase confunde al usuario
- ninguna lectura que decide una escritura usa un nivel más débil
- las escrituras concurrentes usan versión, no marca de tiempo
- ninguna resolución de conflicto es «gana la última escritura»
- los cambios se expresan como incremento o hecho, no como asignación
- está escrito si una invariante se coordina o se compensa
- el momento del hecho y el de publicación están separados
- se ha ensayado un desajuste de reloj

Y el cierre que enlaza con la clase siguiente: estas decisiones se apoyan en cómo el almacén copia y reparte los datos por debajo. Qué garantiza cada forma de replicar, qué cuesta el consenso y por qué el número de copias no es una decisión menor es la materia de la clase 150.

Ejemplo trabajado

CloudShop clasifica sus veinte operaciones principales por lo que cuesta una lectura desfasada. El ejercicio dura una tarde, cambia tres diseños y explica dos incidentes antiguos que nadie había relacionado.

La clasificación.

```
NADA VISIBLE → eventual                                          12
  listado de catálogo, búsqueda, recomendaciones,
  contadores de visitas, valoraciones, productos relacionados,
  histórico de pedidos, panel de informes, sugerencias,
  productos vistos, novedades, tendencias

CONFUNDE AL USUARIO → garantías de sesión                         5
  ver mi perfil tras editarlo
  ver mi pedido recién hecho
  ver mi cesta tras añadir
  ver mi dirección tras cambiarla
  ver el estado de mi devolución tras solicitarla

DECISIÓN ERRÓNEA → causal o lectura del líder                     1
  mostrar disponibilidad en la ficha de producto

DINERO O NORMA → fuerte, dentro de una unidad                     2
  reservar la última unidad
  aplicar el cobro
```

Doce de veinte no necesitaban nada, y dos necesitaban todo. Antes del ejercicio, las veinte leían del líder «por seguridad».

```
operaciones que leen del líder                  20              3
carga de lectura sobre el líder            12.000/s          1.900/s
latencia p99 de listados                      41 ms            12 ms
coste de la base                           5.100 €         3.200 €
```

Las garantías de sesión, implantadas sin cambiar de base.

El mecanismo fue el de la clase 109, generalizado:

```
tras cualquier escritura, esa sesión lee del líder durante 5 s
y las respuestas llevan la versión leída, que el cliente devuelve
```

```
quejas de «no veo lo que acabo de guardar»   31 / 6 semanas      0
operaciones que leen del líder siempre            20             3
lecturas dirigidas al líder por sesión            —          4,1 % del total
```

El caso de la disponibilidad, que era el interesante.

```
mostrar «en stock» en la ficha de producto
  si está desfasado 5 s → el cliente añade al carrito algo agotado
  → y se entera al pagar
```

La primera propuesta fue leer del líder, y el cálculo lo desaconsejó:

```
lecturas de ficha de producto                      9.400/s
lecturas del líder que eso añadiría                9.400/s
capacidad del líder                                1.500/s
→ inviable
```

La solución fue distinguir dos cosas que se llamaban igual:

```
MOSTRAR disponibilidad     eventual, con margen
                           «quedan pocas unidades» en vez de un número
                           desfase tolerado: 30 s

RESERVAR                   fuerte, con escritura condicional
                           es donde de verdad importa
```

```
lecturas al líder por disponibilidad         9.400/s          0
clientes que llegan al pago con algo agotado   1,4 %         0,3 %
reservas fallidas por conflicto                 —           0,3 %
                                                        (con mensaje claro)
```

El incidente que la clasificación explicó: gana la última escritura.

```
síntoma histórico   cambios de dirección de envío que «se deshacían»
frecuencia          unos 8 casos al mes, sin patrón aparente
diagnóstico previo  «los clientes se confunden»
```

El almacén de perfiles resolvía conflictos por marca de tiempo:

```
el cliente cambia la dirección desde el móvil     10:00:00,150
un proceso de normalización de direcciones
reescribe el registro                             10:00:00,090
                                                  (reloj adelantado 200 ms)
→ gana el proceso, y el cambio del cliente desaparece
→ sin error, sin conflicto y sin rastro
```

```
resolución de conflictos              marca de tiempo    versión por elemento
qué pasa con un conflicto             se pierde uno      falla y se reintenta
                                      en silencio        releyendo
casos de «se deshizo mi cambio»          8 / mes             0
desviación de reloj medida entre nodos   hasta 340 ms    vigilada, alerta > 50 ms
```

Y el ensayo de la clase 131 que se añadió:

```
desajustar el reloj de un nodo 2 segundos, a propósito
primera vez           3 comportamientos rotos
  → expiración de sesiones, firma de avisos y ordenación de eventos
tras corregirlos      sin efecto observable
```

La conmutatividad, aplicada al inventario.

El modelo original asignaba el resultado:

```
UPDATE producto SET stock = 4 WHERE id = 'X'
→ dos reservas simultáneas: una se pierde
→ se vendían dos unidades de la última
```

Y al expresarlo como hechos que conmutan:

```
INSERT INTO movimientos (producto, delta, motivo, id_operacion)
VALUES ('X', -1, 'reserva', 'op-9f2c')

con restricción única sobre id_operacion             clase 116
y el stock actual como proyección mantenida
```

```
reservas simultáneas                  una se pierde     ambas se registran
sobreventa                            posible           controlada
cómo se impide vender de más          no se impedía     escritura condicional
                                                        sobre la proyección
histórico de por qué cambió el stock  no existía        completo
```

Y la última fila fue un beneficio inesperado: el historial de movimientos resolvió las discrepancias de inventario que antes se investigaban a mano.

Y la decisión de negocio, escrita explícitamente:

```
¿coordinar para no vender nunca de más, o compensar?
  coordinar        reservar exige lectura fuerte; latencia +40 ms
                   y con el líder caído no se puede vender
  compensar        se acepta sobreventa en la ventana de partición
```

```
                  y se avisa al cliente

decisión   COORDINAR para el flujo normal
           COMPENSAR durante una partición: se acepta el pedido,
           se marca «pendiente de confirmar disponibilidad»
           y se resuelve al recuperarse

casos de compensación en 12 meses                              6
de ellos, resueltos con stock disponible                       5
de ellos, con disculpa y cupón                                 1
```

Lo que hace el sistema durante una partición, escrito.

```
si la réplica no ve al líder
  lecturas eventuales      se sirven de la réplica           ✓
  garantías de sesión      se degradan: se avisa «datos de hace X»
  disponibilidad mostrada  se sirve con margen mayor
  reservas y cobros        se RECHAZAN con mensaje claro
                           salvo el modo compensado, si se activa
```

Y eso se ensayó con el experimento correspondiente:

```
partición provocada de 4 minutos
  peticiones servidas                                    94 %
  peticiones rechazadas                                   6 %  (reservas)
  datos incorrectos servidos                              0
  pedidos perdidos                                        0
```

A los seis meses.

```
operaciones con nivel decidido                  0 de 20      20 de 20
operaciones que leen del líder                  20             3
carga de lectura sobre el líder              12.000/s       1.900/s
latencia p99 de listados                       41 ms          12 ms
quejas de «no veo lo que guardé»            31 / 6 sem          0
casos de «se deshizo mi cambio»                8 / mes          0
sobreventa de la última unidad              4 / trimestre       0
resolución de conflictos por marca de tiempo    sí              no
histórico de movimientos de stock               no              sí
comportamiento durante partición, escrito       no              sí
y ensayado                                      no          semestral
```

La lección que esta clase traslada a la parte 12: de veinte operaciones, doce no necesitaban ninguna garantía y dos las necesitaban todas, y antes del ejercicio las veinte pagaban el precio de las dos. Y el incidente que más tiempo llevaba sin explicación —ocho cambios de dirección al mes que «se deshacían»— no era un problema de consistencia insuficiente: era una resolución de conflictos por marca de tiempo con los relojes desajustados 340 milisegundos, que perdía el dato correcto sin dejar ningún rastro.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-12-cloud-native-distributed-architecture/149-cap-pacelc-y-consistencia-por-operacion/lab.py
```

El laboratorio selecciona el motor de práctica distributed 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-consistencia 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 traza de consistencia, reintento o fallo parcial. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye matriz-consistencia 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
Todas las lecturas van al líder y este se satura	La consistencia se decidió para el sistema entero en vez de por operación	Clasifica cada operación por lo que cuesta una lectura desfasada; casi siempre la mayoría tolera eventual.
El usuario no ve lo que acaba de guardar	Falta la garantía de leer lo propio	Ruta al líder durante unos segundos tras escribir, o lleva la versión leída en la sesión.
Un cambio correcto desaparece sin error ni conflicto	La resolución es por marca de tiempo y los relojes no coinciden	Resuelve por versión por elemento con escritura condicional, y vigila la desviación de reloj.
Dos operaciones concurrentes y una se pierde	El cambio se expresa como asignación del resultado	Exprésalo como incremento o como hecho añadido; si conmuta, el orden deja de importar.
Se vende la última unidad dos veces	Una lectura débil decide una escritura	Toda lectura que decide una escritura necesita el nivel de esa escritura; y escribe si la invariante se coordina o se compensa.
Nadie sabe qué hace el sistema si se parte la red	No se ha decidido; se descubrirá durante el incidente	Escríbelo por familia de operación y ensáyalo provocando una partición.

■ 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 dice exactamente el teorema y qué no dice?
2. ¿Qué elección se hace el 99,99 % del tiempo, cuando no hay partición?
3. ¿Qué tres garantías de sesión existen y qué problemas visibles resuelven?
4. ¿Por qué «gana la última escritura» pierde datos en silencio?
5. ¿Qué gana expresar un cambio como incremento en vez de como asignación?

Referencias

- Brewer, E. (2012). CAP twelve years later: how the rules have changed — el enunciado correcto y sus matices. <https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/>

- Abadi, D. (2012). Consistency tradeoffs in modern distributed database design — la extensión que incluye la latencia. <https://www.cs.umd.edu/abadi/papers/abadi-pacelc.pdf>
- Terry, D. y otros (1994). Session guarantees for weakly consistent replicated data — leer lo propio y no retroceder. <https://dl.acm.org/doi/10.5555/645792.668302>
- Kleppmann, M. (2017). Designing Data-Intensive Applications, cap. 5 y 9 — replicación, relojes y consistencia. <https://dataintensive.net/>
- Shapiro, M. y otros (2011). Conflict-free replicated data types — estructuras que conmutan por construcción. <https://inria.hal.science/inria-00555588/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 12 en PDF · Manual integral

Anterior	Índice	Siguiente
← 148 · Monolito modular, microservicios y función	Parte 12 · Programa	150 · Replicación, particionado y consenso →

Evaluación — 149 CAP, PACELC y consistencia por operación

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-consistencia 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.

150 — Replicación, particionado y consenso

Parte: 12 — Arquitectura cloud-native y sistemas distribuidos Nivel: avanzado-experto · Horas estimadas: 4
Laboratorio: distributed · Estado: EXECUTABLECORE

Propósito

Entender qué hace por debajo el almacén que sostiene las decisiones de la clase 149: copiar los datos, que da disponibilidad y capacidad de lectura, y repartirlos, que da capacidad de escritura y tamaño. La clase separa los dos ejes, muestra lo que garantiza cada forma de copiar y lo que cuesta cada una, explica el mecanismo que impide el peor fallo de todos —dos nodos creyéndose líderes a la vez— y sitúa el consenso donde corresponde: para poco dato y muy crítico, nunca en el camino de las peticiones.

Resultados de aprendizaje

Al finalizar podrás:

1. Separar replicar de repartir, y componer los dos.
2. Elegir la topología de copia según quién escribe y desde dónde.
3. Cuantificar lo que se pierde en una conmutación asíncrona.
4. Explicar cómo se evita que dos nodos actúen como líder a la vez.
5. Repartir de forma que reequilibrar no rompa nada, y situar el consenso.

Conceptos centrales

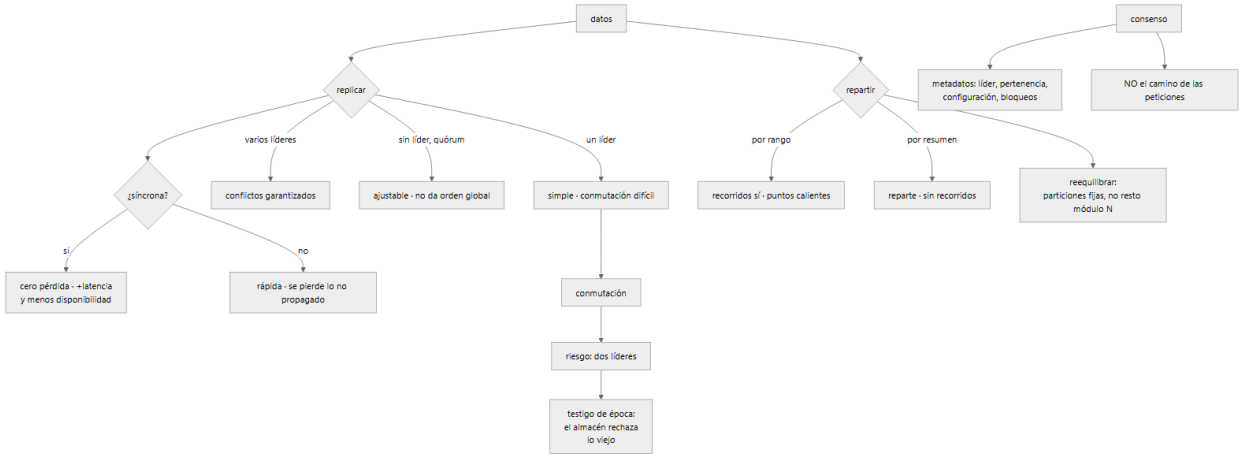
Concepto	Comprensión verificable
replicación	Varias copias del mismo dato. Da disponibilidad y capacidad de lectura; no da capacidad de escritura.
particionado	Repartir datos distintos en nodos distintos. Da capacidad de escritura y tamaño; no da disponibilidad por sí solo.
punto de recuperación	Cuántos datos se pierden como máximo al conmutar. Con copia asíncrona, es mayor que cero por definición.
cerebro dividido	Dos nodos creyéndose líderes a la vez. Produce escrituras divergentes que después nadie sabe reconciliar.
testigo de época	Número creciente que acompaña a cada acción del líder. El almacén rechaza lo que llegue con una época vieja.
particiones fijas	Crear muchas más particiones que nodos y mover particiones enteras al reequilibrar, sin recalcular la asignación de claves.
consenso	Acordar un valor pese a fallos. Necesita mayoría, cuesta viajes de red y empeora al añadir nodos.

Modelo mental

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

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

■ Flujo de razonamiento



Desarrollo

1. Dos ejes que se confunden

```
REPLICAR    varias copias del MISMO dato
  da        disponibilidad ante la pérdida de un nodo
            capacidad de lectura
  no da     capacidad de escritura: todas las copias escriben lo mismo

REPARTIR    datos DISTINTOS en nodos distintos
  da        capacidad de escritura y tamaño
  no da     disponibilidad: si cae la partición 3, se pierde su parte
```

Y casi todo sistema real necesita los dos, compuestos:

```
24 particiones × 3 copias cada una = 72 unidades de datos
→ cada partición tiene su líder y sus réplicas
→ y el líder de cada partición puede estar en un nodo distinto
```

Y el error habitual es pedirle a uno lo que da el otro:

```
«añadimos réplicas porque no damos abasto escribiendo»
  → no sirve: las réplicas escriben lo mismo
«particionamos para tolerar la caída de un nodo»
  → no sirve: se pierde esa partición
```

Las topologías de copia, por quién puede escribir:

```
UN SOLO LÍDER
  todas las escrituras van a un nodo; las copias lo siguen
  + simple, sin conflictos, orden claro
  – el líder es un límite de escritura
  – y la conmutación es el problema difícil

VARIOS LÍDERES
  se escribe en cualquiera y se propagan entre ellos
  + escritura local en varias regiones; tolera clientes desconectados
  – LOS CONFLICTOS SON INEVITABLES, no excepcionales
  → solo se justifica por escritura multirregión o trabajo sin conexión

SIN LÍDER, POR QUÓRUM
  el cliente escribe en varias copias y lee de varias
  si lecturas + escrituras > total, se ve lo último escrito
  + tolera nodos caídos sin conmutar nada
  – no da orden global ni operaciones condicionales fáciles
  – y las reparaciones en segundo plano son parte del diseño
```

Y una precisión que se pasa por alto: la desigualdad del quórum no garantiza linealizabilidad. Garantiza que alguna copia leída tiene el último valor confirmado, y con escrituras concurrentes o fallidas a medias siguen apareciendo casos que hay que resolver con versiones.

2. Síncrono, asíncrono y lo que se pierde

La decisión con más consecuencias del apartado anterior:

```
SÍNCRONA     el líder no confirma hasta que la copia ha recibido
  + no se pierde nada al conmutar
  − cada escritura paga el viaje de ida y vuelta
  − y si la copia no responde, el líder SE BLOQUEA

ASÍNCRONA    el líder confirma y propaga después
  + rápida, y el líder no depende de la copia
  − al conmutar se pierde lo que no llegó a propagarse

SEMISÍNCRONA el compromiso habitual
  una copia síncrona, las demás asíncronas
  → si esa copia se atasca, se promueve otra a síncrona
```

Y la cifra que hay que conocer y casi nadie mide:

```
PUNTO DE RECUPERACIÓN = cuántos datos se pierden al conmutar

con copia asíncrona, se estima con el retardo de replicación
  retardo habitual        20-200 ms  → decenas de escrituras
  retardo bajo carga      1-30 s     → miles de escrituras
```

Y el mismo dato dicho de la forma que obliga a decidir:

```
«si el líder se pierde ahora mismo, ¿cuántos pedidos desaparecen?»
```

Y la elección se hace por dato, no por sistema, igual que en la clase 149:

```
pedidos y cobros            síncrona o semisíncrona: no se pierden
sesiones y carritos         asíncrona: se acepta perder unos segundos
telemetría y contadores     asíncrona sin dudarlo
```

La conmutación, que es donde falla todo. Sus dos peligros:

```
ESCRITURAS PERDIDAS
  se promueve una copia que iba retrasada
  → lo que confirmó el líder viejo y no llegó, desaparece
  → y si el líder viejo vuelve, hay que decidir qué hacer con ello

DOS LÍDERES A LA VEZ
  el líder viejo no está caído: está aislado o pausado
  y sigue creyendo que es el líder
  → dos nodos aceptan escrituras divergentes
  → y después nadie sabe reconciliarlas
```

Y el segundo es el que produce daños irreparables, porque no da ningún error mientras ocurre.

El mecanismo que lo impide es sencillo y conviene conocerlo:

```
TESTIGO DE ÉPOCA
  cada vez que se elige un líder, el número de época sube
  toda operación del líder lleva su época
  y el almacén RECHAZA cualquier operación con una época menor
    que la mayor que ha visto

→ el líder viejo puede creer lo que quiera; sus escrituras se rechazan
→ la garantía no está en el líder: está en el que recibe
```

Y la lección general, que vale para cualquier sistema con bloqueos distribuidos:

```
nunca confíes en que quien tenía un permiso siga teniéndolo
que lo compruebe quien ejecuta, con un número que solo crece
```

3. Repartir, y reequilibrar sin romper

Cómo se decide en qué partición va cada dato:

```
POR RANGO DE CLAVE
  A-F en la 1, G-M en la 2, …
  + los recorridos por rango son eficientes
  − puntos calientes si las claves no se distribuyen
    → «todo lo de hoy» va a la misma partición        clase 114

POR RESUMEN DE LA CLAVE
  + reparte muy bien
  − se pierden los recorridos por rango: quedan desperdigados

COMBINADO
```

```
  resumen del primer componente, rango del segundo
  → «todos los eventos del pedido X, ordenados por tiempo»
  → es lo que hacen los almacenes de columna ancha    clase 110

DIRECTORIO EXPLÍCITO
  una tabla dice dónde está cada rango
  + máxima flexibilidad para mover
  − ese directorio es un componente crítico más
```

El reequilibrado, que es donde está el error clásico:

```
mal    partición = resumen(clave) mod N
       → al cambiar N, casi TODAS las claves cambian de sitio
       → y el orden por clave se rompe                clase 114

bien   PARTICIONES FIJAS
       se crean muchas más particiones que nodos: 256 para 8 nodos
       cada nodo aloja varias
       al añadir un nodo, se le MUEVEN particiones enteras
       → la asignación clave→partición nunca cambia

bien   RESUMEN CONSISTENTE
       nodos y claves en un anillo; al añadir uno, solo se mueve
       la parte contigua
```

Y el número de particiones vuelve a ser la decisión irreversible de la clase 114:

```
demasiadas   metadatos, ficheros y reequilibrados lentos
demasiado pocas  techo de paralelismo, y ampliar duele
regla práctica   suficientes para el crecimiento previsto ×2 o ×3
```

Los índices secundarios, que es el problema que aparece en cuanto se reparte:

```
LOCAL A CADA PARTICIÓN
  cada partición indexa lo suyo
  + escribir es barato: solo toca la partición del dato
  − consultar exige preguntar a TODAS y unir los resultados
  → y la latencia la marca la partición más lenta   clase 124

GLOBAL, PARTICIONADO POR EL CAMPO INDEXADO
  + consultar toca una sola partición del índice
  − escribir toca dos particiones distintas
  → y esa escritura ya no es atómica: suele ser asíncrona
```

Y la consecuencia práctica del segundo: el índice global va por detrás, así que una consulta por ese índice es eventualmente consistente aunque el dato no lo sea. Es un caso más de la clasificación de la clase 149.

4. Consenso: para qué sí y para qué no

El consenso resuelve un problema muy concreto: que un grupo de nodos acuerde un valor pese a que algunos fallen o no se vean.

Y se usa para poco dato y muy crítico:

```
quién es el líder de cada partición
qué nodos forman el grupo
la configuración: dónde está cada partición
bloqueos y arrendamientos
y el número de época del apartado segundo
```

Sus propiedades, que hay que aceptar:

```
NECESITA MAYORÍA
  con 5 nodos, funciona con 3; con 2 de 5, se detiene
  → detenerse es la respuesta CORRECTA: seguir sería divergir

