Multi-Cloud Engineering Program

Parte 15 — Arquitectura de sistemas e ingeniería de requisitos

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	15 de 23
Clases	181–192
Evaluaciones	12
Horas estimadas	52

Índice

Parte 15 — Arquitectura de sistemas e ingeniería de requisitos

← Parte 14 · Índice completo · Parte 16 →

Descargar: Esta parte en PDF · Manual integral

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

Pasar de requisitos ambiguos a arquitecturas comprobables antes de elegir servicios cloud.

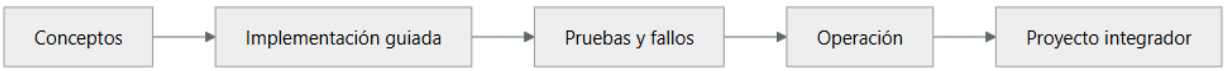
Resultados de la parte

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 sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

Secuencia



Clases

ID	Clase	Laboratorio	Horas
181	Requisitos funcionales, restricciones y atributos de calidad	architecture	4
182	Contexto, contenedores, componentes y código con C4	architecture	4
183	Acoplamiento, cohesión, modularidad y fronteras	architecture	4
184	Arquitectura monolítica, modular y de microservicios	decision	4
185	Disponibilidad, confiabilidad y análisis de puntos de fallo	reliability	4
186	Capacidad, latencia, throughput y teoría de colas	performance	4
187	Consistencia, particiones, relojes y consenso	distributed	4
188	Contratos de API, eventos y compatibilidad evolutiva	api	4
189	Modelado de amenazas y arquitectura de confianza cero	security	4
190	ADRs, fitness functions y gobierno de decisiones	architecture	4
191	Architecture review y comunicación con stakeholders	governance	4
192	Proyecto: arquitectura completa de CloudShop	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

- Software Architecture in Practice — Bass, Clements y Kazman.
- Fundamentals of Software Architecture — Richards y Ford.
- Designing Data-Intensive Applications — Martin Kleppmann.

181 — Requisitos funcionales, restricciones y atributos de calidad

Parte: 15 — Arquitectura de sistemas e ingeniería de requisitos Nivel: intermedio-avanzado · Horas estimadas: 4
Laboratorio: architecture · Estado: EXECUTABLECORE

Propósito

Separar tres cosas que casi siempre llegan mezcladas en la misma frase: lo que el sistema debe hacer, lo que lo condiciona sin negociación posible, y lo bien que debe hacerlo. La distinción no es académica: los requisitos funcionales casi nunca deciden la arquitectura, las restricciones la limitan antes de empezar, y los atributos de calidad son los que la determinan y los que se pagan todos los meses.

Resultados de aprendizaje

Al finalizar podrás:

1. Distinguir requisito funcional, restricción y atributo de calidad.
2. Escribir atributos de calidad como escenarios medibles y no como adjetivos.
3. Detectar las restricciones reales, incluidas las que nadie declara.
4. Resolver los conflictos entre atributos diciendo cuál cede.
5. Priorizar por coste de cambio y no por importancia declarada.

Conceptos centrales

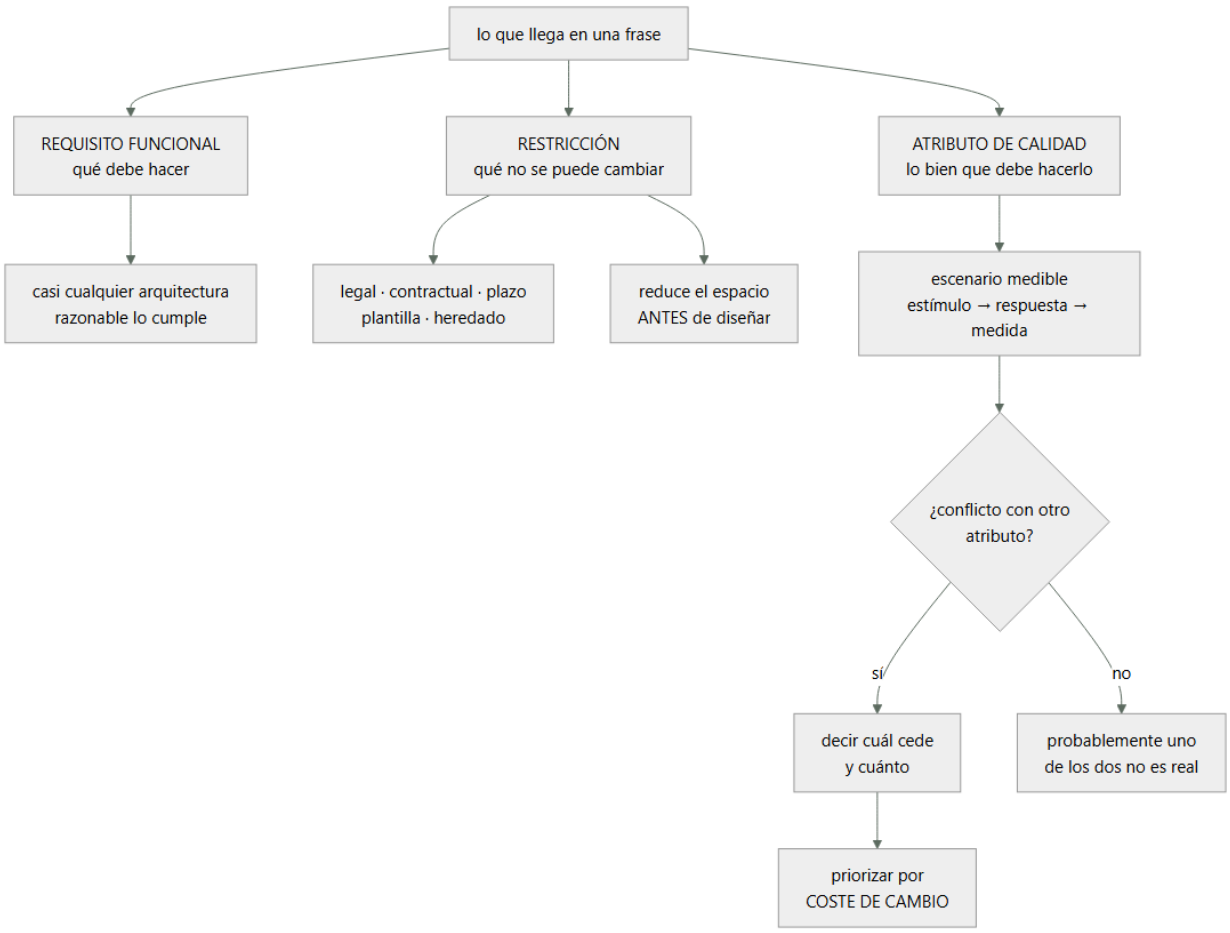
Concepto	Comprensión verificable
requisito funcional	Lo que el sistema debe hacer. Casi cualquier arquitectura razonable puede cumplirlo.
restricción	Decisión ya tomada que no se puede cambiar: legal, contractual, de plazo, de plantilla o técnica heredada.
atributo de calidad	Lo bien que el sistema debe hacer lo que hace. Es lo que decide la arquitectura.
escenario de calidad	Forma medible de un atributo: estímulo, fuente, entorno, respuesta y medida de respuesta.
conflicto entre atributos	Situación en que mejorar uno empeora otro. Se resuelve diciendo cuál cede y cuánto.
coste de cambio	Lo que costaría revertir una decisión más tarde. Es el criterio de prioridad, no la importancia declarada.