CUESTA VIAJES DE RED
  cada decisión son al menos dos rondas

NO ESCALA AÑADIENDO NODOS
  más nodos → más mensajes → decisiones más lentas
  → 3 o 5 nodos; 7 solo si hay motivo; nunca 50

EL NÚMERO IMPAR IMPORTA
  4 nodos toleran 1 fallo, igual que 3
```

```
  → el cuarto añade coste sin añadir tolerancia
```

Y la regla que ordena su uso:

```
CONSENSO PARA LOS METADATOS
NUNCA EN EL CAMINO DE CADA PETICIÓN
```

Si cada operación de negocio necesita un acuerdo entre nodos, el sistema tendrá la latencia y la disponibilidad del acuerdo. Lo que hacen los almacenes bien diseñados es usar consenso para decidir quién manda, y luego escribir contra el que manda.

Y una advertencia sobre los bloqueos distribuidos, que es donde más se equivoca la gente:

```
un bloqueo distribuido con caducidad NO garantiza exclusión
  el proceso puede pausarse justo después de obtenerlo
  la caducidad vence, otro lo toma
  y el primero despierta creyendo que aún lo tiene
→ hace falta el testigo de época comprobado por quien EJECUTA
→ y si el recurso no puede comprobarlo, el bloqueo es orientativo
```

Y el corolario práctico: muchos usos de un bloqueo distribuido se resuelven mejor con idempotencia —clase 116—, porque entonces da igual que se ejecute dos veces.

Y qué mirar de todo esto en un servicio gestionado, que es lo habitual:

```
¿la copia es síncrona, asíncrona o mixta? ¿se puede elegir?
¿cuál es el punto de recuperación anunciado y el medido?
¿cuánto tarda una conmutación, medido?                clase 109
¿qué pasa con las escrituras en vuelo?
¿hay protección contra dos líderes, y cómo?
¿cuántas particiones tiene y se pueden ampliar?       clase 114
¿los índices secundarios son locales o globales?
¿qué ocurre cuando se pierde la mayoría?
```

Y la lista de comprobación de la clase:

- ■ está claro qué problema resuelve replicar y cuál repartir
- ■ la topología de copia corresponde a quién escribe y desde dónde
- ■ hay varios líderes solo si hay motivo multirregión o sin conexión
- ■ está decidido por dato si la copia es síncrona o asíncrona
- ■ el punto de recuperación está medido, no solo anunciado
- ■ existe protección contra dos líderes y se ha comprobado
- ■ la asignación clave→partición no depende del número de nodos
- ■ el número de particiones cubre el crecimiento previsto
- ■ está claro si los índices secundarios son locales o globales
- ■ el consenso solo gobierna metadatos, no el camino de las peticiones
- ■ los grupos de consenso tienen número impar y pocos nodos
- ■ ningún bloqueo distribuido se usa sin comprobación en quien ejecuta

Y el cierre que enlaza con la clase siguiente: replicar y repartir reducen la probabilidad de perderlo todo y aumentan la de que algo falle en parte. Cómo se comporta un sistema cuando una pieza responde mal, tarde o a medias es la materia de la clase 151.

Ejemplo trabajado

CloudShop tiene su base principal con una réplica y un almacén no relacional con 24 particiones. El ejercicio consiste en medir lo que de verdad garantizan, y produce dos sorpresas: el punto de recuperación real y un caso de dos líderes que había ocurrido sin que nadie lo supiera.

Medir el punto de recuperación.

Lo anunciado y lo medido:

```
anunciado por el proveedor        «pérdida mínima»
configuración real                copia asíncrona
retardo de replicación, mediana                      41 ms
retardo, percentil 99                               380 ms
retardo durante el cierre mensual                   11 s
escrituras por segundo en el pico                   1.900

pérdida estimada al conmutar
  en condiciones normales                  ~80 escrituras
  durante el cierre mensual              ~20.900 escrituras
```

Veinte mil escrituras en el peor momento. Y la pregunta que lo hizo accionable:

```
«si el líder se pierde durante el cierre, ¿cuántos pedidos desaparecen?»
respuesta       entre 400 y 600
respuesta esperada por el negocio      «ninguno»
```

Y la decisión por dato, no por sistema:

```
pedidos y cobros                      asíncrona       SEMISÍNCRONA
                                                      (1 copia síncrona)
sesiones y carritos                   asíncrona       asíncrona
telemetría                            asíncrona       asíncrona

latencia de escritura de pedidos       4,1 ms          6,8 ms
pérdida estimada de pedidos al conmutar  400-600            0
coste                                una réplica más    +310 €/mes
```

Dos coma siete milisegundos más por escritura de pedido, y ninguna pérdida. Para sesiones no se cambió nada porque perder cinco segundos de carrito no cuesta nada.

El caso de los dos líderes, encontrado mirando hacia atrás.

Al revisar un incidente antiguo con este vocabulario:

```
hace 9 meses
  el nodo líder sufrió una pausa larga de recolección de memoria: 47 s
  el sistema lo dio por muerto a los 30 s y promovió la réplica
  a los 47 s el líder viejo despertó y siguió aceptando escrituras
  durante 90 s hasta que se dio cuenta

escrituras aceptadas por el líder viejo tras la promoción       412
escrituras que se perdieron al reincorporarse                   412
cómo se detectó   un descuadre de inventario, 3 días después
cómo se explicó entonces   «un problema puntual de la base»
```

Cuatrocientas doce escrituras desaparecidas sin ningún error, y tres días para notarlo.

Y al comprobar la protección:

```
¿había testigo de época?                      sí, en el motor
¿lo comprobaba el almacenamiento?             sí
¿por qué se perdieron entonces?    las 412 se aceptaron en memoria
                                   y se rechazaron al intentar persistir,
                                   pero la aplicación ya había
                                   respondido «correcto» al cliente
```

Y ahí estaba el fallo real, que no era de la base:

```
la aplicación confirmaba al cliente antes de que la escritura
estuviera confirmada por el almacenamiento
→ nivel de confirmación relajado «por rendimiento»

nivel de confirmación de escritura       relajado      confirmada por la
                                                       mayoría
latencia de escritura                     4,1 ms         6,8 ms
escrituras confirmadas al cliente y
perdidas después                        412 (una vez)        0
ensayo de pausa larga del líder            nunca        semestral
```

Y el ensayo correspondiente, añadido al catálogo de la clase 131:

```
pausar el proceso líder 60 segundos a propósito
  primera ejecución    2 comportamientos incorrectos
    → un servicio reintentaba contra el nodo viejo indefinidamente
    → un trabajo por lotes no detectaba el cambio de líder
  tras corregir        conmutación limpia en 52 s, 0 pérdidas
```

El reparto, y el reequilibrado que no se podía hacer.

El almacén no relacional tenía 24 particiones, asignadas por resto:

```
partición = resumen(id_pedido) mod 24
```

Y al querer pasar a 32 nodos:

```
claves que cambiarían de partición                          97 %
tiempo de migración estimado                          5 semanas
orden por clave durante la migración                  no garantizado
```

Es el mismo problema de la clase 114 en otro sistema. La corrección fue pasar a particiones fijas:

```
particiones lógicas                          24                512
nodos                                         8                  8
particiones por nodo                          3                 64
al añadir un nodo                      migrar el 97 %     mover 57 particiones
tiempo de reequilibrado                 5 semanas           41 min
orden por clave durante el cambio       se rompe           intacto
```

Cuarenta y un minutos frente a cinco semanas, porque la asignación de clave a partición no cambia nunca.

Y el coste de las 512 particiones lógicas, medido:

```
metadatos adicionales                                    +18 MB
latencia añadida                                       no medible
coste                                                       0
```

Los índices secundarios.

```
consulta «pedidos de un cliente»          índice por cliente
implementación actual                     local a cada partición
→ cada consulta preguntaba a las 24 particiones

latencia p50                                             38 ms
latencia p99                                            410 ms  ← la más lenta
coste por consulta                          24 lecturas
```

Y el índice global, con su compromiso:

```
latencia p50                              38 ms               8 ms
latencia p99                             410 ms              21 ms
lecturas por consulta                        24                 1
escrituras por pedido                         1                 2
consistencia del índice                  inmediata      eventual (~200 ms)
```

Y la decisión se tomó con la clasificación de la clase 149:

```
¿qué cuesta que el índice vaya 200 ms por detrás?
  «mis pedidos» → el cliente puede no ver el recién hecho
  → se resuelve con la garantía de sesión ya implantada
→ índice GLOBAL, con lectura del principal durante 5 s tras escribir
```

El consenso, revisado.

```
grupos de consenso en el sistema                              3
  coordinación del almacén no relacional      5 nodos      correcto
  elección de líder de la base                3 nodos      correcto
  bloqueos distribuidos de la aplicación      4 nodos      ← mal
```

El tercero tenía dos problemas:

```
1. cuatro nodos toleran los mismos fallos que tres, y cuestan más
2. los bloqueos se usaban en el camino de las peticiones
   → cada reserva de inventario pedía un bloqueo distribuido
   → +14 ms por operación, y una dependencia más
```

Y la corrección aplicó el corolario del apartado cuarto:

```
                                                      y escritura condicional
latencia añadida por reserva                14 ms                0
dependencia adicional                        sí                  no
comportamiento si el servicio de
bloqueos cae                        no se puede reservar    sigue funcionando
casos de doble reserva                        0                   0
nodos de consenso                             4                   0
```

Y el único uso que se conservó de bloqueo distribuido —un proceso nocturno que no debe ejecutarse dos veces— se dejó con comprobación de época en el destino, no solo con caducidad.

A los seis meses.

```
copia de pedidos                           asíncrona    semisíncrona
pérdida estimada al conmutar              400-600 pedidos     0
nivel de confirmación                      relajado      mayoría
escrituras confirmadas y perdidas         412 (histórico)      0
ensayo de pausa del líder                     nunca        semestral
asignación de partición                  resto módulo N   particiones fijas
tiempo de reequilibrado                    5 semanas        41 min
```

```
índice secundario                          local        global
latencia p99 de «mis pedidos»             410 ms         21 ms
grupos de consenso                           3              2
bloqueos distribuidos en el camino
de peticiones                               sí             no
```

La lección que esta clase traslada a la parte 12: la base anunciaba «pérdida mínima» y perdía entre cuatrocientos y seiscientos pedidos en el peor momento del mes, y nadie lo sabía porque nadie había multiplicado el retardo de replicación por las escrituras por segundo. Y el incidente de hace nueve meses —cuatrocientas doce escrituras confirmadas al cliente y desaparecidas después— no lo causó la base: lo causó una aplicación que respondía «correcto» antes de que el almacenamiento lo hubiera confirmado, con la protección contra dos líderes funcionando perfectamente por debajo.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-12-cloud-native-distributed-architecture/150-replicacion-particionado-y-consenso/
lab.py
```

El laboratorio selecciona el motor de práctica distributed 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-replicacion 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 traza de consistencia, reintento o fallo parcial. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye modelo-replicacion 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 réplicas y la capacidad de escritura no mejora	Se confunde replicar con repartir	Replicar da disponibilidad y lectura; para escribir más hace falta particionar.
Al conmutar desaparecen datos que el cliente creía guardados	Copia asíncrona y confirmación al cliente antes de que el almacenamiento confirme	Mide el punto de recuperación multiplicando retardo por escrituras; usa copia semisíncrona y confirmación por mayoría para lo que no se puede perder.
Dos nodos aceptan escrituras a la vez tras una pausa larga	El líder viejo no sabe que fue reemplazado	Testigo de época comprobado por quien persiste, y ensaya pausando el líder a propósito.
Añadir un nodo obliga a migrar casi todos los datos	La partición se calcula como resto del número de nodos	Usa particiones fijas en número mucho mayor que los nodos, o resumen consistente.
Una consulta por índice secundario tiene un percentil 99 pésimo	El índice es local y hay que preguntar a todas las particiones	Valora un índice global particionado por el campo consultado, aceptando que irá eventualmente por detrás.
Cada operación de negocio necesita un acuerdo entre nodos	El consenso está en el camino de las peticiones	Usa consenso solo para metadatos; sustituye los bloqueos distribuidos por idempotencia y escritura condicional.

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 da replicar y qué da repartir, y qué no da cada uno?
2. ¿Cómo se estima cuántos datos se pierden en una conmutación asíncrona?
3. ¿Cómo impide un testigo de época que dos nodos actúen como líder?
4. ¿Por qué calcular la partición como resto del número de nodos es un error?
5. ¿Para qué debe usarse el consenso y para qué no?

Referencias

- Kleppmann, M. (2017). Designing Data-Intensive Applications, caps. 5, 6 y 9 — replicación, particionado y consenso. <https://dataintensive.net/>
- Ongaro, D. y Ousterhout, J. (2014). In search of an understandable consensus algorithm (Raft) — elección de líder y mayorías. <https://raft.github.io/raft.pdf>
- Kleppmann, M. (2016). How to do distributed locking — por qué un bloqueo con caducidad necesita testigo comprobado en el destino. <https://martin.kleppmann.com/2016/02/08/how-to-do-distributed-locking.html>
- Karger, D. y otros (1997). Consistent hashing and random trees — reequilibrado sin recalcular la asignación. <https://dl.acm.org/doi/10.1145/258533.258660>
- DeCandia, G. y otros (2007). Dynamo: Amazon's highly available key-value store — quórum, reparación y sus límites. <https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 12 en PDF · Manual integral

Evaluación — 150 Replicación, particionado y consenso

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-replicacion 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.

151 — Fallos parciales y patrones de resiliencia

Parte: 12 — Arquitectura cloud-native y sistemas distribuidos Nivel: avanzado-experto · Horas estimadas: 4
Laboratorio: reliability · Estado: EXECUTABLECORE

Propósito

Entender por qué un sistema repartido falla de formas que un programa en una sola máquina no conoce, y qué se hace a nivel de sistema —no de llamada, que fue la clase 130—. La clase parte de una imposibilidad que lo explica casi todo: no se puede distinguir lento de muerto de respuesta perdida, así que todo plazo es una conjetura. De ahí salen el modo de fallo más traicionero, que es el nodo que se cree sano mientras sus clientes fallan; el fallo que se sostiene solo después de que la causa desaparezca; y los dos patrones que de verdad acotan el daño: hacer siempre el mismo trabajo y dividir el sistema en celdas.

Resultados de aprendizaje

Al finalizar podrás:

1. Enumerar los cuatro resultados posibles de una llamada remota.
2. Reconocer el fallo gris y diseñar comprobaciones de salud que no mientan.
3. Evitar que un fallo se sostenga solo tras desaparecer la causa.
4. Aplicar trabajo constante y contrapresión de extremo a extremo.
5. Acotar el radio de un fallo con celdas y reparto por sorteo.

Conceptos centrales

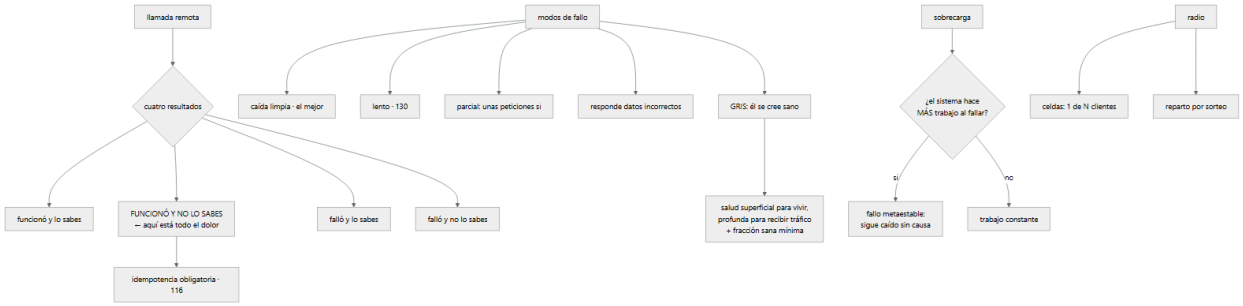
Concepto	Comprensión verificable
indistinguibilidad	No hay forma de saber si el otro extremo está lento, muerto o si la respuesta se perdió. Todo plazo es una conjetura.
fallo gris	El componente se considera sano a sí mismo y sus clientes no lo ven así. Es el modo más difícil de detectar y el que más dura.
fallo metaestable	El sistema sigue caído después de que la causa desaparezca, porque el propio trabajo acumulado lo sostiene.
trabajo constante	Diseñar para hacer siempre la misma cantidad de trabajo, falle o no algo. Elimina los picos de carga provocados por fallos.
contrapresión	Que la saturación se propague hacia atrás rechazando trabajo, en vez de absorberlo en colas sin límite.
celda	Copia completa e independiente del sistema que atiende a una parte de los clientes. Convierte una caída total en una parcial.
dependencia blanda	Aquella sin la cual el sistema sigue sirviendo, degradado. Solo lo es si se ha comprobado.
reparto por sorteo	Asignar a cada cliente un subconjunto distinto de recursos, de modo que dos clientes rara vez compartan todos.
fracción sana mínima	Límite que impide retirar del servicio a más de una parte de las instancias, aunque todas parezcan enfermas.

Modelo mental

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

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 hace distinto un fallo repartido

En un solo proceso, una llamada funciona o lanza un error. En cuanto hay red, hay cuatro resultados:

```
1. funcionó y lo sabes
2. FUNCIONÓ y NO lo sabes          ← la respuesta se perdió
3. falló y lo sabes
4. falló y no lo sabes             ← el plazo venció y sigue ejecutándose
```

Y el segundo y el cuarto son indistinguibles desde fuera. De ahí la imposibilidad que gobierna esta materia:

```
no se puede distinguir
  «está muerto»  de  «está lento»  de  «la respuesta se perdió»
→ todo plazo es una CONJETURA sobre cuál de los tres es
```

Y sus consecuencias, que este programa ya ha ido encontrando:

```
reintentar puede duplicar el efecto       → idempotencia    clase 116
conmutar puede dejar dos líderes          → testigo         clase 150
la entrega exactamente una vez no existe  →                 clase 113
y un plazo demasiado corto convierte
  éxitos lentos en fallos                 →                 clase 130
```

Los modos de fallo, ordenados de más benigno a más dañino:

```
CAÍDA LIMPIA        el proceso desaparece
  → el mejor caso: se detecta rápido y todo el mundo reacciona

LENTITUD            responde, tarde
  → peor que caerse: consume recursos de quien llama    clase 130

FALLO PARCIAL       una parte de las peticiones falla y otra no
  → los promedios lo esconden; hace falta percentil y desglose

RESPUESTAS INCORRECTAS   contesta rápido y mal
  → ninguna comprobación de salud lo detecta
  → se detecta validando lo que llega                   clase 131

FALLO GRIS          él se considera sano y sus clientes no
  → el más difícil, y el que más dura
```

Y una regla que se deduce de la lista: preferir caerse a degradarse en silencio. Un proceso que detecta que no puede cumplir su función debería terminar, no seguir aceptando trabajo que no va a hacer bien.

2. El fallo gris y las comprobaciones que mienten

El caso característico:

```
la comprobación de salud responde correcto
el panel está verde
y el 30 % de las peticiones reales falla
```

Y la causa es siempre la misma: la comprobación no hace lo que hace el tráfico real.

```
comprobación   devuelve «correcto» sin tocar nada
tráfico real   consulta la base, llama a dos servicios y escribe en una cola
→ si la base rechaza conexiones, la comprobación sigue diciendo correcto
```

Y las variantes concretas que producen fallo gris:

```
agotamiento de recursos en un nodo: descriptores, memoria, hilos
un disco que responde muy lento pero responde
pérdida de paquetes del 2 % en un enlace concreto
una zona con latencia degradada
un nodo con el reloj desviado                          clase 149
y la comprobación que se ejecuta en un hilo distinto del que está bloqueado
```

La última es especialmente traicionera: el proceso está bloqueado para el tráfico y sano para la comprobación.

La corrección tiene dos mitades y hay que hacer las dos:

```
1. QUE LA COMPROBACIÓN SE PAREZCA AL TRÁFICO
   vivo      superficial: ¿el proceso responde? → si no, reiniciar
   listo     profunda: ¿puedo atender de verdad? → si no, sacar del reparto
   → y la profunda usa la misma ruta que el tráfico: mismo grupo de hilos,
     misma conexión, mismas dependencias imprescindibles

2. QUE UNA DEPENDENCIA CAÍDA NO SAQUE A TODAS LAS INSTANCIAS
   si la comprobación profunda depende de la base y la base falla,
   las 40 instancias se declaran no listas a la vez
   → y entonces no queda nadie para servir lo que SÍ funcionaba
```

Y el mecanismo que resuelve la segunda es imprescindible y poco conocido:

```
FRACCIÓN SANA MÍNIMA
  el reparto nunca saca de servicio más de un porcentaje de instancias
  si todas parecen enfermas, se asume que el problema es de la
  comprobación o de una dependencia común, y se sigue enviando tráfico
→ mejor un sistema degradado que ninguno
```

Y la otra defensa, que viene de la clase 121: medir desde fuera. La salud real es la que ven los clientes, no la que declara el propio nodo.

```
salud declarada por el nodo          útil para reiniciar y para el reparto
proporción de errores observada
  por sus clientes                   ← esta es la verdad
```

3. El fallo que se sostiene solo

Hay una clase de caída que sorprende siempre: la causa desaparece y el sistema sigue caído.

```
09:12  una dependencia se degrada 4 minutos
09:16  la dependencia se recupera
09:52  el sistema sigue sin servir
```

Y el motivo es que el propio sistema genera el trabajo que lo mantiene caído:

```
las peticiones se acumulan en colas sin límite
los clientes reintentan, y cada reintento añade carga
los plazos vencen, y el trabajo hecho se descarta
→ se trabaja mucho y no se completa nada
→ y el sistema tiene DOS estados estables: sano y atascado
```

Lo que lo produce, en una frase: el sistema hace MÁS trabajo cuando las cosas van mal.