Modelo mental

Una arquitectura es un conjunto de decisiones sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

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

■ Flujo de razonamiento



Desarrollo

1. Las tres cosas que llegan mezcladas

Una frase típica de arranque de proyecto contiene las tres, sin marcarlas:

```
«necesitamos un sistema de reservas rápido, que aguante el Black
 Friday, en Azure porque ya tenemos el acuerdo, y para septiembre»

  sistema de reservas           REQUISITO FUNCIONAL
  rápido                        atributo de calidad, sin medir
  que aguante el Black Friday   atributo de calidad, sin medir
  en Azure, acuerdo firmado     RESTRICCIÓN
  para septiembre               RESTRICCIÓN
```

Y la observación que ordena todo lo demás:

```
el requisito funcional casi nunca decide la arquitectura
→ «reservar una habitación» se puede hacer con un monolito,
  con veinte servicios o con una hoja de cálculo

lo que decide la arquitectura es
  cuántas reservas por segundo en el pico
  cuánto puede tardar
  cuánto puede estar caído
  cuánto cuesta cambiarlo dentro de dos años
```

Las restricciones merecen atención aparte porque reducen el espacio de diseño antes de empezar y porque la mitad no se declaran:

```
DECLARADAS
  proveedor por contrato
  región por normativa                              clase 177
  plazo con fecha externa (campaña, ley, cliente)
  presupuesto anual
```

```
NO DECLARADAS, y suelen pesar más
  el equipo son cuatro personas y dos se van en marzo
  nadie ha operado nunca una base distribuida
  el sistema heredado de facturación no se puede tocar
  hay una guardia y no habrá una segunda
  la organización no aprueba nada en agosto
```

Y la regla práctica:

```
una restricción no se discute: se comprueba que es real
→ «tiene que ser Azure» → ¿contrato o costumbre?
→ «para septiembre» → ¿qué pasa el 1 de octubre si no está?

si la respuesta es «nada en concreto», no era una restricción
```

Y el error simétrico, más caro: tratar como negociable algo que no lo es. Un plazo legal o un requisito de residencia no cede porque el diseño sea elegante.

2. Atributos de calidad: del adjetivo al escenario

«Rápido», «escalable», «seguro» y «fiable» no son requisitos: son adjetivos. Un adjetivo no se puede cumplir ni incumplir, y por eso siempre se da por cumplido.

La forma útil es el escenario, con seis partes:

```
FUENTE       quién o qué provoca el estímulo
ESTÍMULO     qué ocurre
ENTORNO      en qué condiciones (normal, pico, degradado)
ARTEFACTO    sobre qué parte del sistema
RESPUESTA    qué debe hacer el sistema
MEDIDA       con qué número se comprueba
```

Y el mismo requisito, antes y después:

```
ANTES   «el buscador tiene que ser rápido»

DESPUÉS
  fuente      un usuario final desde la aplicación móvil
  estímulo    busca disponibilidad por ciudad y fechas
  entorno     pico de campaña, 3.000 búsquedas por segundo
  artefacto   servicio de búsqueda y su almacén
  respuesta   devuelve resultados o un mensaje de degradación
  medida      p99 ≤ 400 ms medido en el borde; sin errores 5xx
              por encima del 0,1 %
```

Y la diferencia práctica es que el segundo se puede probar, incumplir y discutir. El primero no.

Los atributos que más decisiones cambian, y por qué:

```
MODIFICABILIDAD    lo que cuesta cambiar algo
  → es el único que se paga TODOS los meses
  → y el único que nadie escribe como escenario
  escenario: «añadir un método de pago nuevo debe poder hacerlo
              un equipo sin tocar el servicio de reservas, en
              menos de 5 días, sin coordinar despliegues»

DISPONIBILIDAD     cuánto puede estar caído, y cuánto se pierde
  → tiene techo por dependencias                      clase 185

RENDIMIENTO        latencia, caudal y su comportamiento cerca
                   del codo                          clase 186

SEGURIDAD          alcance desde cada punto de entrada  clase 133

OBSERVABILIDAD     cuánto se tarda en saber qué pasa
  escenario: «ante una subida de errores, un ingeniero de guardia
              debe poder identificar el servicio y el cambio
              responsable en menos de 10 minutos, sin acceso
              a producción»

OPERABILIDAD       cuánto cuesta operarlo con la plantilla real
COSTE              coste por unidad de negocio          clase 142
```