Y las tres defensas:

```
1. COLAS ACOTADAS Y DESCARTE                           clase 130
   rechazar deprisa en vez de aceptar y no poder atender
   y descartar lo más antiguo, que probablemente ya no lo espera nadie

2. PRESUPUESTO DE REINTENTOS
   los reintentos no pueden superar un porcentaje del tráfico normal
   → se cortan solos justo cuando más daño harían

3. TRABAJO CONSTANTE
   diseñar para hacer siempre la misma cantidad de trabajo
```

El tercero es el más potente y el menos intuitivo:

```
mal   cuando algo cambia, se envía el cambio a quien corresponda
      → un fallo masivo produce muchos cambios a la vez
      → y el pico de trabajo llega cuando el sistema está peor

bien  cada N segundos se envía el estado COMPLETO, cambie o no
```

```
      → la carga es la misma un martes tranquilo que durante una caída
      → y no hay nada que reintentar: en el siguiente ciclo se corrige
```

Y sus aplicaciones concretas:

```
distribuir configuración: enviar el fichero entero periódicamente
listas de instancias sanas: publicar la lista completa
reconciliación: comparar el estado deseado con el real   clase 103
cachés: refrescar por ciclo en vez de por invalidación   clase 111
```

Y el precio, que hay que aceptar: se hace trabajo inútil el 99 % del tiempo. A cambio, el sistema se comporta igual en el peor día que en el mejor, y eso vale más de lo que cuesta.

La contrapresión es la versión de extremo a extremo del mismo principio:

```
si un componente no da abasto, debe RECHAZAR, no encolar
y quien le llama debe enterarse y a su vez rechazar hacia atrás
→ hasta llegar al borde, que devuelve un error rápido al cliente

si en algún punto de la cadena hay una cola sin límite,
la contrapresión se corta ahí y el sistema se atasca
```

4. Acotar el radio

Todo lo anterior reduce la probabilidad de fallar. Lo que reduce el daño cuando se falla igualmente son dos patrones.

Celdas. En lugar de un sistema grande, varias copias completas e independientes:

```
sin celdas   un sistema; un fallo grave afecta al 100 % de los clientes
con celdas   8 celdas; cada cliente vive en una
             → un fallo grave afecta al 12,5 %
```

Y sus propiedades:

```
cada celda tiene su base, su caché y sus instancias
nada se comparte entre celdas, salvo el enrutador que asigna
un despliegue se hace celda a celda: es un canario real  clase 102
y el tamaño de la celda se elige por lo que se puede probar,
  no por lo que cabe
```

Y el coste, que es real:

```
menos aprovechamiento: cada celda tiene su margen
operar N copias en vez de una
y el enrutador pasa a ser el punto crítico
  → debe ser lo más simple posible, con trabajo constante
```

Y la pregunta que decide si compensan:

```
¿cuánto cuesta una caída total frente a una caída del 12,5 %?
→ en un sistema con muchos clientes y un objetivo alto, mucho
→ en un sistema pequeño, no compensa
```

Reparto por sorteo. El truco barato que consigue algo parecido sin duplicar nada:

```
8 nodos; cada cliente se asigna a 2 de ellos, elegidos por su identificador

probabilidad de que dos clientes compartan LOS DOS nodos
  = 1 / combinaciones de 8 tomadas de 2 = 1/28

→ un cliente que satura sus dos nodos afecta al 3,6 % de los demás
→ frente al 100 % si todos comparten los 8
```

Y con más nodos y subconjuntos pequeños, el aislamiento mejora deprisa. Es una de las mejores relaciones entre coste y efecto que existen, y solo requiere cambiar cómo se asignan los clientes.

Y la clasificación de dependencias, que hay que escribir y comprobar:

```
DURA     sin ella el sistema no puede cumplir su función
         → su disponibilidad multiplica a la tuya       clase 126
BLANDA   sin ella el sistema sirve, degradado
         → y solo es blanda si se ha COMPROBADO         clase 131
```

Y lo que ocurre siempre al comprobarlo la primera vez:

```
dependencias declaradas como blandas          n
dependencias que de verdad lo son             menos que n
```

Y la lista de comprobación de la clase:

- toda operación remota tolera «funcionó y no lo sé»
- las comprobaciones de salud usan la misma ruta que el tráfico
- vivo y listo están separados y significan cosas distintas
- existe fracción sana mínima: nunca se retira todo el servicio
- la salud real se mide también desde fuera
- ninguna cola de la cadena carece de límite
- hay presupuesto de reintentos, no solo límite por llamada
- lo crítico usa trabajo constante en vez de notificar cambios
- la contrapresión llega hasta el borde sin cortarse
- está escrito qué dependencias son duras y cuáles blandas
- las blandas se han comprobado provocando su caída
- está decidido si se acota el radio con celdas o con reparto por sorteo

Y el cierre que enlaza con la clase siguiente: para que todo esto funcione, cada servicio tiene que encontrar a los demás, saber cuáles están sanos y hablar con ellos de forma segura. Quién resuelve eso —la aplicación, la plataforma o una malla— y qué cuesta cada opción es la materia de la clase 152.

Ejemplo trabajado

CloudShop sufre dos incidentes que sus mecanismos de la clase 130 no evitaron: uno en el que todo estaba verde mientras fallaba un tercio del tráfico, y otro en el que el sistema siguió caído treinta y seis minutos después de que la causa desapareciera.

Incidente 1: el fallo gris.

```
14:02  el 31 % de las peticiones al catálogo empieza a fallar
14:02  las comprobaciones de salud: 40 de 40 correctas
14:02  el reparto sigue enviando tráfico a las 40 instancias
14:09  alerta de proporción de errores (clase 125)
14:31  se identifica: 12 de las 40 instancias tienen agotados
       los descriptores de fichero
14:34  se reinician esas 12
duración                                          32 min
```

Y el diagnóstico:

```
comprobación de salud     devolvía 200 sin tocar nada
tráfico real              abría conexiones a la base y al caché
→ una instancia sin descriptores respondía la comprobación
  y fallaba todo lo demás
```

La corrección, con las dos mitades del apartado segundo:

```
comprobación de vivo         devuelve 200      devuelve 200 (igual)
comprobación de listo        no existía        consulta la base y el caché
                                               por la MISMA ruta y con
                                               el mismo grupo de hilos

tiempo hasta sacar del reparto una
instancia enferma            no ocurría        9 s
```

Y el primer día tras activarla, ocurrió justo lo que el apartado advierte:

```
03:40  la base tiene un pico de latencia de 12 s
03:40  las 40 instancias fallan la comprobación profunda
03:40  el reparto las saca TODAS
03:40  caída total, peor que el problema original
duración                                           6 min

fracción sana mínima                          no había        50 %
qué pasa si todas fallan la comprobación   se sacan todas   se conservan
                                                           la mitad


ensayo del mismo pico, repetido
  instancias retiradas                          40             20
  peticiones servidas                            0 %           58 %
  duración de la degradación                  6 min          90 s
```

Y la tercera pieza, de la clase 121: medir la salud desde fuera.

```
nueva alerta   proporción de errores observada POR LOS CLIENTES
               de cada instancia, no declarada por ella
tiempo de detección del mismo caso, en un ensayo
  con salud declarada                           no se detecta
```

```
  con salud observada desde fuera                    70 s
```

Incidente 2: el fallo que se sostuvo solo.

```
09:12  el servicio de precios se degrada por un despliegue
09:16  se revierte; precios vuelve a funcionar
09:16  el flujo de compra sigue sin servir
09:24  se añaden instancias; no mejora
09:52  se paran los consumidores y se vacían las colas a mano
09:54  el sistema se recupera

duración de la causa                                4 min
duración del incidente                             42 min
```

Y la reconstrucción con el vocabulario del apartado tercero:

```
al degradarse precios, las peticiones tardaban más
→ la cola de entrada de pedidos creció sin límite
→ los clientes reintentaban: +3,4× de tráfico
→ los plazos vencían y el trabajo hecho se descartaba
→ el sistema trabajaba al 100 % y completaba el 6 %
y al recuperarse precios, la cola acumulada mantenía el estado
```

```
cola de entrada                          sin límite      800, con descarte
qué se descarta                                —         lo más antiguo
presupuesto de reintentos                no había          10 %

ensayo: degradar precios 4 minutos a propósito
  duración del incidente                   42 min          4 min 40 s
  peticiones servidas durante la degradación   6 %            71 %
  tiempo de recuperación tras restaurar    36 min            25 s
```

El trabajo constante, aplicado a dos sitios.

```
CASO 1: distribución de configuración
  antes   al cambiar algo, se notificaba a los 15 servicios
          → un cambio masivo generaba 15 × N notificaciones
          → y durante un incidente, los reintentos se acumulaban
  después cada servicio pide la configuración COMPLETA cada 30 s
          → carga idéntica siempre; nada que reintentar

  carga en un día normal           +2 %
  carga durante un incidente       de ×18 a ×1

CASO 2: lista de instancias sanas
  antes   se publicaban altas y bajas
          → una caída de zona generaba un aluvión de bajas
  después se publica la lista completa cada 5 s
          → una caída de zona no genera ningún pico
```

Y el coste, medido y aceptado:

```
tráfico adicional en régimen normal                 +1,4 %
latencia de propagación de un cambio           de ~1 s a ~15 s
decisión   se acepta: ningún cambio de configuración necesita
           propagarse en menos de 30 s
```

Las celdas, evaluadas y descartadas; el sorteo, adoptado.

```
evaluación de celdas
  clientes                                          190 empresas
  objetivo del flujo de compra                      99,5 %
  coste de 8 celdas                  +40 % de infraestructura
  beneficio                     una caída afecta al 12,5 % en vez del 100 %
  decisión                      NO por ahora; se revisará si el objetivo
                                sube o si los clientes crecen mucho
```

Y en su lugar, el reparto por sorteo, que costó una tarde:

```
nodos del servicio de API                   12              12
nodos por cliente                           12               2
probabilidad de compartir ambos nodos      100 %          1/66 = 1,5 %

ensayo: un cliente satura sus nodos con 40× su tráfico habitual
  clientes afectados, antes                 190
  clientes afectados, después                 3
```

De ciento noventa clientes afectados a tres, sin añadir un solo nodo.

Las dependencias, clasificadas y comprobadas.

```
declaradas como blandas                                    6
comprobadas provocando su caída (clase 131)                6
resultaron ser blandas de verdad                           4
resultaron DURAS sin que nadie lo supiera                  2
  → el servicio de precios: sin él no se podía mostrar el catálogo
  → el de identidad: sin él no se podía ni navegar sin sesión
```

Y las dos se convirtieron en blandas de verdad:

```
precios      caché con valor caducado servible y precio base
identidad    navegación anónima sin llamar a identidad

disponibilidad del flujo de compra
  techo por dependencias antes                        99,05 %
  techo después                                       99,74 %
```

A los seis meses.

```
comprobación de listo separada de vivo          no             sí
comprobación que usa la ruta real               no             sí
fracción sana mínima                            no            50 %
salud medida desde fuera                        no             sí
tiempo de detección de un fallo gris          32 min          70 s
colas sin límite                              3 de 7        0 de 7
presupuesto de reintentos                       no            10 %
duración de un incidente con causa de 4 min   42 min       4 min 40 s
componentes con trabajo constante               0              2
clientes afectados por un vecino ruidoso      190              3
dependencias declaradas blandas                 6              6
de ellas, comprobadas                           0              6
de ellas, que lo eran de verdad                 4              6
```

La lección que esta clase traslada a la parte 12: los dos incidentes ocurrieron con todos los mecanismos de la clase 130 correctamente configurados, porque ninguno de los dos era un problema de la llamada. El primero fue una comprobación de salud que no hacía lo que hace el tráfico, y su primera corrección produjo una caída peor por retirar las cuarenta instancias a la vez. El segundo fue un sistema que hacía más trabajo cuanto peor iba, y que por eso siguió caído treinta y seis minutos después de que la causa desapareciera. Y el cambio con mejor relación entre coste y efecto no fue ninguno de los dos: asignar a cada cliente dos nodos de doce en vez de los doce, que redujo el alcance de un vecino ruidoso de ciento noventa clientes a tres y no costó nada.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-12-cloud-native-distributed-architecture/151-fallos-parciales-y-patrones-de-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 mapa-fallos 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 mapa-fallos para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

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

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
El panel está verde y una parte importante del tráfico falla	Fallo gris: la comprobación de salud no hace lo que hace el tráfico real	Separa vivo de listo, haz que la comprobación de listo use la misma ruta y grupo de hilos, y mide además la salud observada por los clientes.
Una dependencia con un pico deja el servicio entero fuera del reparto	Todas las instancias fallan la comprobación profunda a la vez	Fracción sana mínima: nunca retirar más de un porcentaje; mejor degradado que sin nadie.
El sistema sigue caído después de que la causa desaparezca	Fallo metaestable: colas sin límite y reintentos crean trabajo que se sostiene solo	Acota todas las colas con descarte, añade presupuesto de reintentos y reduce el trabajo bajo estrés en vez de aumentarlo.
Un fallo masivo genera un pico de trabajo justo cuando peor va todo	El diseño notifica cambios en vez de publicar estado completo	Trabajo constante: publica el estado completo con periodicidad fija, cambie o no.
Un cliente que abusa afecta a todos los demás	Todos comparten los mismos recursos	Reparto por sorteo: asigna a cada cliente un subconjunto pequeño y distinto; valora celdas si el objetivo lo justifica.
Una dependencia declarada opcional tumba el sistema	Nunca se comprobó que fuera opcional	Escribe qué dependencias son duras y cuáles blandas, y comprueba cada blanda provocando su caída.

■ Seguridad, ética y costo

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

■ Preguntas de comprobación

1. ¿Cuáles son los cuatro resultados de una llamada remota y cuál causa más problemas?
2. ¿Qué es un fallo gris y por qué las comprobaciones de salud no lo detectan?
3. ¿Por qué una comprobación de salud profunda puede provocar una caída total, y qué lo impide?
4. ¿Qué mantiene caído a un sistema cuando la causa ya desapareció?
5. ¿Qué consigue el reparto por sorteo y a qué coste?

Referencias

- Huang, P. y otros (2017). Gray failure: the Achilles' heel of cloud-scale systems — el fallo que el componente no reconoce. <https://www.microsoft.com/en-us/research/publication/gray-failure-the-achilles-heel-of-cloud-scale-systems/>
- Bronson, N. y otros (2021). Metastable failures in distributed systems — fallos que se sostienen tras desaparecer la causa. <https://sigops.org/s/conferences/hotos/2021/papers/hotos21-s11-bronson.pdf>
- AWS (2025). Reliability and constant work — diseñar para hacer siempre la misma cantidad de trabajo. <https://aws.amazon.com/builders-library/reliability-and-constant-work/>
- AWS (2025). Workload isolation using shuffle sharding — reparto por sorteo y su aritmética. <https://aws.amazon.com/builders-library/workload-isolation-using-shuffle-sharding/>
- AWS (2025). Implementing health checks — vivo frente a listo y fracción sana mínima. <https://aws.amazon.com/builders-library/implementing-health-checks/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 12 en PDF · Manual integral

Anterior	Índice	Siguiente
← 150 · Replicación, particionado y consenso	Parte 12 · Programa	152 · Service discovery, malla y comunicación →

Evaluación — 151 Fallos parciales y patrones de 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 mapa-fallos 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.

152 — Service discovery, malla y comunicación

Parte: 12 — Arquitectura cloud-native y sistemas distribuidos Nivel: avanzado-experto · Horas estimadas: 4
Laboratorio: network · Estado: EXECUTABLECORE

Propósito

Resolver cómo se encuentran y se hablan las piezas cuando las direcciones cambian cada pocos minutos: quién traduce «el servicio de pagos» en una dirección concreta, quién decide qué instancias están sanas y dónde vive la política de reintentos, plazos y cifrado. La clase compara los tres sitios donde puede vivir esa lógica, dice qué aporta de verdad una malla y qué no, y se detiene en dos detalles que causan incidentes reales y casi nunca se explican: el reparto rotatorio manda trabajo a la instancia lenta, y una conexión multiplexada anula el reparto.

Resultados de aprendizaje

Al finalizar podrás:

1. Situar el descubrimiento y la política en la biblioteca, la plataforma o un intermediario.
2. Decidir si una malla compensa, con un criterio numérico.
3. Enumerar lo que una malla no resuelve aunque lo parezca.
4. Elegir el algoritmo de reparto sabiendo qué hace cada uno con un nodo lento.
5. Evitar que las conexiones persistentes rompan el reparto.

Conceptos centrales

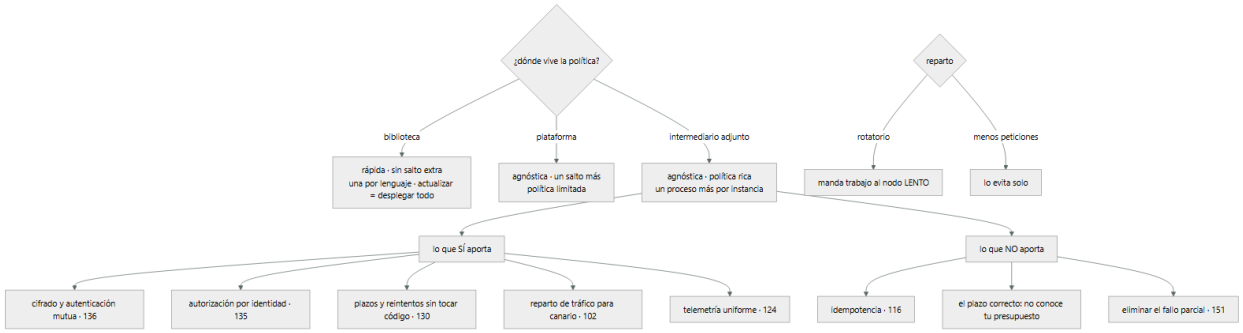
Concepto	Comprensión verificable
descubrimiento	Traducir un nombre lógico en direcciones concretas y sanas. En entornos dinámicos cambia constantemente.
intermediario adjunto	Proceso que acompaña a cada instancia e intercepta su tráfico para aplicar política sin tocar el código.
identidad de carga en el transporte	Cada servicio tiene un certificado propio; la autenticación mutua cifra y a la vez dice quién llama.
reparto por menos peticiones	Enviar a la instancia con menos peticiones en curso. Evita mandar trabajo a la que está lenta.
multiplexación	Varias peticiones sobre una misma conexión. Reduce coste y hace que el reparto por conexión deje de repartir.
conversación excesiva	Muchas llamadas pequeñas para una sola operación de negocio. Es el problema de rendimiento más común entre servicios.

Modelo mental

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

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 sitios donde puede vivir

El problema es el mismo en los tres casos:

```
«llamar al servicio de precios» exige saber
  qué direcciones lo sirven ahora
  cuáles están sanas                                  clase 151
  a cuál enviar esta petición
  con qué plazo, reintentos y cifrado                 clases 130, 136
```

Y hay tres lugares donde puede resolverse:

```
1. EN LA BIBLIOTECA DEL CLIENTE
   la aplicación consulta el registro y elige destino
   + sin saltos extra; latencia mínima
   + puede usar información que solo ella tiene
   – una biblioteca por lenguaje
   – actualizarla exige desplegar TODOS los servicios
   – y una versión antigua se comporta distinto que una nueva

2. EN LA PLATAFORMA
   un nombre estable y un repartidor por delante
   + independiente del lenguaje; nada que instalar
   + es lo que ya hace un servicio de la clase 076
   – un salto de red más
   – política limitada a lo que ofrezca el repartidor

3. EN UN INTERMEDIARIO ADJUNTO A CADA INSTANCIA
   un proceso acompaña a la aplicación e intercepta su tráfico
   + independiente del lenguaje y con política rica
   + se cambia sin tocar ni desplegar la aplicación
   – un proceso más por instancia: memoria, procesador y latencia
   – un plano de control más que operar
   – y un modo de fallo nuevo: el intermediario
```

Y existe una cuarta forma en crecimiento —aplicar la política en el núcleo del sistema operativo, sin proceso adjunto— que reduce el coste del tercero y no cambia el razonamiento de esta clase.

Y el criterio para elegir, que es numérico:

```
pocos servicios y un solo lenguaje
  → biblioteca o plataforma; una malla cuesta más de lo que da

muchos servicios y varios lenguajes, con política común obligatoria
  → intermediario adjunto

umbral orientativo   por debajo de ~10 servicios, casi nunca compensa
                     por encima de ~30 y con 3 lenguajes, casi siempre
```

Y una advertencia sobre el segundo caso, que aparece mucho: si todo está en un lenguaje, una biblioteca bien hecha da el 80 % de lo que da una malla, sin plano de control. El problema empieza cuando hay que actualizarla en cuarenta servicios.

2. Qué aporta una malla y qué no

Lo que aporta de verdad, ordenado por lo que más se nota:

```
CIFRADO Y AUTENTICACIÓN MUTUA ENTRE SERVICIOS
  cada servicio tiene su certificado, rotado solo
  → resuelve el cifrado interno que la clase 136 encontró ausente
  → y de paso da identidad verificable en el transporte

AUTORIZACIÓN POR IDENTIDAD, NO POR DIRECCIÓN
  «pedidos puede llamar a precios»                      clase 135
  → sobrevive a que todo se recree

PLAZOS, REINTENTOS Y CORTE SIN TOCAR CÓDIGO
  y aplicables de forma uniforme                        clase 130

REPARTO DE TRÁFICO POR PORCENTAJE
  canarios y despliegues escalonados sin lógica propia  clase 102

TELEMETRÍA UNIFORME
  métricas y trazas de todas las llamadas, con el mismo formato
                                                        clase 124
```

Y la lista es real: cinco cosas que de otro modo hay que implementar en cada lenguaje.

Lo que NO aporta, y se le atribuye:

```
NO HACE SEGURO REINTENTAR
  el intermediario puede reintentar; que eso no duplique un cobro
  es responsabilidad de la aplicación                   clase 116
  → y una malla mal configurada AÑADE reintentos que nadie pidió

NO SABE CUÁL ES EL PLAZO CORRECTO
  no conoce el presupuesto de la petición original      clase 130
  → hay que configurárselo, y propagar el plazo restante sigue
    siendo trabajo de la aplicación

NO ELIMINA EL FALLO PARCIAL
  todo lo de la clase 151 sigue igual

NO ARREGLA UN DISEÑO CONVERSADOR
  si una operación hace 340 llamadas, las hará igual
  → y ahora cada una pasa por dos intermediarios más

NO SUSTITUYE A LOS CONTRATOS
  la compatibilidad entre versiones es de la clase 153
```

Y el coste, que hay que medir antes de decidir:

```
latencia añadida     0,3-2 ms por salto, y hay dos saltos por llamada
memoria              50-150 MB por instancia
procesador           5-15 % del total
plano de control     un sistema más que operar y actualizar
modo de fallo nuevo  si el intermediario no arranca, la aplicación
                     no habla con nadie
```

Y una cuenta que conviene hacer:

```
300 instancias × 100 MB = 30 GB de memoria solo en intermediarios
```

Y dos precauciones operativas que se descubren tarde:

```
el intermediario debe arrancar ANTES que la aplicación
  y terminar DESPUÉS                                    clase 146
  → si no, hay peticiones al arrancar y al parar que no salen

las actualizaciones del plano de control son un despliegue
  que afecta a todo a la vez
  → escalonado, con canario, como cualquier otro        clase 102
```

3. Cómo se reparte, y las dos trampas

Los algoritmos, y lo que hace cada uno cuando una instancia va mal:

```
ROTATORIO
  una a cada instancia, por turnos
  + trivial y justo si todas son iguales
  − MANDA TRABAJO A LA QUE ESTÁ LENTA, exactamente igual que a las sanas
  → y si una instancia tarda 10× más, acumula 10× más trabajo pendiente
```

```
MENOS PETICIONES EN CURSO
  a la que tiene menos trabajo sin terminar
  + una instancia lenta acumula peticiones y deja de recibir sola
  → resuelve el problema anterior sin configurar nada

POR LATENCIA OBSERVADA
  pondera por lo que ha tardado últimamente
  + reacciona a nodos degradados
  − puede oscilar si el peso cambia demasiado deprisa

POR CLAVE
  la misma clave siempre al mismo destino
  + aprovecha cachés locales
  − puntos calientes, y hay que rehacerlo al cambiar el número
    de destinos                                          clase 150
```

Y la primera trampa, en una frase:

```
el reparto rotatorio es la opción por defecto
y es la peor ante una instancia lenta, que es el fallo más común
```

Y una técnica barata que mejora cualquiera de ellos: elegir dos al azar y quedarse con la menos cargada. Casi todo el beneficio de «menos peticiones» sin necesidad de conocer el estado de todas.

La segunda trampa: las conexiones persistentes.

```
con una conexión por petición
  el reparto elige destino en cada llamada          → reparte bien

con conexiones persistentes y multiplexadas
  se abre UNA conexión y todas las peticiones van por ella
  → el reparto eligió destino una vez, al abrirla
  → y desde entonces, todo va al mismo sitio
```

Y el efecto observado:

```
12 instancias de destino
40 instancias de origen, con una conexión persistente cada una
→ si el reparto fue rotatorio al abrir, quedan 3-4 conexiones
  por destino, y el reparto real depende de qué origen tiene tráfico
→ y al añadir una instancia de destino, NO recibe nada:
  nadie abre conexiones nuevas
```

Las correcciones:

```
reparto por PETICIÓN, no por conexión
  → es lo que hace un intermediario que entiende el protocolo
caducidad de conexión: renovarlas cada N minutos o N peticiones
  → para que las instancias nuevas entren en el reparto
y un número mínimo de conexiones por destino
```

Y el mismo problema, en su versión más citada: una instancia nueva no recibe tráfico hasta que alguien abre una conexión, y eso hace que el autoescalado parezca no funcionar.

Y una tercera cuestión relacionada: el arranque en frío del destino. Una instancia recién arrancada acepta la misma carga que las demás y aún no tiene sus cachés llenas ni su código optimizado. Un arranque progresivo —darle poco tráfico al principio e ir subiendo— evita ese pico de latencia.

4. Cómo se habla

Síncrono o asíncrono se decide por interacción, con los atributos de la clase 145:

```
SÍNCRONO      quien llama necesita la respuesta para continuar
  precio de un producto, validar un dato, comprobar existencias

ASÍNCRONO     el resultado no se necesita ahora
  notificar, facturar, indexar, recalcular            parte 09
```

Y la regla que más latencia ahorra:

```
si puede ser asíncrono, que lo sea
→ cada llamada síncrona añade su latencia y su probabilidad de fallo
  a la petición del usuario                            clase 126
```

La granularidad, que es el problema de rendimiento más frecuente entre servicios:

```
conversación excesiva
  «dame el pedido» → «dame cada línea» × 40 → «dame cada producto» × 40
  → 81 llamadas de red para una pantalla
  → es la escalera de la clase 124, ahora entre servicios

corrección
  operaciones que devuelven lo que la pantalla necesita
  o consultas por lotes: «dame estos 40 productos»
  y evitar que cada consumidor tenga su propia operación a medida
```

Y la tensión que hay que resolver conscientemente: operaciones a medida para cada consumidor acoplan al proveedor con sus clientes; operaciones genéricas producen conversación excesiva. La salida habitual es una capa de composición del lado del consumidor —una fachada por tipo de cliente— que agrega en un solo salto.

Los protocolos, con honestidad:

```
TEXTO SOBRE HTTP     universal, depurable, verboso
                     suficiente para la mayoría
BINARIO CON ESQUEMA  menos bytes y menos tiempo de serialización
                     necesita generación de código y registro de esquemas
                     → y entonces el contrato es explícito, que es bueno
                                                    clase 153
FLUJO CONTINUO       útil cuando hay muchos mensajes en una conversación
                     o resultados que llegan poco a poco
```

Y la nota honesta: el protocolo rara vez es el cuello de botella. Antes de cambiarlo, conviene comprobar dónde se va el tiempo con una traza; casi siempre está en el número de llamadas, en la base o en una dependencia lenta, no en la serialización.

Y la lista de comprobación de la clase:

- ■ está decidido dónde vive la política: biblioteca, plataforma o adjunto
- ■ si hay malla, está justificada por número de servicios y lenguajes
- ■ está medido lo que cuesta: latencia, memoria y procesador
- ■ el intermediario arranca antes y termina después que la aplicación
- ■ los reintentos de la malla no duplican efectos: hay idempotencia
- ■ los plazos configurados respetan el presupuesto de la petición
- ■ el reparto no es rotatorio si las instancias pueden ir lentas
- ■ el reparto es por petición, no por conexión
- ■ las conexiones caducan para que entren las instancias nuevas
- ■ las instancias nuevas reciben tráfico progresivamente
- ■ cada interacción está clasificada como síncrona o asíncrona
- ■ está medido cuántas llamadas hace una operación de negocio típica

Y el cierre que enlaza con la clase siguiente: para que dos servicios se hablen sin coordinarse hace falta algo que ninguna malla proporciona: un contrato que pueda cambiar sin romper a quien lo consume. Es la materia de la clase 153.

Ejemplo trabajado

CloudShop, con cinco unidades desplegables tras la clase 148, se plantea si necesita una malla. La evaluación dice que no, y el mismo ejercicio descubre dos problemas de reparto que llevaban meses causando incidentes.

La evaluación de la malla.

```
servicios                                                  5
lenguajes                                                  2
instancias totales                                        48

lo que se quería de la malla
  cifrado interno y autenticación mutua        clase 136   ← necesario
  autorización por identidad                   clase 135   ← ya resuelto
  plazos y reintentos uniformes                clase 130   ← ya en código
  reparto para canarios                        clase 102   ← ya en plataforma
  telemetría uniforme                          clase 124   ← ya resuelto
```

Uno de cinco. Y el coste estimado:

```
memoria adicional        48 × 90 MB                    4,3 GB
procesador adicional                                    ~9 %
latencia añadida         2 saltos × 0,6 ms             1,2 ms
plano de control                          un sistema más que operar
coste mensual estimado                                 620 €
```

```
tiempo de implantación                                  6 semanas
```

Y la alternativa para lo único que faltaba:

```
cifrado interno y autenticación mutua
  opción A   malla completa                    6 semanas, 620 €/mes
  opción B   certificados gestionados por la plataforma
             y bibliotecas de los 2 lenguajes    1 semana, 0 €/mes

decisión   opción B
revisión   se reconsiderará por encima de 15 servicios
           o si aparece un tercer lenguaje
```

Y lo que se anotó como criterio, para no rediscutirlo cada trimestre:

```
la malla se adopta cuando el coste de mantener la política en N
bibliotecas supere el de operar un plano de control
→ con 2 lenguajes y 5 servicios, no lo supera
```

El problema del reparto rotatorio.

Un incidente recurrente sin explicación:

```
síntoma      cada pocas semanas, el percentil 99 del catálogo se
             multiplicaba por 8 durante 20-40 minutos
frecuencia   1-2 veces al mes
diagnóstico previo    «picos de tráfico»
```

Y al mirarlo con trazas (clase 124):

```
instancias del catálogo                                    12
latencia p99 de 11 de ellas                              48 ms
latencia p99 de 1 de ellas                            3.900 ms
reparto                                                rotatorio
→ la instancia lenta recibía la misma proporción de peticiones
→ y como tardaba más, acumulaba peticiones en curso
→ el 8,3 % de las peticiones tardaba 3,9 s
```

Y la causa de la instancia lenta variaba —un disco degradado, un vecino ruidoso, una pausa de memoria— pero el efecto era siempre el mismo.

```
p99 global con una instancia lenta        3.900 ms            71 ms
proporción de peticiones afectadas           8,3 %            0,4 %
peticiones que recibe la instancia lenta     8,3 %            0,9 %
cambios de código necesarios                    —               0
```

Un cambio de algoritmo en la configuración del repartidor eliminó una familia entera de incidentes.

Y se añadió la comprobación correspondiente al catálogo de la clase 131:

```
ensayo: ralentizar una instancia a propósito
  con rotatorio          p99 global ×80
  con menos peticiones   p99 global ×1,5
```

El problema de las conexiones persistentes.

Otro síntoma sin explicación:

```
al escalar el catálogo de 12 a 20 instancias durante un pico,
la latencia no mejoraba
y las 8 instancias nuevas tenían el 3 % del tráfico
```

Y la causa era la del apartado tercero:

```
conexiones persistentes multiplexadas entre servicios
el reparto elegía destino AL ABRIR la conexión
las conexiones llevaban abiertas horas
→ las instancias nuevas no recibían casi nada
```

```
reparto                              por conexión      por petición
caducidad de conexión                 no había        5 min o 10.000
                                                      peticiones
tráfico que recibe una instancia
nueva a los 2 min                        3 %              97 %
tiempo hasta que el escalado surte
efecto                                >30 min             90 s
```

Y una tercera corrección que redujo el pico de latencia al escalar:

```
arranque progresivo de las instancias nuevas
  primeros 30 s     10 % del tráfico que le tocaría
  hasta 90 s        subida gradual al 100 %

p99 de las instancias nuevas en su primer minuto
  sin arranque progresivo                          1.900 ms
  con arranque progresivo                            180 ms
```

La conversación excesiva entre servicios.

Al medir cuántas llamadas hacía cada operación de negocio:

```
operación                          llamadas entre servicios
ver ficha de producto                                     3
añadir a la cesta                                         4
ver la cesta                                             11
confirmar pedido                                         14
ver «mis pedidos»                                        62   ← problema
```

Y las 62 eran la escalera de siempre, ahora entre servicios:

```
1 llamada para la lista de pedidos
20 llamadas, una por pedido, para su estado de envío
41 llamadas, una por producto, para su nombre e imagen
```

```
llamadas para «mis pedidos»                      62              3
cómo                                  una por elemento   consultas por lotes
latencia p99                               2.100 ms        190 ms
cambios en los servicios llamados                 —      2 operaciones por lote
```

Y se añadió a la revisión de diseño una pregunta: ¿cuántas llamadas entre servicios hace esta pantalla?, con umbral de aviso en diez.

Síncrono frente a asíncrono, revisado.

```
interacciones entre servicios                                  23
síncronas antes                                                19
síncronas que no necesitaban serlo                              7
  notificar al cliente, indexar en búsqueda, actualizar
  el panel de informes, registrar en el lago, avisar a
  logística, recalcular recomendaciones, enviar a analítica
```

```
interacciones síncronas                          19             12
latencia p99 de confirmar pedido              840 ms         310 ms
dependencias duras del flujo de compra           6              3
techo de disponibilidad por dependencias     99,05 %        99,74 %
```

Siete interacciones que se hicieron asíncronas bajaron la latencia a un tercio y subieron el techo de disponibilidad, que es exactamente la cadena de la clase 126.

A los cuatro meses.

```
malla adoptada                                   —         no, con criterio
                                                           escrito
cifrado interno entre servicios               4 de 5        5 de 5
algoritmo de reparto                        rotatorio    menos peticiones
incidentes por instancia lenta             1-2 / mes           0
reparto                                   por conexión    por petición
tráfico a instancias nuevas a los 2 min         3 %            97 %
tiempo hasta que el escalado surte efecto    >30 min          90 s
llamadas para «mis pedidos»                      62             3
interacciones síncronas                          19            12
latencia p99 de confirmar pedido              840 ms         310 ms
```

La lección que esta clase traslada a la parte 12: la malla se descartó porque de las cinco cosas que aporta, cuatro ya estaban resueltas y la quinta costaba una semana en vez de seis. Y los dos incidentes recurrentes que llevaban meses sin explicación no eran problemas de arquitectura ni de código: eran el algoritmo de reparto por defecto, que manda trabajo a la instancia lenta, y conexiones persistentes que hacían que las instancias nuevas no recibieran tráfico. Los dos se corrigieron cambiando configuración, y los dos habrían seguido igual con una malla mal configurada.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-12-cloud-native-distributed-architecture/152-service-discovery-malla-y-
```

```
comunicacion/lab.py
```

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

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa topologia-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 topología con rutas, puertos y controles. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye topologia-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 sola instancia degradada dispara el percentil 99 de todo el servicio	El reparto rotatorio le envía la misma proporción de tráfico aunque tarde diez veces más	Reparte por menos peticiones en curso, o elige dos al azar y quédate con la menos cargada.
Escalar no mejora nada y las instancias nuevas casi no reciben tráfico	Las conexiones persistentes eligieron destino al abrirse y no se renuevan	Reparte por petición, caduca las conexiones por tiempo o número y da tráfico progresivo a las instancias nuevas.
Se adopta una malla y la mayoría de sus ventajas ya estaban resueltas	No se comparó lo que aporta con lo que ya existe ni se midió su coste	Enumera las cinco capacidades, marca cuáles faltan y decide con el número de servicios y lenguajes.
La malla reintenta y se duplican efectos	Los reintentos automáticos no saben si la operación es idempotente	Haz idempotente lo que se pueda reintentar y desactiva los reintentos donde no lo sea.
Una pantalla hace decenas de llamadas entre servicios	Operaciones demasiado finas y consultas por elemento	Añade consultas por lotes o una fachada de composición, y mide llamadas por operación de negocio.
La latencia del flujo principal es alta y su disponibilidad baja	Hay interacciones síncronas que no necesitan serlo	Clasifica cada interacción; lo que no se necesita para responder, hazlo asíncrono.

■ 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 ventajas e inconvenientes tiene poner la política en la biblioteca, en la plataforma o en un intermediario?
2. ¿Qué cinco cosas aporta una malla y cuáles cuatro no aporta aunque se le atribuyan?
3. ¿Por qué el reparto rotatorio es la peor opción ante una instancia lenta?
4. ¿Por qué las conexiones persistentes rompen el reparto y cómo se corrige?
5. ¿Qué regla reduce más la latencia percibida al clasificar interacciones?

Referencias

• Istio (2025). Traffic management and mutual TLS — capacidades de una malla y su configuración. <https://istio.io/latest/docs/concepts/traffic-management/>

• Envoy (2025). Load balancing: algorithms and panic threshold — menos peticiones, dos al azar y fracción sana mínima. <https://www.envoyproxy.io/docs/envoy/latest/intro/archoverview/upstream/loadbalancing/overview>

• Mitzenmacher, M. (2001). The power of two choices in randomized load balancing — por qué elegir dos al azar funciona tan bien. <https://www.eecs.harvard.edu/michaelm/postscripts/tpds2001.pdf>

• gRPC (2025). Load balancing with long-lived connections — el problema del reparto con conexiones persistentes. <https://grpc.io/blog/grpc-load-balancing/>

• Google Cloud (2025). Service discovery patterns — descubrimiento en la biblioteca, en la plataforma y con intermediario. <https://cloud.google.com/architecture/service-discovery-patterns>

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

Descargar: Parte 12 en PDF · Manual integral

Anterior	Índice	Siguiente
← 151 · Fallos parciales y patrones de resiliencia	Parte 12 · Programa	153 · Contratos API, compatibilidad y evolución →

Evaluación — 152 Service discovery, malla y comunicación

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 topologia-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.

153 — Contratos API, compatibilidad y evolución

Parte: 12 — Arquitectura cloud-native y sistemas distribuidos Nivel: avanzado-experto · Horas estimadas: 4
Laboratorio: api · Estado: EXECUTABLECORE

Propósito

Hacer que dos piezas puedan evolucionar sin coordinarse, que es la única razón por la que se separaron. La clase sostiene tres cosas incómodas: que el contrato es mucho más que el esquema, y que la mayoría de las roturas ocurren en lo que el esquema no describe; que con suficientes consumidores, cualquier comportamiento observable se convierte en parte del contrato aunque nadie lo prometiera; y que la compatibilidad no se consigue con disciplina sino comprobándola automáticamente, con una técnica que además resuelve la pregunta de quién consume esto.

Resultados de aprendizaje

Al finalizar podrás:

1. Enumerar todo lo que forma parte de un contrato además del esquema.
2. Reconocer los comportamientos no prometidos de los que alguien ya depende.
3. Escribir clientes tolerantes que no se rompan con cambios compatibles.
4. Comprobar la compatibilidad de forma automática, no por disciplina.
5. Evolucionar un contrato sin versionar, y versionar solo cuando no quede otra.

Conceptos centrales

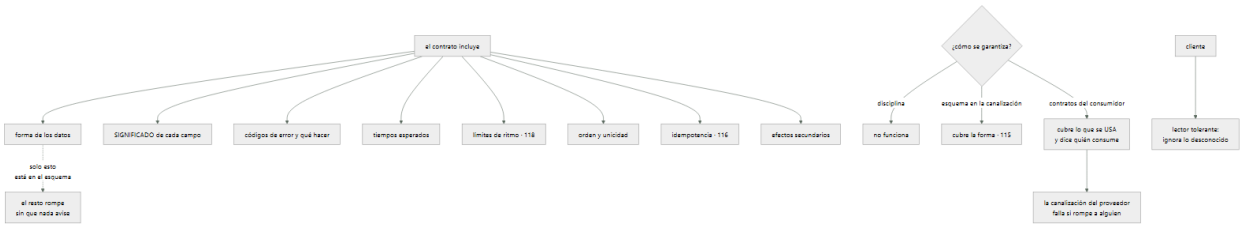
Concepto	Comprensión verificable
contrato	Todo lo que quien consume puede observar y de lo que puede depender: forma, significado, errores, tiempos, orden, límites y efectos.
comportamiento observable	Cualquier cosa que se pueda medir desde fuera. Con bastantes consumidores, alguien depende de ella aunque no esté documentada.
lector tolerante	Cliente que ignora lo que no conoce, valida solo lo que usa y no depende de lo que no se le prometió.
contrato dirigido por el consumidor	Cada consumidor publica lo que realmente usa; la canalización del proveedor comprueba que sigue cumpliéndolo.
campo con alias	Publicar el campo viejo y el nuevo a la vez durante la transición, para poder retirar el viejo sin coordinar.
error como contrato	Los códigos de error, su significado y si conviene reintentar forman parte del contrato tanto como los campos.

Modelo mental

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

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 contrato es más que el esquema

Un esquema describe la forma de los datos. El contrato es todo lo que quien consume puede observar:

```
FORMA          campos, tipos, obligatoriedad, anidamiento
SIGNIFICADO    qué representa cada campo, en qué unidad, con qué precisión
ERRORES        qué códigos existen, qué significan, cuáles conviene reintentar
TIEMPOS        cuánto tarda normalmente, qué plazo tiene sentido    clase 130
LÍMITES        cuántas peticiones se admiten y qué pasa al superarlos
                                                                    clase 118
ORDEN          si los resultados vienen ordenados, y por qué
UNICIDAD       si un identificador puede repetirse
IDEMPOTENCIA   qué ocurre si la misma petición llega dos veces      clase 116
EFECTOS        qué cambia en el mundo al llamar: cobros, correos, reservas
PAGINACIÓN     cómo se recorre, y si es estable mientras se recorre
```

Y el dato que ordena la clase:

```
el esquema cubre la primera línea
y la mayoría de las roturas ocurren en las otras nueve
```

El caso de la clase 115 es el ejemplo perfecto: el campo importe pasó de céntimos a unidades. El tipo no cambió, el esquema siguió validando, ninguna herramienta detectó nada y se emitieron mil doscientas facturas mal.

Y las otras roturas frecuentes que ningún esquema ve:

```
un campo opcional que antes venía siempre y ahora a veces falta
un listado que antes venía ordenado y ahora no
un código de error que cambia de 400 a 409
una operación que era idempotente y deja de serlo
una llamada que tardaba 40 ms y pasa a tardar 900
una paginación que se desordena si hay escrituras mientras se recorre
y un efecto nuevo: la operación ahora también envía un correo
```

La última merece atención: añadir un efecto es un cambio incompatible, aunque la respuesta sea idéntica.

Y de ahí la regla de redacción de un contrato:

```
lo que no se promete, se dice que no se promete
  «el orden de este listado no está garantizado»
  «este campo puede faltar»
  «esta operación no es idempotente»
→ decirlo explícitamente es lo único que permite cambiarlo después
```

2. Todo lo observable acaba siendo contrato

Hay una regularidad conocida y muy incómoda:

```
con suficientes consumidores, CUALQUIER comportamiento observable
de tu sistema acaba siendo algo de lo que alguien depende
aunque nunca lo hayas prometido
```

Y sus manifestaciones concretas:

```
el orden en que devuelves una lista, que nunca ordenaste a propósito
el texto exacto de un mensaje de error, que alguien compara
que un identificador tenga siempre 24 caracteres
que un campo numérico nunca sea negativo
que la respuesta llegue en menos de 100 ms
que dos llamadas seguidas devuelvan lo mismo
que un campo opcional venga siempre
```

Y las dos consecuencias prácticas:

```
1. lo que no quieras que se convierta en contrato, HAZLO VARIAR
   → si el orden no está garantizado, devuélvelo desordenado a propósito
   → si el identificador es opaco, que no tenga longitud fija

2. lo que ya lleva años estable, YA ES CONTRATO
   → cambiarlo romperá a alguien, esté documentado o no
   → y la pregunta ya no es si es legítimo, sino a quién avisas
```

Y la primera es una técnica real y poco usada: introducir variación deliberada en lo que no se promete. Es la misma idea que la variación aleatoria de las caducidades de la clase 111, aplicada a los contratos.

El lector tolerante es la contramedida del lado del consumidor, y es la disciplina que más roturas evita:

```
IGNORA lo que no conoce
  un campo nuevo no debe provocar un error de validación
VALIDA solo lo que usa
  si no lees ese campo, no compruebes su formato
NO DEPENDE de lo no prometido
  ni del orden, ni de la longitud, ni del texto de los mensajes
TRATA lo desconocido con un caso por defecto
  un valor nuevo en una enumeración no debe romper nada
```

Y la última es la que permite que un proveedor añada estados sin coordinar: si el consumidor tiene un caso «desconocido», el proveedor puede evolucionar; si no, no.

Y conviene decirlo al revés, porque es lo que hay que exigir en la documentación:

```
«los clientes deben ignorar campos que no conozcan
 y tratar valores desconocidos de enumeración como no soportados»
→ si no está escrito, no ocurrirá
```

3. Comprobar, no confiar

La compatibilidad no se consigue con buenas intenciones. Los tres mecanismos, de menor a mayor cobertura:

```
1. COMPROBACIÓN DE ESQUEMA EN LA CANALIZACIÓN            clase 115
   compara el esquema nuevo con el anterior y bloquea lo incompatible
   + barato y automático
   − solo cubre la forma; no ve significados ni comportamientos

2. CONTRATOS DIRIGIDOS POR EL CONSUMIDOR      ← el que más aporta
   cada consumidor publica un fichero con lo que REALMENTE usa:
   qué operaciones llama, con qué datos y qué espera recibir
   la canalización del proveedor ejecuta todos esos ficheros contra
   su versión nueva y falla si rompe a alguno
   + cubre lo que se usa de verdad, no lo que dice el esquema
   + y responde estructuralmente a «¿quién consume esto?»
   − exige que los consumidores participen: es un coste organizativo

3. PRUEBAS CONTRA EL SISTEMA REAL
   + realismo máximo
   − lentas, frágiles y no escalan con el número de parejas
```

Y la segunda merece detalle porque cambia la dinámica del equipo:

```
el consumidor escribe lo que espera
  «cuando pido el pedido 1421, recibo un objeto con id, estado
   e importe_centimos; el estado es uno de estos cinco valores»

eso se publica en un sitio común

la canalización del proveedor, en cada cambio
  toma todos los ficheros publicados
  levanta su versión nueva
  y comprueba que satisface a todos
→ si un cambio rompe a un consumidor, la construcción falla ANTES
  de desplegar
```

Y lo que resuelve además, que es lo más valioso:

```
el proveedor SABE quiénes son sus consumidores y qué usan
→ el problema de la clase 118 —no poder retirar una versión porque
  no se sabe quién la usa— desaparece
→ y un campo que nadie declara usar se puede retirar sin miedo
```

Y sus límites honestos:

```
cubre a los consumidores que participan; no a los que no
→ por eso sigue haciendo falta medir el uso real            clase 118

no comprueba el SIGNIFICADO
→ el caso de los céntimos pasaría igual, salvo que el consumidor
  compruebe un valor concreto
→ y por eso la regla de la clase 115 sigue siendo obligatoria:
  si cambia el significado, cambia el nombre
```

Y una tercera comprobación, barata y muy eficaz, que casi nadie hace:

```
reproducir tráfico real contra la versión nueva y comparar respuestas
→ detecta diferencias en campos, en orden y en tiempos
→ y no necesita que nadie escriba nada
```

4. Evolucionar sin versionar

Versionar es la última opción, no la primera. Lo que se puede hacer antes:

```
AÑADIR, NUNCA QUITAR NI RENOMBRAR
  campos opcionales, operaciones nuevas, valores nuevos de enumeración
  → si los consumidores son tolerantes, esto no rompe nada

CAMPO CON ALIAS DURANTE LA TRANSICIÓN
  se publica el viejo y el nuevo a la vez
  se avisa
  se mide quién sigue usando el viejo
  y se retira cuando nadie lo use
  → es expandir y contraer de la clase 102 aplicado al contrato

VALOR POR DEFECTO EN EL PROVEEDOR
  un campo que pasa a ser obligatorio se rellena con un valor
  razonable para los clientes que no lo envían

TOLERAR LO QUE YA NO SE NECESITA
  aceptar el campo viejo e ignorarlo, en vez de rechazarlo
```

Y solo cuando ninguna de esas sirve:

```
VERSIÓN MAYOR                                          clase 118
  con el procedimiento de retirada: medir, avisar, cortes de prueba, apagar
  y con un número máximo de versiones vivas escrito de antemano
```

Los errores, que son la parte del contrato peor documentada:

```
cada error necesita
  un código estable, que no cambia aunque cambie el mensaje
  un significado escrito
  si conviene reintentar o no                          clase 113
  y qué debería hacer el cliente

y el formato debe ser uniforme en toda la organización   clase 118
```

Y la distinción que más incidentes evita:

```
error del cliente, no reintentable    petición mal formada, falta permiso
error del cliente, reintentable       conflicto de versión, límite superado
error del servidor, reintentable      no disponible, tiempo agotado
error del servidor, no reintentable   fallo permanente en el procesamiento
```

Sin esa clasificación, el cliente reintenta lo que no debe y no reintenta lo que sí, que es la mitad de los problemas de la clase 113.

Contratos internos y externos tienen ritmos distintos, y conviene no tratarlos igual:

```
INTERNOS   se conocen todos los consumidores
           se pueden cambiar en semanas, con contratos comprobados
           versiones vivas: 1, casi siempre

EXTERNOS   no se controla a los consumidores
           cambios en meses, con procedimiento de retirada  clase 118
           versiones vivas: 2
```

Y la línea que los separa no es técnica: es si se sabe quién consume y se le puede avisar. Un contrato interno cuyo consumidor está en otro país y otra empresa es, a efectos prácticos, externo.

Y la lista de comprobación de la clase:

- el contrato documenta significado, errores, tiempos, orden e idempotencia
- lo que no se promete está escrito como no prometido
- lo no prometido varía a propósito, para que nadie dependa de ello
- los clientes son lectores tolerantes, y está exigido por escrito
- hay comprobación de compatibilidad de esquema en la canalización
- hay contratos publicados por los consumidores y verificados por el proveedor
- se conoce quién consume cada operación y cada campo
- los cambios se hacen añadiendo, con alias durante la transición
- los errores tienen código estable y clasificación de reintentabilidad
- está escrito cuántas versiones vivas se admiten y cómo se retira una
- si cambia el significado de un campo, cambia su nombre

Y el cierre que enlaza con la clase siguiente: un contrato bien hecho permite que varios consumidores usen el mismo sistema. Cuando esos consumidores son clientes distintos que comparten infraestructura, aparecen problemas propios —aislamiento, ruido entre vecinos y coste por cliente— que son la materia de la clase 154.

Ejemplo trabajado

CloudShop tiene cinco unidades desplegables que se llaman entre sí y una API externa con ciento noventa socios. En doce meses hubo once roturas de contrato; el ejercicio consiste en clasificarlas y montar la comprobación que las habría evitado.

Las once roturas, clasificadas.

```
qué cambió                                      ¿lo veía el esquema?
significado de un campo (céntimos)                     no        clase 115
campo opcional que dejó de venir siempre                no
orden de un listado que nadie prometió                  no
código de error de 400 a 409                            no
operación que dejó de ser idempotente                   no
latencia de 40 ms a 900 ms                              no
efecto nuevo: la operación envía un correo              no
paginación inestable con escrituras concurrentes        no
campo renombrado                                        SÍ
tipo cambiado de entero a texto                         SÍ
campo obligatorio nuevo en la petición                  SÍ
```

Ocho de once no las veía ningún esquema, y las tres que sí las habría visto se detuvieron después de implantar la comprobación de la clase 115.

El caso del orden: la ley de lo observable.

```
la operación «pedidos de un cliente» devolvía la lista ordenada
por fecha descendente, porque la consulta llevaba un ORDER BY
que nadie había documentado

al cambiar la consulta por una más eficiente, el orden se perdió
consumidores rotos                                          4
  la aplicación móvil mostraba pedidos desordenados
  un socio asumía que el primero era el más reciente
  un informe interno tomaba el primero
  la aplicación web ordenaba en cliente y no se enteró
```

Y la corrección tuvo dos partes:

```
1. inmediata     documentar el orden y garantizarlo, porque ya era contrato
2. preventiva    en las operaciones donde el orden NO se promete,
                 devolverlo desordenado a propósito

listados con orden no prometido                          11
de ellos, que se descubrió que alguien usaba ordenados    3
  → se documentaron y se garantizaron
los otros 8                    barajados a propósito desde entonces
roturas por orden en 12 meses siguientes                  0
```

Y la lección que se escribió: lo que lleva dos años comportándose igual ya es contrato, y la pregunta no es si era legítimo cambiarlo.

Los contratos dirigidos por el consumidor.

Se implantaron primero entre los cinco servicios internos:

```
parejas proveedor-consumidor                                14
contratos publicados por los consumidores                   14
líneas por contrato, media                                  60
tiempo de implantación                                3 semanas
```

Y el primer efecto fue inmediato:

```
primera ejecución sobre los cambios pendientes de despliegue
  cambios que rompían a algún consumidor                     3
  de ellos, que el esquema consideraba compatibles           2
    → un campo opcional que un consumidor usaba como obligatorio
    → un valor de enumeración nuevo que otro no sabía tratar
```

Y el segundo, que era el objetivo:

```
roturas de contrato entre servicios internos  7 / 12 meses      0
detectadas antes de desplegar                     0             9
tiempo medio de detección                     3 días         2 min
```

Y el efecto lateral más valioso:

```
campos del contrato de pedidos                              41
campos que algún consumidor declara usar                    22
campos que nadie usa                                        19
→ se retiraron los 19, con aviso y ventana de un mes
→ y ninguno produjo ninguna incidencia
```

Diecinueve campos retirados sin miedo, porque por primera vez se sabía que nadie los usaba.

La API externa, donde los contratos del consumidor no sirven.

```
socios                                                     190
socios dispuestos a publicar un contrato                     6
```

Y para el resto se usó la tercera técnica del apartado tercero:

```
reproducción de tráfico real contra la versión nueva
  peticiones reproducidas por ejecución                 50.000
  comparación de respuestas: campos, valores, códigos y orden

diferencias detectadas en 8 meses                           14
  de ellas, intencionadas                                   11
  de ellas, NO intencionadas                                 3
    → un campo que dejaba de venir cuando estaba vacío
    → un código de error que cambiaba
    → un redondeo distinto en un importe
```

Tres roturas detectadas sin que ningún socio tuviera que hacer nada, y con un coste de veinte minutos por ejecución.

Los errores, documentados y clasificados.

```
códigos de error distintos que devolvía la API              47
documentados                                                11
con clasificación de reintentabilidad                        0
```

Y el efecto de no tenerla, medido en el tráfico de socios:

```
reintentos de socios ante errores no reintentables         31 % de
                                                            los reintentos
carga inútil generada                                ~180.000 peticiones/mes
socios que NO reintentaban errores transitorios             41
  → y perdían operaciones que se habrían resuelto solas
```

```
códigos documentados                          11 de 47       47 de 47
con clasificación de reintentabilidad             0             47
formato uniforme                                 no             sí
reintentos de errores no reintentables          31 %            3 %
operaciones perdidas por no reintentar     ~900 / mes       ~40 / mes
```

La evolución sin versionar.

En doce meses hubo veintitrés cambios en el contrato externo:

```
añadidos compatibles                                        17
con campo alias durante transición                           4
```

```
  → el caso de importe_centimos, hecho bien esta vez
versión mayor nueva                                          0
versiones vivas                                              2
```

Y el caso del alias, con las cifras:

```
se publica «importe_centimos» junto a «importe»
aviso a los 190 socios, con fecha de retirada a 6 meses
socios que migraron en el primer mes                        31
socios que migraron tras el primer corte de prueba          94
socios que migraron tras el segundo                          58
socios que quedaban al apagar                                7
  → todos inactivos desde hacía más de 3 meses
roturas al retirar el campo viejo                            0
```

Cero roturas, con el mismo cambio que la primera vez produjo mil doscientas facturas mal.

A los doce meses.

```
roturas de contrato                          11 / año          1
detectadas antes de desplegar                     0           12
contratos publicados por consumidores             0           14
campos del contrato que nadie usa                19            0
listados con orden no prometido, barajados        0            8
códigos de error documentados                 11 de 47     47 de 47
con clasificación de reintentabilidad             0           47
reintentos inútiles de socios                  31 %           3 %
versiones mayores vivas                       2 (sin política) 2 (escrito)
cambios que exigieron versión nueva               3            0
reproducción de tráfico en la canalización        no           sí
```

La lección que esta clase traslada a la parte 12: de once roturas en un año, ocho ocurrieron en cosas que ningún esquema describe: el significado de un campo, el orden de una lista, un código de error, la latencia y un efecto nuevo. Y la técnica que más aportó no fue una comprobación mejor: fue hacer que cada consumidor publicara lo que de verdad usa, porque eso convirtió «no sabemos quién consume esto» —el problema que la clase 118 no pudo resolver en dos años— en una lista de veintidós campos usados y diecinueve que se pudieron retirar el mismo mes.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-12-cloud-native-distributed-architecture/153-contratos-api-compatibilidad-y-evolucion/lab.py
```

El laboratorio selecciona el motor de práctica api 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 contrato-versionado 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 contrato versionado con pruebas positivas y negativas. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye contrato-versionado 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 cambio que el esquema considera compatible rompe a los consumidores	El contrato incluye significado, errores, tiempos, orden e idempotencia, y el esquema solo cubre la forma	Documenta las nueve dimensiones, y si cambia el significado de un campo, cambia su nombre.
Alguien depende de algo que nunca se prometió	Todo comportamiento observable acaba siendo contrato	Escribe explícitamente lo que no se promete y hazlo variar a propósito para que nadie pueda depender de ello.
Un campo nuevo rompe a un cliente	El cliente valida todo lo que recibe en vez de solo lo que usa	Exige por escrito lectores tolerantes: ignorar lo desconocido y tratar valores nuevos de enumeración con un caso por defecto.
No se puede retirar un campo porque no se sabe quién lo usa	No hay contratos publicados por los consumidores ni medición de uso	Contratos dirigidos por el consumidor para lo interno y medición de uso por campo y cliente para lo externo.
Los clientes reintentan lo que no deben y no reintentan lo que sí	Los errores no están documentados ni clasificados por reintentabilidad	Código estable, significado escrito y clasificación explícita de qué conviene reintentar.
Cada cambio pequeño exige una versión nueva	Se versiona antes de agotar las técnicas compatibles	Añade sin quitar, usa campos con alias durante la transición y reserva la versión mayor para cuando no quede otra.

■ 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 nueve cosas incluye un contrato además de la forma de los datos?
2. ¿Por qué conviene hacer variar a propósito lo que no se promete?
3. ¿Qué es un lector tolerante y qué permite al proveedor?
4. ¿Qué resuelve un contrato dirigido por el consumidor además de detectar roturas?
5. ¿Qué técnica sirve cuando los consumidores no pueden participar?

Referencias

- Fowler, M. (2011). Tolerant reader — ignorar lo desconocido y depender de lo mínimo. <https://martinfowler.com/bliki/TolerantReader.html>
- Fowler, M. (2011). Consumer-driven contracts — el consumidor publica lo que usa y el proveedor lo verifica. <https://martinfowler.com/articles/consumerDrivenContracts.html>
- Wright, H. (2025). Hyrum's law — con suficientes consumidores, todo comportamiento observable es contrato. <https://www.hyrumslaw.com/>

• Pact (2025). Contract testing: broker and provider verification — implantación de contratos del consumidor. <https://docs.pact.io/>
• Google (2025). API design guide: compatibility — qué cambios son compatibles y cuáles no. <https://cloud.google.com/apis/design/compatibility>
• Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 12 en PDF · Manual integral

Anterior	Índice	Siguiente
← 152 · Service discovery, malla y comunicación	Parte 12 · Programa	154 · Multi-tenancy, aislamiento y noisy neighbor →

Evaluación — 153 Contratos API, compatibilidad y evolución

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 contrato-versionado 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.

154 — Multi-tenancy, aislamiento y noisy neighbor

Parte: 12 — Arquitectura cloud-native y sistemas distribuidos Nivel: avanzado-experto · Horas estimadas: 4
Laboratorio: architecture · Estado: EXECUTABLECORE

Propósito

Servir a muchos clientes con la misma infraestructura sin que se vean entre sí ni se estorben. La clase sustituye la pregunta binaria —compartido o dedicado— por la que sirve: el aislamiento tiene seis dimensiones y cada una puede estar en un nivel distinto. Desarrolla con detalle la que no admite errores, porque un solo fallo la rompe entera —que un cliente vea datos de otro—, da la aritmética del vecino ruidoso y termina con lo que de verdad hunde a los productos multiinquilino: las operaciones que hay que hacer cliente a cliente.

Resultados de aprendizaje

Al finalizar podrás:

1. Situar cada recurso en el nivel de aislamiento adecuado, por dimensiones.
2. Impedir por construcción que un cliente vea datos de otro.
3. Acotar el consumo de cada cliente en todos los recursos compartidos.
4. Operar migraciones, copias y borrados cliente a cliente sin que crezca el coste.
5. Calcular el coste por cliente y ofrecer niveles con camino entre ellos.

Conceptos centrales

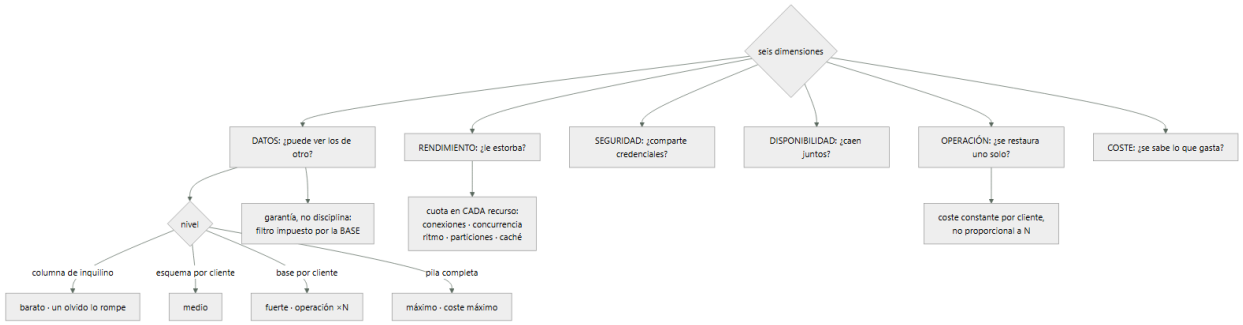
Concepto	Comprensión verificable
dimensiones de aislamiento	Datos, rendimiento, seguridad, disponibilidad, operación y coste. Cada una puede estar en un nivel distinto.
identificador de inquilino	Dato que acompaña a toda operación. Si falta en un solo sitio, se filtra información entre clientes.
seguridad a nivel de fila	La base impone el filtro por cliente, no la aplicación. Convierte una disciplina en una garantía.
vecino ruidoso	Cliente cuyo consumo degrada a los demás. Se acota con cuotas en cada recurso compartido.
operación por cliente	Migrar, copiar, restaurar, borrar o mover un cliente. Su coste debe ser constante, no proporcional al número de clientes.
nivel de servicio	Grado de aislamiento ofrecido como producto. Necesita camino de migración entre niveles desde el primer día.

Modelo mental

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

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. Seis dimensiones, no una decisión

La pregunta habitual —«¿compartido o dedicado?»— es demasiado gruesa. Lo que hay que decidir es por dimensión y por recurso:

```
DATOS           ¿puede un cliente ver los de otro?
RENDIMIENTO     ¿el consumo de uno degrada a los demás?
SEGURIDAD       ¿comparten credenciales, claves o permisos?
DISPONIBILIDAD  ¿una caída afecta a todos a la vez?
OPERACIÓN       ¿se puede restaurar, migrar o borrar uno solo?
COSTE           ¿se sabe cuánto gasta cada uno?
```

Y cada una admite niveles distintos:

```
datos           esquema por cliente
rendimiento     cuotas sobre recursos compartidos
seguridad       clave de cifrado por cliente            clase 136
disponibilidad  compartida, salvo tres clientes con celda propia
operación       herramientas por cliente
coste           medido por cliente                      clase 142
```

Y los niveles clásicos, con lo que cuesta cada uno:

```
COLUMNA DE INQUILINO EN CADA TABLA
  + el más barato; una base para todos
  – un olvido en una consulta lo rompe entero
  – y restaurar un cliente exige extraerlo del conjunto

ESQUEMA POR CLIENTE, MISMA BASE
  + el filtro deja de depender de cada consulta
  + copias y borrado por cliente son viables
  – las migraciones se ejecutan N veces

BASE POR CLIENTE
  + aislamiento fuerte de datos y de rendimiento
  – coste y operación proporcionales al número de clientes

PILA COMPLETA POR CLIENTE
  + máximo aislamiento; es una celda de la clase 151
  – coste máximo; solo para pocos clientes grandes
```

Y la regla que evita elegir mal:

```
el nivel lo decide el REQUISITO, no la comodidad
  ¿lo exige una norma o un contrato?            clase 141
  ¿el cliente paga por ello?
  ¿qué pasa si se rompe esta dimensión?
```

Y una decisión que hay que tomar el primer día porque después es irreversible —ley 14—:

```
¿existe un camino para MOVER un cliente de un nivel a otro?
  si no existe, el nivel inicial es para siempre
  y el primer cliente grande obligará a rehacerlo todo
```

2. Que no pueda ver lo de otro

Esta es la dimensión que no admite grados: basta un fallo en un sitio para romperla entera, y sus consecuencias son legales además de técnicas.