Y dos escenarios que casi nunca se escriben y que este programa ha demostrado que importan:

```
RECUPERABILIDAD  «tras un borrado accidental de la tabla de
                  reservas, restaurar con pérdida ≤ 5 min y
                  servicio recuperado en ≤ 1 h, MEDIDO»  clase 166

DIAGNOSTICABILIDAD «un procedimiento de incidente debe poder
                   ejecutarlo alguien que no lo escribió»  ley 22
```

3. Los conflictos, y quién cede

Los atributos de calidad se estorban entre sí. Un documento que los lista todos como «altos» no ha decidido nada.

Los conflictos que aparecen siempre:

```
DISPONIBILIDAD × CONSISTENCIA
  bajo partición hay que elegir                        clase 187
  → decidir POR OPERACIÓN, no para el sistema entero

RENDIMIENTO × MODIFICABILIDAD
  cada capa de indirección cuesta latencia
  cada atajo cuesta cambio futuro

SEGURIDAD × OPERABILIDAD
  cada control añade fricción; la fricción se rodea    ley 16

COSTE × DISPONIBILIDAD
  la segunda región cuesta aunque no se use            clase 164

RENDIMIENTO × COSTE
  el codo se aleja pagando                             clase 129

MODIFICABILIDAD × PLAZO
  el atajo entrega en septiembre y se paga en marzo
```

Y la forma de resolverlos que evita la discusión infinita:

```
1. escribir los dos escenarios en conflicto, medidos
2. decir cuál cede y CUÁNTO cede, con número
3. decir quién lo decide, con nombre
4. y en qué condición se revisa

ejemplo
  «la disponibilidad del catálogo cede a favor del coste:
   99,5 % en lugar de 99,9 %, sin segunda región. Lo acepta
   la dirección de producto. Se revisa si el catálogo pasa a
   soportar reservas directas.»
```

Y una prueba de si el conflicto es real:

```
si nadie cede, uno de los dos atributos no era real
→ o bien nadie ha medido lo que cuesta el que se declara alto
```

Y la asimetría que conviene tener presente:

```
los atributos que se incumplen ruidosamente (latencia, caídas)
se corrigen pronto porque duelen
los que se incumplen en silencio (modificabilidad, recuperabilidad,
diagnosticabilidad) se descubren años después           ley 13
→ por eso hay que escribir escenarios precisamente de esos
```

4. Priorizar por coste de cambio

La priorización habitual —importancia declarada— produce documentos donde todo es crítico. La priorización útil usa otro eje:

```
¿cuánto cuesta cambiar esta decisión dentro de un año?

CARO DE CAMBIAR, decidir con cuidado y pronto
  modelo de datos y quién escribe cada dato            ley 21
  clave de partición                                    clase 114
  frontera entre módulos o servicios                    clase 183
  modelo de consistencia por operación                  clase 187
  dominio de identidad                                  clase 159
  jerarquía de cuentas                                  clase 169
  contrato público de la API                            clase 188

BARATO DE CAMBIAR, decidir después y con datos
```

```
tamaño de instancia
lenguaje de un servicio interno
proveedor de correo
biblioteca de registro
umbral de una alerta
```

Y la regla que se deriva:

```
decide pronto y con cuidado lo caro de cambiar
decide tarde y con medidas lo barato
→ y no gastes la discusión al revés, que es lo habitual
```

El documento mínimo de requisitos que hace falta antes de diseñar, y que cabe en dos páginas:

1. qué hace el sistema, en una lista corta
2. restricciones comprobadas, con la evidencia de que lo son
3. cinco a ocho escenarios de calidad medibles
   incluidos uno de modificabilidad y uno de recuperabilidad
4. conflictos declarados, con quién cede y cuánto
5. decisiones caras de cambiar, marcadas
6. lo que NO va a hacer el sistema