De dónde salen las fugas, por frecuencia:

```
UNA CONSULTA SIN EL FILTRO
  la más común; suele estar en un informe o en una función nueva

UNA CLAVE DE CACHÉ SIN EL CLIENTE                          clase 111
  el segundo usuario ve la respuesta del primero

UN PROCESO POR LOTES QUE RECORRE EL CONJUNTO ENTERO
  y escribe el resultado en el contexto equivocado

UNA HERRAMIENTA INTERNA                                    clase 137
  que muestra a soporte más de lo que debe

UN ÍNDICE DE BÚSQUEDA COMPARTIDO
  donde el filtro se aplica al consultar y alguien lo olvida

UNA EXPORTACIÓN O UN INFORME
  generado con una consulta distinta a la de la aplicación

UN MENSAJE ENVIADO AL DESTINATARIO EQUIVOCADO
  la fuga que el cliente ve primero
```

Y las defensas, ordenadas por lo que garantizan:

```
1. SEPARACIÓN FÍSICA
   bases o esquemas distintos con credenciales distintas
   → una consulta mal escrita no puede alcanzar otros datos

2. FILTRO IMPUESTO POR LA BASE
   seguridad a nivel de fila: la base añade el filtro siempre
   → una consulta sin filtro devuelve solo lo del cliente actual
   → convierte una disciplina en una garantía

3. FILTRO EN UNA CAPA COMÚN DE ACCESO A DATOS
   ningún código accede directamente; todo pasa por ahí
   → funciona hasta que alguien escribe una consulta directa

4. DISCIPLINA Y REVISIÓN
   → falla; es cuestión de tiempo
```

Y el nivel 2 es el que mejor relación tiene entre coste y garantía en una base relacional: se configura una vez y protege también a los informes y a los procesos por lotes.

Y la comprobación que hay que automatizar, porque es la única que demuestra que funciona:

```
en cada construcción
  crear datos de dos clientes
  ejecutar TODAS las operaciones como cliente A
  y comprobar que no aparece ningún dato de B

y la versión más útil: hacerlo también con las consultas de informes,
las exportaciones y las herramientas internas
```

Y una comprobación adicional que detecta el caso del caché:

```
dos sesiones de clientes distintos piden lo mismo
y se comprueba que las respuestas DIFIEREN                  clase 122
```

Y el identificador de cliente debe viajar en el contexto, no pasarse a mano:

```
en la petición, en el mensaje de la cola, en el evento,
en el trabajo programado y en el registro                  clases 113, 115
→ si alguien tiene que acordarse de pasarlo, alguna vez se olvidará
```

3. El vecino ruidoso

Con recursos compartidos, un cliente puede consumirlos todos. Y la lista de recursos compartidos es más larga de lo que parece:

```
conexiones a la base                                       clase 109
concurrencia de funciones                                  clase 117
ritmo de peticiones a la API                               clase 118
hilos y compartimentos del servicio                        clase 130
particiones de una cola o registro                         clase 114
espacio y operaciones de almacenamiento
```

```
espacio del caché                                    clase 111
procesador y memoria del nodo
y la propia base de datos: bloqueos y consultas caras
```

Y la regla: cada uno necesita su cuota, o el más ruidoso se lleva todo.

```
sin cuotas   un cliente con un proceso mal hecho degrada a 190
con cuotas   ese cliente se degrada a sí mismo
```

Y el problema práctico es que la cuota tiene que estar en el sitio correcto:

```
en la puerta de entrada        limita peticiones                clase 118
en el agrupador de conexiones  un cliente no acapara conexiones
en la concurrencia             reserva por cliente o por nivel
en la cola                     un cliente no llena las particiones
en la base                     límite de tiempo por sentencia   clase 109
```

La última es la que más incidentes evita en sistemas con informes: una consulta de un cliente que tarda cuarenta minutos bloquea recursos de todos.

Y la técnica estructural, que la clase 151 ya presentó y aquí es especialmente aplicable:

```
REPARTO POR SORTEO
  cada cliente usa un subconjunto pequeño y distinto de los recursos
  → un cliente que satura los suyos afecta a pocos
  → y sin duplicar infraestructura
```

Y una variante muy usada: agrupar clientes por comportamiento. Los que hacen mucho trabajo por lotes van a un conjunto de recursos, y los interactivos a otro, para que las cargas pesadas no compitan con las que el usuario espera.

Y lo que hay que vigilar, por cliente:

```
peticiones, errores y latencia                          clase 123
consumo de cada recurso frente a su cuota
veces que ha alcanzado su cuota
y el cliente que más consume de cada recurso, cada semana
```

La última es la que detecta el problema antes de que sea un incidente: casi siempre hay un cliente que consume mucho más de lo que paga, y eso es a la vez un problema de rendimiento y de coste.

4. Lo que hay que hacer cliente a cliente

Aquí está lo que hunde a los productos multiinquilino, y casi nunca se planea:

```
MIGRACIONES DE ESQUEMA
  con esquema por cliente, una migración se ejecuta N veces
  → con 190 clientes y 40 s cada una, son 2 horas
  → y si falla en el cliente 137, hay que saber reanudar

COPIA Y RESTAURACIÓN DE UN SOLO CLIENTE
  «restaura los datos de este cliente a ayer»
  → con tabla compartida, es un proyecto
  → con esquema o base por cliente, es rutina

BORRADO DE UN CLIENTE                                   clase 141
  incluidos lago, registros conservados, copias y caché
  → la respuesta elegante es el borrado criptográfico
    con clave por cliente                               clase 136

MOVER UN CLIENTE
  de nivel, de región o de celda
  → exige poder exportar e importar su estado completo

PERSONALIZACIÓN POR CLIENTE
  el que más daño hace a largo plazo
```

Y la regla que resume la lista:

```
el coste de una operación por cliente debe ser CONSTANTE,
no proporcional al número de clientes
→ y eso exige herramientas, no procedimientos manuales
```

Y sobre la personalización, una advertencia que conviene decir pronto:

```
cada campo, regla o flujo específico de un cliente
  se prueba aparte
  se despliega con cuidado aparte
```

```
  y aparece en cada cambio futuro

→ un producto con 20 personalizaciones por cliente y 190 clientes
  no es un producto: son 190 productos
```

Y la salida sana: personalizar por configuración e interruptores, no por código —clase 105—, con un catálogo cerrado de opciones.

El coste por cliente, que es la clase 142 en su forma más difícil:

```
directo        lo que se puede atribuir: su base, su almacenamiento
compartido     lo que hay que repartir: nodos, red, plataforma
               → proporcional a su consumo medido, y explicable

y la cifra que decide el negocio:
  coste por cliente frente a lo que paga
  → casi siempre hay clientes que cuestan más de lo que ingresan
```

Los niveles de servicio, que es cómo se ofrece el aislamiento como producto:

```
básico       compartido, con cuotas
profesional  cuotas mayores, copia por cliente, clave propia
dedicado     base propia o celda propia, y objetivos propios
```

Y la condición sin la cual esto no funciona, que ya se enunció:

```
el camino entre niveles debe existir y estar probado
→ mover un cliente de básico a dedicado no puede ser un proyecto
→ y se ensaya con un cliente de prueba, periódicamente
```

Y la lista de comprobación de la clase:

- ■ el nivel de aislamiento está decidido por dimensión, no en bloque
- ■ el identificador de cliente viaja en el contexto, no a mano
- ■ el filtro por cliente lo impone la base, no cada consulta
- ■ las claves de caché incluyen el cliente
- ■ hay prueba automática de que un cliente no ve datos de otro
- ■ esa prueba cubre informes, exportaciones y herramientas internas
- ■ cada recurso compartido tiene cuota por cliente
- ■ hay límite de tiempo por sentencia en la base
- ■ se vigila quién consume más de cada recurso
- ■ las operaciones por cliente tienen herramienta y coste constante
- ■ la personalización es por configuración, no por código
- ■ se mide el coste por cliente frente a lo que paga
- ■ existe camino de migración entre niveles y está ensayado

Y el cierre que enlaza con la clase siguiente: aislamiento, rendimiento, seguridad y coste no se pueden maximizar a la vez, y esta clase ha ido eligiendo entre ellos sin decirlo. Hacer explícitos esos compromisos, y decidirlos con un método, es la materia de la clase 155.

Ejemplo trabajado

CloudShop sirve a 190 socios comerciales desde una infraestructura compartida. El ejercicio empieza por la dimensión que no admite fallos y termina con tres clientes movidos a un nivel dedicado, con el camino ensayado.

La situación de partida, por dimensiones.

```
DATOS           columna de inquilino en 84 tablas
RENDIMIENTO     sin cuotas por cliente en ningún recurso
SEGURIDAD       una clave de cifrado para todos
DISPONIBILIDAD  todos comparten todo
OPERACIÓN       restaurar un cliente: no había forma
COSTE           no se sabía lo que gastaba cada uno
```

Seis dimensiones en el nivel más bajo, y ninguna decidida a propósito.

La dimensión de datos: la auditoría.

Se escribió la prueba del apartado segundo y se ejecutó contra el sistema existente:

```
operaciones probadas como cliente A                          148
operaciones que devolvieron datos de B                         6
  un informe de ventas por categoría                          sí
  una exportación de productos                                sí
  el índice de búsqueda, consultado sin filtro                sí
```

```
  la herramienta interna de soporte                          sí   clase 137
  un proceso por lotes de recálculo de precios               sí
  una clave de caché sin cliente                             sí   clase 111
```

Seis de ciento cuarenta y ocho, y ninguna estaba en el camino principal: las seis estaban en informes, exportaciones, búsqueda, herramientas internas y procesos de fondo, que es exactamente la lista del apartado segundo.

Y la más grave llevaba tiempo activa:

```
el índice de búsqueda era compartido y el filtro se aplicaba
en la consulta de la aplicación
una función nueva consultaba el índice directamente
tiempo desplegada                                        4 meses
clientes que podrían haber visto productos de otros          190
consultas que lo hicieron, según registros                    0
```

Y la corrección se hizo por niveles, no por disciplina:

```
filtro por cliente                    en cada consulta   impuesto por la base
                                                         (nivel de fila)
índice de búsqueda                     compartido        un índice por cliente
                                                         en los 190
claves de caché                        sin cliente       con cliente
herramienta de soporte                 sin filtro        con filtro y registro
prueba automática por construcción     no                sí, 148 operaciones
fugas detectadas después                —                3, todas en la
                                                         canalización
```

Y el nivel de fila en la base resolvió cuatro de las seis de una vez, incluidos el informe y el proceso por lotes que nadie había pensado en revisar.

El vecino ruidoso, con números.

```
semana de medición
  socios                                                   190
  socio que más consumía                          61 % del total
  su facturación                                  1,4 % de los ingresos
```

Sesenta y uno por ciento del consumo y el uno coma cuatro por ciento de los ingresos. Y su patrón:

```
un proceso propio que consultaba el catálogo completo cada 5 minutos
41.000 peticiones/hora, contra una media de 210 por socio
```

Y el efecto sobre los demás, antes de las cuotas:

```
latencia p99 de los otros 189, en horas punta            2.100 ms
latencia p99 cuando ese socio no ejecutaba su proceso      180 ms
```

Las cuotas se pusieron en los cinco recursos del apartado tercero:

```
ritmo de peticiones por socio          sin límite      por nivel de servicio
conexiones a la base por socio         sin límite      máximo 4
concurrencia de funciones por socio    sin límite      reservada por nivel
particiones de cola                    compartidas     sorteo 2 de 12
límite de tiempo por sentencia         no había        15 s

latencia p99 de los demás en horas punta   2.100 ms        190 ms
socios afectados por un vecino ruidoso        189             3
veces que ese socio alcanzó su cuota            —          constantemente
```

Y la conversación comercial que hizo posible el cambio: el socio pasó a un nivel superior o redujo su consumo. Eligió reducirlo, porque su proceso pedía cada cinco minutos algo que cambiaba una vez al día.

Las operaciones por cliente.

```
operación                        antes                  después
migración de esquema        1 esquema compartido     190 esquemas, 2 h 10,
                                                     reanudable, con informe
restaurar un cliente a ayer  no era posible          herramienta, 12 min
borrar un cliente            9 días (clase 141)      40 min, con borrado
                                                     criptográfico
mover a nivel dedicado       no existía              herramienta, 3 h,
                                                     ensayada trimestralmente
exportar el estado completo  no existía              herramienta, 25 min
```

Y la migración de esquema por cliente, que era la que asustaba:

```
primera ejecución sobre 190 esquemas
  duración                                          2 h 10
  fallos                                               3
  reanudación desde el punto de fallo                 sí
  clientes con esquema desalineado al terminar         0
```

Y se añadió una comprobación diaria: ningún esquema puede quedar en una versión distinta a la esperada, que es la ley 13 aplicada a esto.

La personalización, atajada a tiempo.

```
personalizaciones existentes                           31
  por configuración                                    18
  POR CÓDIGO específico de un cliente                  13
     condiciones «si el cliente es X» repartidas en 6 servicios
```

Y el coste medido de las trece:

```
cambios en los que hubo que probarlas                todos
tiempo añadido por cambio, estimado                  ~15 %
incidentes causados por una de ellas en 12 meses         4
```

```
personalizaciones por código                    13          0
por configuración con catálogo cerrado          18         29
condiciones «si el cliente es X» en el código   41          0
opciones del catálogo                      no existía      12
```

Y dos de las trece no cabían en el catálogo: se rechazaron y se renegociaron con los clientes, uno de los cuales aceptó y el otro pasó a nivel dedicado.

Los niveles y el camino entre ellos.

```
BÁSICO         compartido, cuotas estándar          181 socios
PROFESIONAL    cuotas mayores, copia por cliente,
               clave de cifrado propia                6 socios
DEDICADO       base propia, objetivos propios,
               región elegible                        3 socios
```

Y los tres dedicados vinieron de tres motivos distintos, todos escritos:

```
socio A   exige que sus datos estén en su propia región     clase 141
socio B   consumo muy superior al de un básico
socio C   una personalización que no cabía en el catálogo
```

Y el camino de migración, que era la condición del apartado primero:

```
tiempo de mover un socio de básico a dedicado              3 h
parada percibida por el socio                           11 min
ensayo con un socio de prueba                         trimestral
fallos en el primer ensayo                                 4
  → exportación incompleta del índice de búsqueda
  → claves de caché que no se invalidaban
  → el catálogo de servicios no se actualizaba (clase 095)
  → la facturación seguía atribuyendo al conjunto compartido
fallos en el cuarto ensayo                                 0
```

El coste por cliente.

```
coste directo atribuible                                 41 %
coste compartido, repartido por consumo medido           53 %
bolsa sin asignar                                         6 %

coste medio por socio                                 152 €/mes
socios que cuestan más de lo que pagan                   14
de ellos, con consumo desproporcionado                    9
de ellos, con precio mal fijado                           5
```

Y los nueve se resolvieron con cuotas; los cinco pasaron a la conversación comercial.

A los nueve meses.

```
dimensiones decididas a propósito            0 de 6        6 de 6
fugas entre clientes encontradas                6             0
filtro impuesto por la base                    no            sí
prueba automática de aislamiento               no       148 operaciones
recursos compartidos con cuota               0 de 5        5 de 5
latencia p99 con un vecino ruidoso         2.100 ms       190 ms
```

```
socios afectados por un vecino                  189            3
operaciones por cliente con herramienta       0 de 5         5 de 5
restaurar un cliente                        imposible        12 min
borrar un cliente                             9 días         40 min
personalizaciones por código                    13              0
coste por cliente conocido                      no             sí
niveles de servicio                              1              3
camino entre niveles, ensayado                  no        trimestral
```

La lección que esta clase traslada a la parte 12: de las seis operaciones que filtraban datos entre clientes, ninguna estaba en el camino principal: estaban en un informe, una exportación, un índice de búsqueda, una herramienta interna, un proceso por lotes y una clave de caché. Ninguna revisión de código las habría encontrado, y cuatro se cerraron de golpe poniendo el filtro en la base en vez de en cada consulta. Y el vecino ruidoso que consumía el 61 % de la capacidad aportaba el 1,4 % de los ingresos: era a la vez el problema de rendimiento y el de coste, que es la misma observación que cerró la parte 11.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-12-cloud-native-distributed-architecture/154-multi-tenancy-aislamiento-y-noisy-neighbor/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 modelo-tenancy 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 modelo-tenancy 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 cliente ve datos de otro en un informe o una exportación	El filtro por cliente lo aplica cada consulta y alguien lo olvidó fuera del camino principal	Impón el filtro en la base con seguridad a nivel de fila y prueba automáticamente todas las operaciones, incluidos informes y herramientas internas.
Dos clientes distintos reciben la misma respuesta cacheada	La clave de caché no incluye el identificador de cliente	Incluye siempre el cliente en la clave y añade una prueba que compare respuestas de dos sesiones distintas.
Un cliente degrada el servicio de todos los demás	No hay cuotas en los recursos compartidos	Cuota por cliente en ritmo, conexiones, concurrencia, colas y tiempo por sentencia; y reparto por sorteo para acotar el alcance.
Restaurar o borrar los datos de un solo cliente es un proyecto	Los datos están mezclados y no hay herramientas por cliente	Esquema o clave de cifrado por cliente, y herramientas cuyo coste sea constante y no proporcional al número de clientes.
Cada cambio hay que probarlo contra decenas de casos particulares	Hay personalización por código específica de clientes	Personaliza por configuración con un catálogo cerrado de opciones; lo que no quepa, renegócialo o llévalo a un nivel dedicado.
El primer cliente grande obliga a rehacer la arquitectura	No existe camino de migración entre niveles de aislamiento	Diseña el camino desde el primer día y ensáyalo periódicamente con un cliente de prueba.

Seguridad, ética y costo

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

Preguntas de comprobación

1. ¿Cuáles son las seis dimensiones de aislamiento y por qué no se deciden en bloque?
2. ¿De dónde salen las fugas entre clientes y por qué no están en el camino principal?
3. ¿Qué convierte el filtro por cliente en una garantía en vez de una disciplina?
4. ¿En qué recursos hace falta cuota por cliente?
5. ¿Por qué el coste de una operación por cliente debe ser constante?

Referencias

- AWS (2025). SaaS tenant isolation strategies — niveles de aislamiento y modelos de despliegue. <https://docs.aws.amazon.com/whitepapers/latest/saas-tenant-isolation-strategies/saas-tenant-isolation-strategies.html>
- PostgreSQL (2025). Row security policies — filtro impuesto por la base y no por la consulta. <https://www.postgresql.org/docs/current/ddl-rowsecurity.html>
- Microsoft (2025). Multitenant SaaS patterns: tenancy models and noisy neighbor — modelos y cuotas. <https://learn.microsoft.com/azure/architecture/guide/multitenant/overview>
- AWS (2025). Workload isolation using shuffle sharding — acotar el alcance de un vecino ruidoso. <https://aws.amazon.com/builders-library/workload-isolation-using-shuffle-sharding/>
- Google Cloud (2025). Per-tenant cost attribution — coste por cliente con recursos compartidos. <https://cloud.google.com/architecture/framework/cost-optimization>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Evaluación — 154 Multi-tenancy, aislamiento y noisy neighbor

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-tenancy 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.

155 — Rendimiento, costo, seguridad y operabilidad

Parte: 12 — Arquitectura cloud-native y sistemas distribuidos Nivel: avanzado-experto · Horas estimadas: 4
Laboratorio: decision · Estado: EXECUTABLECORE

Propósito

Hacer explícito lo que las once clases anteriores han estado eligiendo sin decirlo. Casi ninguna decisión de arquitectura es una mejora: es un traslado entre rendimiento, coste, seguridad y operabilidad, y llamarlo mejora sin decir qué empeoró es lo que convierte una decisión en algo que después nadie puede discutir. La clase da el método para escribir el traslado en cuatro columnas, enseña a medir los cuatro para que la comparación sea cuantitativa, y señala los pocos casos en que no hay traslado y se gana en varios a la vez, que son el trabajo más rentable y el menos vistoso.

Resultados de aprendizaje

Al finalizar podrás:

1. Reconocer el traslado que hay detrás de cada decisión de arquitectura.
2. Escribir una decisión con lo que mejora y lo que empeora, con cifras.
3. Medir los cuatro atributos para poder comparar.
4. Resolver el conflicto con el orden de prioridad, no con la opinión.
5. Buscar primero los cambios que mejoran dos o más a la vez.

Conceptos centrales

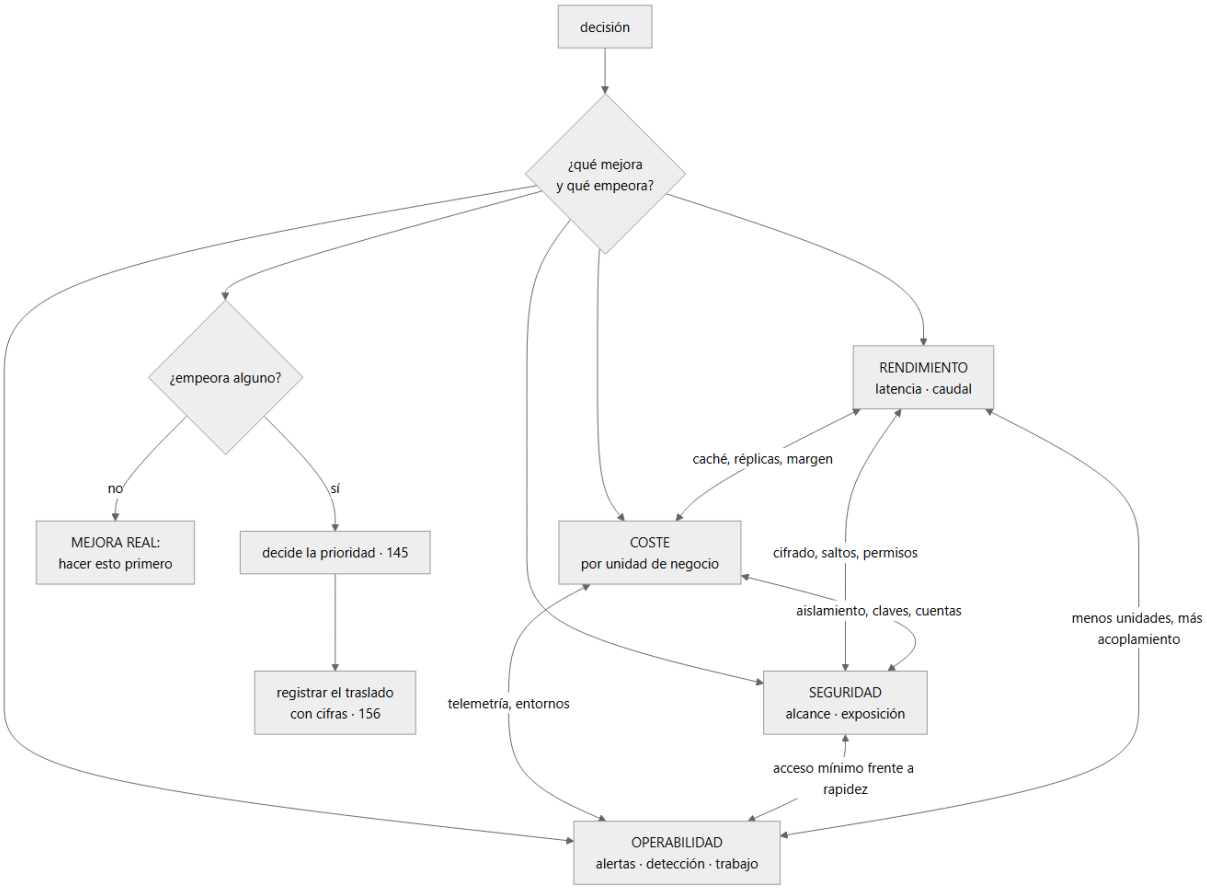
Concepto	Comprensión verificable
traslado	Cambio que mejora un atributo empeorando otro. Es lo que es casi toda decisión de arquitectura.
mejora real	Cambio que mejora dos o más atributos sin empeorar ninguno. Existe, es escaso y casi siempre consiste en quitar algo.
cuatro columnas	Formato mínimo de una decisión: qué mejora, cuánto, qué empeora, cuánto, y qué prioridad decide.
coste oculto de operar	Lo que una decisión añade en canalizaciones, alertas, procedimientos y guardia. Es el atributo que menos se cuantifica.
prioridad declarada	El orden de atributos de la clase 145. Es lo que resuelve el conflicto sin recurrir a la autoridad.
decisión revisable	La que registró su traslado y sus premisas, de modo que se pueda comprobar si siguen siendo ciertas.

Modelo mental

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

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. Casi todo es un traslado

Las tensiones, cada una con el mecanismo concreto de este programa que la produce:

```
RENDIMIENTO ←→ COSTE
  caché, réplicas de lectura, margen hasta el codo, cercanía geográfica
  → todo lo que hace más rápido cuesta capacidad          clases 111, 129

RENDIMIENTO ←→ SEGURIDAD
  cifrado por campo, autenticación mutua, comprobación de permisos,
  saltos de intermediario                                  clases 136, 152
  → 1,2 ms por llamada no es nada, hasta que hay ocho llamadas

RENDIMIENTO ←→ OPERABILIDAD
  menos unidades desplegables es más fácil de operar y acopla más
  más unidades permite escalar por separado y multiplica el trabajo
                                                           clase 148

COSTE ←→ SEGURIDAD
  aislamiento por cliente, claves por ámbito, cuentas separadas,
  entornos separados                                       clases 133, 154
  → el aislamiento cuesta aprovechamiento

COSTE ←→ OPERABILIDAD
  telemetría, entornos, retención de registros
  → saber lo que pasa cuesta dinero: llegó a ser el 70 % del cómputo
                                                           clase 132

SEGURIDAD ←→ OPERABILIDAD
  acceso mínimo y temporal frente a rapidez para intervenir
  inmutabilidad frente a poder depurar                     clases 134, 141
```

Y la afirmación que ordena la clase:

```
casi ninguna decisión de arquitectura es una mejora
```

```
es un TRASLADO entre estos cuatro
```

Y el problema de no decirlo:

```
se presenta como mejora
nadie sabe qué empeoró ni cuánto
no se puede discutir, porque no hay nada que comparar
y cuando el atributo empeorado duele, nadie relaciona una cosa
  con la otra
```

Y el caso característico, que este programa ya vio dos veces:

```
«hemos separado en microservicios: ahora escala mejor»
  mejora    escalado independiente de dos partes
  empeora   latencia (7 fronteras), operabilidad (15 canalizaciones
            y guardias), coste, y consistencia
  y nadie lo escribió                                  clase 148

«hemos añadido observabilidad completa»
  mejora    detección
  empeora   coste: 6.410 € frente a 9.200 € de cómputo    clase 132
```

Y el formato mínimo que lo evita, en cuatro columnas:

```
QUÉ MEJORA        con qué cifra
QUÉ EMPEORA       con qué cifra
QUÉ PRIORIDAD DECIDE   del orden de la clase 145
QUÉ HARÍA REVISARLO    qué premisa, si cambia, invalida la decisión
```

La cuarta es la que convierte una decisión en revisable, y es la materia de la clase 156.

2. Medir los cuatro

Un traslado solo se puede discutir si los dos lados tienen número. Las cuatro columnas se miden así:

```
RENDIMIENTO
  latencia por percentil, medida en el borde              clase 126
  caudal sostenido y distancia al codo                    clase 129
  llamadas de red por operación de negocio                clase 152

COSTE
  coste por unidad de negocio, no absoluto                clase 142
  y desglosado por servicio

SEGURIDAD
  alcance desde cada punto de entrada                     clase 133
  accesos permanentes a lo sensible                       clase 134
  hallazgos expuestos y su antigüedad                     clases 138, 139
  caminos hasta objetivos críticos                        clase 140

OPERABILIDAD
  alertas por turno y proporción accionable               clase 125
  tiempo hasta detectar y hasta mitigar                   clase 127
  trabajo repetitivo como proporción del tiempo           clase 128
  unidades desplegables por persona de guardia            clase 148
```

Y la última fila merece énfasis porque es la que menos se cuantifica y la que más duele después:

```
COSTE OCULTO DE OPERAR
  cada unidad nueva añade
    una canalización con sus puertas                      clase 100
    unos objetivos e indicadores                          clase 126
    unas alertas y unos procedimientos                    clases 125, 128
    una capacidad que medir                               clase 129
    y alguien que la entienda a las tres de la madrugada
```

Y una forma práctica de ponerle número: horas de mantenimiento al mes por unidad, medidas, no estimadas. Con eso, una propuesta de cuatro servicios nuevos deja de ser gratis en la conversación.

Y dos advertencias sobre las medidas, que vienen de la parte 10:

```
la medida que se convierte en objetivo se alcanza sin mejorar   ley 17
  → los cuatro se publican juntos, nunca uno solo

lo que compensa un fallo lo vuelve invisible                    ley 19
  → un traslado puede quedar oculto porque algo lo absorbe
```

```
  → el ejemplo financiero: la elasticidad convirtió una avería
    en una factura                                          clase 142
```

3. Cuando no hay traslado

Hay un conjunto pequeño de cambios que mejoran dos o más atributos sin empeorar ninguno. Son el trabajo más rentable que existe y casi siempre consisten en quitar algo.

```
QUITAR LO QUE NADIE USA
  recursos huérfanos, series de métricas, permisos, campos, alertas
  mejora   coste, seguridad y operabilidad a la vez
  empeora  nada                                    clases 134, 142

REDUCIR SALTOS DE RED
  menos fronteras por operación
  mejora   latencia, coste, modos de fallo y trazabilidad
  empeora  el acoplamiento, solo si se une lo que no debía unirse
                                                    clases 148, 152

MENOS ALERTAS, MEJOR ELEGIDAS
  mejora   detección y operabilidad
  empeora  nada; se detectó más con menos              clase 125

ELIMINAR UN SECRETO
  identidad federada en vez de credencial
  mejora   seguridad y operabilidad (nada que rotar)
  empeora  nada                                        clase 137

ARREGLAR UNA CONSULTA POR ELEMENTO
  mejora   latencia y coste
  empeora  nada                                        clase 124

ESTRUCTURAR EL REGISTRO EN UNA LÍNEA ANCHA
  mejora   coste (÷5) y capacidad de investigar
  empeora  nada                                        clase 122
```

Y la regla que se deduce:

```
antes de discutir un traslado, agota los cambios que no lo son
→ casi siempre queda alguno, y son los que menos se proponen
  porque no lucen
```

Y su explicación: casi todos consisten en retirar algo que existe sin motivo, que es la ley 20 vista desde el lado de la mejora.

Y el caso contrario, que también conviene reconocer: traslados que se hacen en la dirección equivocada porque nadie miró el orden de prioridad.

```
se añade cifrado por campo a datos que no lo necesitan
  empeora   latencia y operabilidad (no se puede buscar ni ordenar)
  mejora    nada, si esos campos no son sensibles

se separan dos módulos que cambian siempre juntos
  empeora   latencia, coste, operabilidad y consistencia
  mejora    nada de lo que estaba priorizado

se guarda telemetría durante 90 días en caliente
  empeora   coste
  mejora    una capacidad de investigar que nadie usa pasados 14 días
```

Los tres tienen la misma forma: se paga un traslado por un beneficio que no está en la lista de prioridades.

4. Decidir y dejar constancia

El método, en cinco pasos:

```
1. ENUNCIAR la decisión y las opciones reales
   dos o tres; si hay una sola, no es una decisión

2. MEDIR los cuatro atributos para cada opción
   con cifras, aunque sean estimaciones con su orden de magnitud

3. COMPROBAR si alguna opción no tiene traslado
   y si la hay, elegirla y terminar
```

```
4. RESOLVER el conflicto con el orden de prioridad     clase 145
   no con la opinión ni con la antigüedad de quien la sostiene

5. REGISTRAR la decisión con su traslado y sus premisas
```

Y el paso 4 es el que evita las discusiones circulares: si el orden está escrito, el conflicto ya está resuelto de antemano. Si no lo está, cada decisión reabre la discusión completa.

Y el paso 5 necesita un formato mínimo, que la clase 156 desarrolla:

```
qué se decidió
qué opciones se consideraron
qué mejora y qué empeora, con cifras
qué prioridad lo decidió
qué premisas se dieron por ciertas
y qué haría revisarlo
```

La última línea es la que más valor tiene a los dos años:

```
«esta decisión se revisa si el tráfico se multiplica por cuatro»
«si aparece un tercer lenguaje»                          clase 152
«si el equipo pasa de doce personas»                     clase 145
«si la campaña de noviembre deja de celebrarse»          clase 142
```

Y una advertencia sobre lo que ocurre cuando esto no se hace, y que es la observación con la que empieza la clase 156:

```
los diagramas se mantienen unos meses y luego divergen
las decisiones no quedan escritas en ningún sitio
y a los dos años nadie sabe POR QUÉ está así
→ y entonces nadie se atreve a cambiarlo, por si acaso
```

Y dos cifras que conviene vigilar sobre el propio proceso:

```
decisiones registradas frente a decisiones tomadas
decisiones revisadas al cumplirse su premisa de revisión
```

La segunda es la que dice si el registro sirve o es un archivo muerto.

Y la lista de comprobación de la clase:

- cada decisión enuncia qué mejora y qué empeora, con cifras
- los cuatro atributos se miden y se publican juntos
- el coste de operar está cuantificado en horas por unidad y mes
- antes de aceptar un traslado se han agotado las mejoras sin traslado
- el conflicto se resuelve con el orden de prioridad escrito
- ninguna decisión paga un traslado por algo que no está priorizado
- cada decisión registra sus premisas y qué haría revisarla
- se comprueba si las premisas siguen siendo ciertas

Y el cierre que enlaza con la clase siguiente: con esto está completo el material de la parte 12. La clase 156 revisa la arquitectura construida, deja el registro de decisiones como artefacto que sobrevive y califica las cinco predicciones de la clase 144, empezando por la que decía que la ley 14 dominaría esta parte.

Ejemplo trabajado

CloudShop repasa las seis decisiones de arquitectura tomadas en esta parte y escribe el traslado de cada una. Dos resultan ser mejoras sin coste, tres son traslados correctos y una estaba en la dirección equivocada.

Decisión 1: pasar de quince servicios a cinco unidades (clase 148).

```
MEJORA     latencia p99 del flujo de compra   840 ms → 210 ms
           coste operativo   15 canalizaciones → 5
           incidentes por fallo parcial        9/año → 2/año
           cambios en un solo repositorio       38 % → 91 %
EMPEORA    escalado independiente: ventas y soporte escalan juntos
           radio de un fallo de memoria: afecta a los dos módulos
CIFRA DEL PERJUICIO   soporte escala con ventas: +2 instancias
                      en campaña que no harían falta = 40 €/mes
PRIORIDAD  evolución y operabilidad, por encima de escalado fino
REVISAR SI soporte necesita escalar por su cuenta, o tiene equipo propio
```

Mejora en cuatro columnas y empeora en una, cuantificada en cuarenta euros. Con las cifras delante, la decisión dejó de ser discutible.

Decisión 2: descartar la malla (clase 152).

```
MEJORA     coste          620 €/mes evitados
           operabilidad   un plano de control menos
           latencia       1,2 ms por llamada evitados
EMPEORA    la política vive en 2 bibliotecas: actualizarla exige
           desplegar los 5 servicios
CIFRA DEL PERJUICIO   ~6 horas por actualización de política,
                      ~3 veces al año = 18 h/año
PRIORIDAD  coste y operabilidad; con 5 servicios y 2 lenguajes,
           el perjuicio es menor que el beneficio
REVISAR SI hay más de 15 servicios o un tercer lenguaje
```

Decisión 3: índice global en vez de local (clase 150).

```
MEJORA     latencia p99 de «mis pedidos»   410 ms → 21 ms
           lecturas por consulta                24 → 1
EMPEORA    escrituras por pedido                 1 → 2
           consistencia del índice     inmediata → ~200 ms
CIFRA DEL PERJUICIO   +6 % de coste de escritura; el desfase se cubre
                      con la garantía de sesión ya existente
PRIORIDAD  rendimiento del flujo del cliente
REVISAR SI la proporción de escrituras sube mucho
```

Decisión 4: cifrado por campo en tres campos (clase 136).

```
MEJORA     seguridad: esos campos son ilegibles para quien comprometa
           la aplicación
EMPEORA    no se pueden buscar ni ordenar por ellos
           +8 ms por lectura que los incluya
CIFRA DEL PERJUICIO   2 informes tuvieron que rehacerse
PRIORIDAD  corrección y protección de datos personales, atributo 1
REVISAR SI cambia el requisito normativo
```

Decisión 5: la que estaba en la dirección equivocada.

Se había cifrado por campo también la dirección de envío:

```
MEJORA     seguridad de un dato que ya estaba protegido por permisos,
           red y cifrado en reposo
EMPEORA    almacén no puede ordenar rutas por código postal
           el proceso de preparación descifra 12.000 registros/día
           +11 ms por lectura, +140 €/mes
PRIORIDAD  el dato es personal, sí, pero no está en la lista de
           categorías especiales, y ninguna norma lo exigía
```

Y el diagnóstico con el criterio del apartado tercero:

```
se pagó un traslado por un beneficio que NO estaba priorizado
→ se revirtió el cifrado por campo de la dirección
→ y se dejó el control de acceso y el cifrado en reposo, que era
  lo que la clasificación de la clase 141 pedía

efecto de revertirlo
  latencia de preparación        −11 ms
  coste                          −140 €/mes
  ordenación de rutas            recuperada
  seguridad                      sin cambio medible en el alcance
```

Las dos mejoras sin traslado.

```
RETIRAR 19 CAMPOS QUE NADIE USABA (clase 153)
  mejora   contrato más pequeño, menos que mantener y menos superficie
  empeora  nada; se comprobó con los contratos publicados

CORREGIR LAS 62 LLAMADAS DE «MIS PEDIDOS» (clase 152)
  mejora   latencia 2.100 ms → 190 ms
           coste: 59 llamadas menos por operación
           modos de fallo: 59 oportunidades menos de fallo parcial
  empeora  nada
```

Y el orden en que se hizo el trabajo, aplicando la regla del apartado tercero:

```
semana 1-2   las dos mejoras sin traslado
semana 3-8   los traslados, con sus cuatro columnas escritas
```

Y el efecto de ese orden:

```
mejora de latencia conseguida en las 2 primeras semanas    64 %
mejora conseguida en las 6 siguientes                      36 %
```

Casi dos tercios de la mejora vinieron de los cambios que no costaban nada, y se habrían pospuesto si se hubiera empezado por lo vistoso.

Las cuatro columnas, medidas para el conjunto de la parte.

```
RENDIMIENTO
  latencia p99 del flujo de compra           840 ms         210 ms
  llamadas de red por operación típica          7              2
  distancia al codo en el pico                71 %           43 %

COSTE
  coste por pedido                         0,057 €        0,041 €
  unidades desplegables                         15              5

SEGURIDAD
  alcance desde un servicio comprometido    2 de 15        2 de 5
  fugas entre clientes                          6              0
  caminos hasta objetivos críticos             14              2

OPERABILIDAD
  canalizaciones que mantener                   15              5
  unidades por persona de guardia             3,75           1,25
  horas de mantenimiento al mes por unidad      11             12
  trabajo repetitivo                          11 %            9 %
```

Y la fila de horas por unidad es la interesante: el mantenimiento por unidad no bajó; lo que bajó fue el número de unidades. Es la cifra que hace visible el coste oculto de operar.

El registro de decisiones, y su medida.

```
decisiones de arquitectura registradas            2             14
con las cuatro columnas                           0             14
con premisa de revisión escrita                   0             14
decisiones revisadas al cumplirse su premisa      —              3
  → la de la malla, al llegar a un tercer lenguaje: se mantuvo
  → la del compromiso de capacidad (clase 142)
  → la del cifrado de la dirección de envío
```

A los seis meses.

```
decisiones con traslado escrito                   0             14
decisiones revertidas por traslado mal dirigido   —              1
mejoras sin traslado identificadas antes de
empezar                                           0              2
proporción de la mejora que vino de ellas         —             64 %
los cuatro atributos publicados juntos            no             sí
horas de mantenimiento por unidad, medidas        no             sí
```

La lección que esta clase traslada a la parte 12: de seis decisiones, dos no tenían ningún coste y produjeron el 64 % de la mejora total, y se habrían pospuesto indefinidamente porque consistían en borrar campos y agrupar consultas. Y la única decisión que hubo que revertir no era técnicamente mala: era un traslado pagado por un beneficio que no estaba en la lista de prioridades, y solo se vio al escribir las cuatro columnas y comprobar que la columna de la mejora estaba vacía.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-12-cloud-native-distributed-architecture/155-rendimiento-costo-seguridad-y-operabilidad/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 tradeoff-analysis 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 tradeoff-analysis 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 decisión se presenta como mejora y nadie sabe qué empeoró	No se escribió el traslado	Escribe qué mejora y qué empeora, con cifras, y qué prioridad lo decide.
Cada decisión reabre la misma discusión	No hay orden de prioridad escrito, así que el conflicto se resuelve por opinión	Fija tres atributos que ganan cuando hay conflicto y aplícalos; es la clase 145.
Se paga complejidad por un beneficio que a nadie le importaba	El traslado mejora algo que no está priorizado	Comprueba que la columna de mejora contiene un atributo de la lista de prioridades; si no, revierte.
Se empieza por los cambios vistosos y la mejora tarda meses	No se buscaron primero los cambios que no tienen traslado	Agota las mejoras sin traslado —quitar lo que nadie usa, reducir saltos, corregir consultas por elemento— antes de aceptar ninguna.
Añadir servicios parece gratis en la conversación	El coste de operar no está cuantificado	Mide horas de mantenimiento al mes por unidad desplegable y úsalo en cada propuesta.
A los dos años nadie sabe por qué el sistema está así y nadie se atreve a cambiarlo	Las decisiones no dejaron constancia de sus premisas	Registra premisas y qué haría revisar cada decisión, y comprueba periódicamente si siguen siendo ciertas.

■ 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 casi ninguna decisión de arquitectura es una mejora?
2. ¿Qué cuatro columnas debe tener el enunciado de una decisión?
3. ¿Cómo se cuantifica el coste de operar y por qué importa?

4. ¿Qué caracteriza a los cambios que mejoran varios atributos a la vez?
5. ¿Qué convierte una decisión en revisable dentro de dos años?

Referencias

- Ford, N. y otros (2021). Software Architecture: The Hard Parts — compromisos explícitos y análisis de alternativas. <https://www.oreilly.com/library/view/software-architecture-the/9781492086888/>
- Bass, L. y otros (2021). Software Architecture in Practice, cap. 20 — evaluación de arquitecturas por atributos. <https://www.oreilly.com/library/view/software-architecture-in/9780136886051/>
- SEI (2025). ATAM: architecture tradeoff analysis method — puntos de compromiso y sensibilidad. <https://insights.sei.cmu.edu/library/architecture-tradeoff-analysis-method-collection/>
- Google SRE (2025). Balancing reliability, cost and velocity — los cuatro atributos publicados juntos. <https://sre.google/workbook/table-of-contents/>
- FinOps Foundation (2025). Unit economics as a decision input — coste por unidad como columna de la decisión. <https://www.finops.org/framework/capabilities/unit-economics/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 12 en PDF · Manual integral

Anterior	Índice	Siguiente
← 154 · Multi-tenancy, aislamiento y noisy neighbor	Parte 12 · Programa	156 · Proyecto: revisión de arquitectura con ADR →

Evaluación — 155 Rendimiento, costo, seguridad y operabilidad

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 tradeoff-analysis 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.

156 — Proyecto: revisión de arquitectura con ADR

Parte: 12 — Arquitectura cloud-native y sistemas distribuidos Nivel: avanzado-experto · Horas estimadas: 8
Laboratorio: capstone · Estado: EXECUTABLECORE

Propósito

Revisar la arquitectura construida y dejar el único artefacto que sobrevive a los cambios de equipo: el registro de por qué está así. La clase da un método de revisión que se puede ejecutar en medio día, el formato mínimo de una decisión y las cifras con las que se comprueba que el registro sirve. Y cierra la parte con las tres piezas de siempre: calificar las cinco predicciones de la clase 144, que esta vez salieron mejor de lo habitual y conviene explicar por qué; incorporar la ley que esta parte ha hecho aparecer cuatro veces; y escribir la predicción que la parte 13 tendrá que corregir.