Y el punto 6 es el más útil y el que más se omite:

```
decir lo que el sistema no hará evita
  el atributo de calidad que aparece en el mes seis
  la integración que nadie pidió y todos asumieron
  y la discusión de alcance sin árbitro
```

Y la lista de comprobación de la clase:

- ■ cada frase del encargo está clasificada en una de las tres
- ■ cada restricción se ha comprobado: contrato, ley o fecha real
- ■ están las restricciones no declaradas: plantilla, guardia, heredado
- ■ ningún atributo de calidad es un adjetivo
- ■ cada escenario tiene medida, punto de medida y entorno
- ■ hay un escenario de modificabilidad
- ■ hay uno de recuperabilidad y uno de diagnosticabilidad
- ■ los conflictos dicen cuál cede, cuánto y quién lo acepta
- ■ las decisiones caras de cambiar están marcadas
- ■ está escrito lo que el sistema NO va a hacer

Y el cierre que enlaza con la clase siguiente: con requisitos, restricciones y atributos escritos, hace falta una forma de dibujar el sistema que sirva para discutirlos sin ahogarse en detalle. Es la materia de la clase 182.

Ejemplo trabajado

Un equipo recibe el encargo de rehacer el sistema de reservas. Lo que sigue es la conversación de arranque, la clasificación de cada frase, los escenarios que salieron y los dos conflictos que hubo que resolver.

Lo que dijo negocio, literalmente, en la reunión de arranque:

1. «queremos que la gente pueda reservar y modificar sin llamar»
2. «tiene que ser rápido, la web actual es lentísima»
3. «no se puede caer en Black Friday como el año pasado»
4. «seguro, que estamos con datos de tarjetas»
5. «tiene que estar en octubre por la campaña de invierno»
6. «usad Azure, tenemos compromiso de consumo hasta 2028»
7. «que sea escalable, para crecer»
8. «y que podamos añadir productos nuevos rápido»

La clasificación:

```
1  funcional        reservar y modificar en autoservicio
2  calidad          rendimiento, sin medir
3  calidad          disponibilidad en pico, sin medir
4  calidad          seguridad, sin medir
5  restricción      fecha externa — se comprueba
6  restricción      contractual — se comprueba
7  ruido            «escalable» sin cifra no dice nada
8  calidad          modificabilidad — la más importante y la
                    que iban a olvidar
```

Comprobación de las restricciones, que cambió una de las dos:

```
5  «octubre por la campaña»
```

```
   ¿qué pasa el 1 de noviembre si no está?
   → la campaña se lanza igual con el sistema actual, pero se
     pierde la promoción cruzada: unos 190.000 € de margen
   → ES una restricción real, con coste conocido

6  «Azure hasta 2028»
   → contrato con compromiso de consumo, penalización si no se
     alcanza. Real y no negociable.
   → efecto: descarta comparar proveedores; NO descarta que
     el equipo diseñe portable donde salga barato    clase 158
```

Y dos restricciones no declaradas que aparecieron al preguntar:

```
el equipo son 5 personas, y 2 están al 50 % en el heredado
no hay guardia nocturna y no la va a haber este año
→ esto elimina de la mesa cualquier diseño que exija
  intervención humana de madrugada
```