Resultados de aprendizaje

Al finalizar podrás:

1. Ejecutar una revisión de arquitectura acotada, con resultado accionable.
2. Registrar una decisión con su traslado, sus premisas y qué la revisaría.
3. Medir si el registro se usa o es un archivo muerto.
4. Calificar las cinco predicciones de la clase 144 con evidencia.
5. Escribir la predicción de la parte 13 en términos que se puedan desmentir.

Conceptos centrales

Concepto	Comprensión verificable
registro de decisión	Documento corto escrito en el momento de decidir, con las opciones, el traslado, las premisas y qué haría revisarla.
revisión de arquitectura	Sesión acotada que compara los escenarios de calidad con el sistema real y produce cambios, no un informe.
divergencia	Distancia entre la arquitectura documentada y la real. Se mide, y siempre existe.
ley 21	El acoplamiento real está en quién puede escribir cada dato. Separar procesos sin separar escritores no separa nada.
premisa de revisión	Condición que, al cumplirse, obliga a reconsiderar una decisión. Es lo que la hace revisable.
hipótesis de la parte 13	Predicción escrita ahora sobre lo que ocurrirá cuando el sujeto sea usar varios proveedores y sobrevivir a la pérdida de uno.

Modelo mental

Una arquitectura es un conjunto de decisiones costosas de cambiar; su calidad se juzga contra escenarios concretos, no por la cantidad de servicios usados.

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. Revisar sin escribir un informe

Una revisión de arquitectura útil cabe en medio día y produce cambios, no un documento. Sus cinco pasos:

```
1. ¿SIGUEN SIENDO VÁLIDOS LOS ESCENARIOS?              clase 145
   se leen los 5-12 escenarios con sus cifras
   y se marca cuáles han cambiado
   → es lo que más se olvida y lo que más decisiones invalida

2. COMPARAR CON EL SISTEMA REAL, con datos
   no con el diagrama: con lo que dicen las trazas y los registros

3. MEDIR LOS CUATRO ATRIBUTOS                          clase 155

4. LISTAR DIVERGENCIAS Y DECIDIR QUÉ HACER CON CADA UNA
   corregir el sistema, corregir el documento, o aceptarla por escrito

5. REGISTRAR las decisiones tomadas
```

Y el paso 2 se hace con lo que ya existe, sin pedirle nada a nadie:

```
grafo de dependencias real, agregando trazas            clase 124
conexiones observadas entre servicios                   clase 135
quién escribe cada tabla, del registro de auditoría     clase 147
qué campos consume cada cliente                         clase 153
qué operaciones cruzan cuántas fronteras                clase 152
coste por servicio y por unidad                         clase 142
```

Y la divergencia siempre existe, y este programa la ha medido tres veces:

```
dependencias documentadas 23, observadas 41             clase 124
conexiones documentadas 23, observadas 58               clase 135
decisiones registradas 2, tomadas 14                    clase 155
```

Y lo que hay que hacer con cada divergencia:

```
el sistema está mal          → corregir el sistema
el documento está viejo      → corregir el documento
era una decisión implícita   → convertirla en explícita y registrarla
es deuda conocida            → aceptarla por escrito, con fecha de revisión
```

La tercera es la más frecuente: algo que nadie decidió y que funciona, y que conviene registrar antes de que alguien lo cambie sin saber por qué estaba así.

Y dos reglas sobre la dinámica de la sesión, con la disciplina de la clase 140:

```
están quienes construyeron el sistema
el resultado es una lista de cambios con dueño, no un acta
```

2. El registro de decisiones

Los diagramas se mantienen unos meses. Lo que sobrevive es lo que se escribió en el momento de decidir, porque no hay que mantenerlo: describe un instante.

El formato mínimo, en una página:

```
TÍTULO       una frase con la decisión
FECHA y ESTADO   propuesta · aceptada · sustituida por otra
CONTEXTO     qué problema hay y qué escenarios aplican      clase 145
OPCIONES     dos o tres reales; si hay una, no era decisión
DECISIÓN     cuál y por qué
TRASLADO     qué mejora y qué empeora, con cifras           clase 155
PRIORIDAD    cuál de los atributos lo decidió
PREMISAS     lo que se da por cierto
REVISAR SI   qué haría reconsiderarla
```

Y las tres últimas líneas son las que distinguen un registro útil de uno decorativo:

```
sin premisas    a los dos años nadie sabe si sigue teniendo sentido
sin traslado    parece que fue gratis, y no lo fue
sin «revisar si» nadie la revisa nunca
```

Y ejemplos de premisas de revisión de esta parte, que ya se han usado:

```
«si hay más de 15 servicios o un tercer lenguaje»          clase 152
«si soporte necesita escalar por su cuenta»                clase 148
«si el tráfico se multiplica por cuatro»                   clase 145
«si cambia el requisito normativo»                         clase 136
«si la campaña de noviembre deja de celebrarse»            clase 142
```

Y las reglas prácticas que hacen que el registro se mantenga vivo:

```
vive en el repositorio, junto al código
se escribe ANTES de implementar, no después
es inmutable: no se edita, se sustituye por otro que la reemplaza
se numera, y se enlaza desde el código cuando explica algo raro
una página; si necesita más, la decisión no está clara
```

La tercera importa más de lo que parece: una decisión editada pierde su valor histórico. Si cambia, se escribe otra que dice que sustituye a la anterior, y la anterior queda con su fecha y su contexto.

Y qué merece un registro y qué no:

```
sí   lo que costaría caro deshacer
     lo que alguien preguntará dentro de un año
     lo que se decidió en contra de lo obvio
     lo que se decidió NO hacer, y por qué
no   convenciones de estilo
     decisiones reversibles en una tarde
```

La cuarta de la primera lista es la más valiosa y la que menos se escribe: por qué NO se hizo lo que parecía evidente. Sin ella, alguien lo propondrá otra vez cada seis meses.

Y las dos medidas que dicen si el registro sirve:

```
decisiones registradas frente a tomadas
decisiones revisadas al cumplirse su premisa
```

La segunda, si es cero, significa que el registro es un archivo muerto.

3. Calificación de las cinco predicciones

Predicción 1: «la parte 12 no introducirá mecanismos nuevos; nombrará decisiones ya tomadas. Más de la mitad de sus clases formalizarán algo ya hecho».

```
veredicto: EQUIVOCADA por poco, y el error es lo interesante

clases que formalizan lo ya hecho                          5 de 11
  146  propiedades de una aplicación operable
  147  la frontera que ya decidían las claves de partición
  148  la granularidad que ya se había elegido sin criterio
  149  la consistencia que ya se elegía por operación sin decirlo
  153  los contratos de las clases 115 y 118

clases con material genuinamente nuevo                     4 de 11
  150  testigo de época, particiones fijas
  151  fallo gris, fallo metaestable, trabajo constante, celdas
  153  contratos dirigidos por el consumidor
  154  seis dimensiones de aislamiento

clases de encuadre, ni una cosa ni otra                    2 de 11
  145  atributos y escenarios
  155  el traslado entre cuatro atributos
```

Y lo que la predicción no vio: todo el material nuevo trata de lo mismo, y es lo que las partes anteriores no habían tocado:

```
cómo falla el CONJUNTO, y cómo se comprueba
  las partes 05 a 11 trataron el fallo de cada componente
  la 12 trata el fallo del sistema: gris, metaestable, por celdas
```

Predicción 2: «la ley 14 dominará, con más apariciones que en cualquier otra parte».

```
veredicto: ACERTADA en las dos mitades

ley 14 en la parte 12                                      5
  145  identificar qué decisiones serán irreversibles
  147  la propiedad de los datos, que después no se mueve
  148  dividir es fácil; unir no se propone nunca
  150  número de particiones y asignación de claves
  154  el nivel de aislamiento sin camino de migración

máximo anterior en una sola parte                          3  (parte 09)
otras leyes en la parte 12
  ley 20   3        ley 16   3        ley 13   2
```

Predicción 3: «la clase más difícil será la de dividir, y la división la deciden los datos y los equipos, no la tecnología».

```
veredicto: ACERTADA

en la extracción de un servicio
  semanas dedicadas al código                              4
  semanas dedicadas a los DATOS                            5
la causa real de los problemas de la división existente
  41 tablas con más de un escritor, no el número de servicios
la restricción que descartó la propuesta de 22 servicios
  el tamaño del equipo                                      clase 145
```

Predicción 4: «la arquitectura documentada no coincidirá con el sistema real».

```
veredicto: ACERTADA, con tres medidas independientes

dependencias documentadas 23, observadas 41              clase 124
conexiones documentadas 23, observadas 58                clase 135
decisiones registradas 2, tomadas 14                     clase 155
5 de 15 servicios no correspondían a ninguna frontera    clase 147
```

Predicción 5: «monolito o microservicios será la decisión menos importante; las consecuentes serán la propiedad de los datos, la consistencia por operación y el contrato».

```
veredicto: ACERTADA, y se puede formular mejor
```

Lo que la evidencia muestra no es solo que importe poco, sino algo más fuerte:

```
no es una decisión independiente
la determinan las otras tres
  quién escribe cada dato          → dónde puede estar la frontera
  qué consistencia exige cada operación → qué puede quedar separado
```

```
  qué contrato se puede sostener   → qué se puede desplegar aparte
→ una vez respondidas, cuántos procesos hay es casi mecánico
```

Y una observación sobre el conjunto, porque cuatro aciertos de cinco es mucho mejor que en las partes anteriores y conviene explicar por qué:

```
las predicciones de las partes 08 a 11 eran sobre materia nueva
las de la parte 12 eran sobre la SISTEMATIZACIÓN de lo ya vivido
→ predecir cómo se ordenará lo que uno ya ha hecho es mucho más fácil
  que predecir lo que aún no ha hecho
→ y por eso el acierto aquí vale menos que los errores de antes
```

4. La ley 21, el recuento y la hipótesis de la parte 13

```
LEY 21
  El acoplamiento real de un sistema está en quién puede escribir
  cada dato. Separar procesos sin separar escritores no separa nada.
```

Sus cuatro apariciones en esta parte:

```
clase 147   41 tablas con más de un escritor; 5 semanas por cambio
            de esquema y 3 equipos para tocar un descuento
clase 148   el monolito repartido: 11 de 15 servicios compartían tablas,
            y por eso el 41 % de los despliegues había que coordinarlos
clase 150   un solo escritor por partición es lo que permite razonar;
            dos líderes a la vez es el peor fallo posible
clase 154   el aislamiento entre clientes se decide en los datos:
            las 6 fugas estaban en consultas, no en despliegues
```

Y lo que añade al cuestionario:

```
¿quién puede escribir este dato? ¿cuántos son?
si separo estos dos procesos, ¿siguen escribiendo lo mismo?
¿qué tendría que pasar para que solo uno pudiera escribirlo?
```

Y su relación con la ley 14, que es su vecina:

```
ley 14   las decisiones de creación son irreversibles
ley 21   y la más irreversible de todas es quién escribe cada dato
```

Recuento tras la parte 12:

```
ley 13  el bucle que no corre no da error                        24
ley 15  una señal con demasiados elementos deja de ser señal     20
ley 16  un control que estorba acaba desactivado o rodeado       21
ley 14  las decisiones de creación son irreversibles             16
ley 11  lo que entra en un sistema de solo-añadir se queda        9
ley 20  lo que no tiene dueño no se apaga ni se corrige           8
ley 19  lo que compensa un fallo lo vuelve invisible              7
ley 17  la medida que se vuelve objetivo se alcanza sin mejorar   6
ley 18  lo asíncrono traslada la garantía, no la elimina          5
ley 21  el acoplamiento está en quién escribe                     4
        NUEVA en esta parte
```

La hipótesis de la parte 13. La parte siguiente trata de usar varios proveedores, de conectar lo propio con lo ajeno y de sobrevivir a perder una región o un proveedor entero. La predicción, escrita para poder desmentirla:

```
1. La mayoría de los motivos declarados para usar varios proveedores
   no resistirán la pregunta de la clase 145: «¿qué pasa si no?».
   → predigo que de los motivos habituales, MENOS DE LA MITAD
     sobrevivirán a esa pregunta, y que el que más sobrevive no será
     el que más se cita

2. El coste dominante de operar entre nubes no será el cómputo:
   será la SALIDA DE DATOS y la duplicación de lo que ya está
   resuelto en un proveedor.
   → predigo que la salida de datos aparecerá como partida de dos
     cifras porcentuales en el ejemplo de coste

3. La ley dominante será la 21, la recién estrenada, en su forma
   entre proveedores: quién puede escribir el dato decide si una
   configuración activa-activa es posible.
   → y predigo que la conclusión honesta de la parte será que
     activa-activa con escritura en los dos lados casi nunca compensa

4. Lo más difícil no será mover la carga: será la IDENTIDAD y la RED.
   → predigo que las clases 159 y 160 tendrán más problemas reales
```

```
     que las de portabilidad de carga

5. Y la predicción que puede salir del revés: los objetivos de
   recuperación declarados no se corresponderán con nada medido,
   igual que ocurrió con la copia de seguridad de la clase 088
   y con el punto de recuperación de la clase 150.
   → predigo que, al medirlos, el tiempo real será entre 3 y 10 veces
     el declarado
```

Y lo que se anota para calificar sin trampa:

```
lo que ya sabemos    que lo declarado y lo medido difieren, dos veces
lo que creemos       que el problema es de identidad y red, no de carga
lo que no sabemos    si existe algún caso en que activa-activa con
                     escritura en ambos lados sí compense
```

Ejemplo trabajado

CloudShop ejecuta la revisión de arquitectura sobre lo construido en esta parte y monta el registro de decisiones. La revisión dura cinco horas y encuentra tres divergencias, una de las cuales invalidaba una decisión tomada seis meses antes.

Paso 1: los escenarios, revisados.

```
escenarios escritos en la clase 145                              8
siguen siendo válidos                                            6
cambiaron                                                        2
  E1  el pico de referencia bajó de 5.000/s a 2.100/s
      → la campaña dejó de celebrarse                  clase 142
  E5  tres clientes exigen ahora datos en su propia región
      → escenario reescrito                            clase 154
```

Y el primero invalidó una decisión anterior:

```
decisión de hace 6 meses   dimensionar para 5.000 peticiones/s
capacidad reservada        para ese pico
coste asociado             1.400 €/mes de margen permanente
con el pico real de 2.100/s, el margen sobra
→ decisión sustituida; ahorro 900 €/mes
```

Una decisión correcta en su momento y cara hoy, detectada solo porque los escenarios llevaban fecha.

Paso 2: el sistema real, con datos.

```
unidades desplegables                        5            5     ✓
dependencias entre unidades                  7            9     ✗
tablas con más de un escritor                0            2     ✗
campos del contrato en uso                  22           22     ✓
fronteras que cruza «confirmar pedido»       2            2     ✓
operaciones síncronas                       12           12     ✓
```

Y las dos divergencias:

```
DIVERGENCIA 1: dos dependencias no documentadas
  el módulo de soporte llamaba directamente al servicio de pagos
  motivo   una urgencia de hace 4 meses
  decisión corregir el sistema: pasa por ventas, como estaba diseñado

DIVERGENCIA 2: dos tablas con dos escritores
  la tabla de reservas la escribían almacén y un trabajo nocturno
  motivo   el trabajo nocturno se escribió antes de la clase 147
  decisión corregir el sistema: el trabajo publica un hecho y almacén
           escribe                                          ley 21
```

Y una tercera que se aceptó por escrito:

```
DIVERGENCIA 3: el adaptador de correo sigue siendo una función
  aunque el volumen ha crecido y ya no es tan irregular
  cuenta de la clase 117   sigue compensando, por poco
  decisión   aceptada, con revisión cuando supere 200 envíos/hora
```

Paso 3: los cuatro atributos.

```
latencia p99 del flujo de compra              840 ms        190 ms
coste por pedido                            0,057 €       0,038 €
alcance desde un servicio comprometido      2 de 15        2 de 5
```

```
caminos hasta objetivos críticos                14            2
unidades por persona de guardia               3,75         1,25
horas de mantenimiento al mes por unidad        11           12
trabajo repetitivo                            11 %          9 %
```

El registro de decisiones, montado.

```
decisiones tomadas en la parte 12                              14
registradas en su momento                                       0
registradas retroactivamente durante la revisión               14
  → con fecha real, marcadas como reconstruidas
tiempo empleado                                             3 h 20
```

Y el efecto en los seis meses siguientes:

```
decisiones registradas frente a tomadas       0 de 14      19 de 19
escritas antes de implementar                     —        17 de 19
sustituidas por otra posterior                    —              3
revisadas al cumplirse su premisa                 —              4
preguntas de «¿por qué está así?» respondidas
con el registro                                   —             11
```

Y las cuatro revisadas por premisa:

```
«si el pico baja de 3.000/s»              se cumplió → dimensionado
«si aparece un tercer lenguaje»           se cumplió → malla revisada
                                                       y mantenida
«si soporte necesita escalar solo»        no se cumplió
«si cambia el requisito normativo»        se cumplió → región propia
                                                      para 3 clientes
```

Y una decisión de las que más valen, del tipo «por qué NO se hizo lo obvio»:

```
registro 007   «no separamos el motor de precios en un servicio»
contexto       es la parte más compleja y lo pedía el equipo
decisión       módulo con interfaz estricta
motivo         ninguno de los seis criterios de la clase 148 aplicaba
revisar si     tiene equipo propio o perfil de escala distinto

veces que se ha vuelto a proponer en 6 meses                    2
veces que la discusión se cerró leyendo el registro             2
tiempo empleado en cada una                                ~10 min
```

Sin ese registro, cada propuesta habría reabierto una discusión de horas.

El estado de la parte 12, completo.

```
escenarios de calidad escritos                   0             8
atributos priorizados                            0             3
unidades desplegables                           15             5
contextos con frontera y escritor único         no             5
tablas con más de un escritor                   41             0
fronteras que cruza una operación típica         7             2
latencia p99 del flujo de compra             840 ms        190 ms
consistencia decidida por operación         0 de 20       20 de 20
punto de recuperación medido                    no            sí
protección contra dos líderes, ensayada         no            sí
fallo gris: tiempo de detección              32 min          70 s
clientes afectados por un vecino ruidoso        190             3
fugas entre clientes                              6             0
roturas de contrato                         11 / año           1
decisiones registradas                            2            19
```

Lo que la parte 12 no resolvió, dicho con claridad.

```
el motor de precios sigue siendo un módulo, y si el equipo crece
  habrá que extraerlo con el coste que eso tiene
las dos personalizaciones que no cupieron en el catálogo
  se resolvieron pasando un cliente a nivel dedicado, que es caro
y la divergencia entre lo documentado y lo real vuelve a aparecer
  en cuanto se deja de medir: la revisión es periódica, no única
```

La conclusión que cierra la parte 12: la revisión encontró tres divergencias en cinco horas, y la más cara no era un fallo del sistema sino de las premisas: se seguía dimensionando para un pico que había dejado de existir catorce meses antes, y eso costaba novecientos euros al mes. Y el artefacto que más valor produjo en los seis meses siguientes no fue ningún diagrama: fue un registro de una página explicando por qué NO se separó el motor de precios, que cerró en

diez minutos dos discusiones que antes habrían durado horas.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-12-cloud-native-distributed-architecture/156-proyecto-revision-de-arquitectura-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 architecture-review 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 architecture-review 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 de arquitectura produce un informe que nadie lee	Se comparó el sistema con el diagrama en vez de con datos, y no salieron cambios con dueño	Compara con trazas, conexiones observadas y escritores reales, y sal con una lista de cambios.
A los dos años nadie sabe por qué el sistema está así	Las decisiones no se registraron en el momento	Registro de una página por decisión, escrito antes de implementar, inmutable y viviendo junto al código.
La misma propuesta se rediscute cada seis meses	No se registró por qué NO se hizo lo que parecía obvio	Registra también las decisiones negativas, con su motivo y su premisa de revisión.
El registro existe y nadie lo revisa	Falta la línea de qué haría revisar cada decisión	Escribe la premisa de revisión y mide cuántas decisiones se revisan al cumplirse.
Se mantiene una decisión correcta que hoy es cara	Los escenarios cambiaron y nadie los revisó	Empieza toda revisión comprobando si los escenarios de calidad siguen siendo válidos.
Se separan procesos y el sistema sigue igual de acoplado	Ley 21: los escritores siguen siendo los mismos	Comprueba quién puede escribir cada dato; sin separar escritores, no hay separación.

■ 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 cinco pasos tiene una revisión de arquitectura y con qué datos se hace el segundo?
2. ¿Qué tres líneas distinguen un registro de decisión útil de uno decorativo?
3. ¿Por qué se registran también las decisiones de no hacer algo?
4. ¿Qué dice la ley 21 y en qué se diferencia de la ley 14?
5. ¿Qué predice la hipótesis de la parte 13 sobre los motivos del multi-nube y sobre los objetivos de recuperación?

Referencias

• Nygard, M. (2011). Documenting architecture decisions — formato y motivo del registro de decisiones. <https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions>

• SEI (2025). Lightweight architecture evaluation — revisión acotada con resultado accionable. <https://insights.sei.cmu.edu/library/>

• Ford, N. y otros (2017). Building Evolutionary Architectures — funciones de aptitud y revisión continua. <https://evolutionaryarchitecture.com/>

• ADR GitHub organization (2025). Architecture decision records: templates and tooling — plantillas y prácticas. <https://adr.github.io/>

• Ford, N. y otros (2021). Software Architecture: The Hard Parts, cap. 15 — registrar compromisos y revisarlos. <https://www.oreilly.com/library/view/software-architecture-the/9781492086888/>

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

Descargar: Parte 12 en PDF · Manual integral

Anterior	Índice	Siguiente
← 155 · Rendimiento, costo, seguridad y operabilidad	Parte 12 · Programa	157 · Motivaciones y anti-patrones de multi-cloud →

Evaluación — 156 Proyecto: revisión de arquitectura 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 architecture-review 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.