Los siete escenarios que se escribieron.

```
QA-1  RENDIMIENTO
  un usuario móvil busca disponibilidad, en pico de campaña
  (3.000 búsquedas/s), y recibe resultados
  medida   p99 ≤ 400 ms en el borde; errores 5xx ≤ 0,1 %

QA-2  DISPONIBILIDAD
  el flujo de reserva sigue aceptando reservas durante el pico
  aunque el servicio de recomendaciones esté caído
  medida   99,9 % mensual sobre el flujo de reserva;
           las recomendaciones son dependencia blanda

QA-3  MODIFICABILIDAD
  producto quiere añadir un método de pago nuevo
  medida   un equipo lo entrega en ≤ 5 días laborables, sin
           modificar el servicio de reservas y sin coordinar
           despliegue con otros equipos

QA-4  SEGURIDAD
  un atacante obtiene la credencial del servicio de búsqueda
  medida   alcance limitado al índice de búsqueda; sin acceso
           a datos de pago ni a la base de reservas;
           detectado en ≤ 15 min

QA-5  RECUPERABILIDAD
  un despliegue borra por error la tabla de reservas
  medida   restaurar con pérdida ≤ 5 min y servicio en ≤ 1 h,
           MEDIDO en un ensayo antes de salir a producción

QA-6  DIAGNOSTICABILIDAD
  suben los errores en el flujo de pago un martes a las 03:00
  medida   el ingeniero de guardia identifica servicio y cambio
           responsable en ≤ 10 min, desde el móvil, sin acceso
           de escritura a producción

QA-7  COSTE
  el tráfico crece un 40 % en campaña
  medida   el coste por reserva no sube más de un 10 %
```

Y dos observaciones sobre esta lista:

```
QA-3 no lo pidió nadie con esas palabras: salió de «añadir
  productos nuevos rápido», que es la frase que más arquitectura
  decide y la que menos se escribe

QA-6 salió de la restricción no declarada: sin guardia nocturna,
  el diagnóstico tiene que ser posible desde el móvil, y eso
  condiciona la telemetría desde el primer día        clase 121
```

Los dos conflictos, resueltos con nombre y número.

```
CONFLICTO 1   QA-1 (p99 ≤ 400 ms) × QA-3 (modificabilidad)

  el camino rápido sería una consulta que junta reservas,
  precios y disponibilidad en una sola base
  el camino modificable separa los tres, y añade una llamada
```

```
  cede   QA-1, de 400 a 500 ms de p99
  motivo el usuario no distingue 400 de 500; el equipo sí
         distingue 5 días de 6 semanas para añadir un producto
  acepta la dirección de producto
  revisa si la medida en el borde supera 700 ms

CONFLICTO 2   QA-2 (99,9 %) × QA-7 (coste por reserva)

  99,9 % en el flujo de reserva exigía segunda región activa
  el cálculo de dependencias daba un techo de 99,82 %  clase 185

  cede   QA-2, de 99,9 % a 99,7 %, con segunda región en frío
         y conmutación ensayada
  motivo la segunda región activa costaba 6.400 €/mes para
         ganar 0,2 puntos; el coste de la indisponibilidad
         medido en el histórico era de 1.900 €/mes
  acepta la dirección de tecnología
  revisa si el negocio de empresa (contratos con penalización)
         supera el 15 % de los ingresos
```

Las decisiones marcadas como caras de cambiar:

```
quién escribe la reserva                      ← una sola cosa
clave de partición de reservas                ← fecha + destino
frontera entre reservas, precios y catálogo
consistencia del inventario                   ← fuerte al reservar
contrato público de la API móvil
```

Y lo que el sistema no va a hacer, escrito para evitar la discusión del mes seis:

```
no sustituye a facturación, que sigue en el heredado
no gestiona la atención al cliente
no soporta reservas de grupo (más de 9 habitaciones)
no tiene aplicación de escritorio propia
no se internacionaliza más allá de ES y PT en esta fase
```

La lección que esta clase deja para la siguiente: de las ocho frases del encargo, una era funcional, dos eran restricciones, cuatro eran adjetivos que hubo que convertir en escenarios y una —«escalable»— no significaba nada. Y el escenario que más condicionó el diseño, QA-3, no lo pidió nadie con esas palabras: se dedujo de una frase suelta al final de la reunión.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-15-systems-architecture-engineering/181-requisitos-funcionales-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
El documento de requisitos no ayuda a decidir nada	Solo contiene requisitos funcionales, que casi cualquier arquitectura cumple	Añade escenarios de calidad medibles; son los que deciden la arquitectura.
Todos los atributos figuran como «altos» y no se puede diseñar	No se declararon los conflictos ni quién cede	Escribe los pares en conflicto, di cuál cede, cuánto, quién lo acepta y cuándo se revisa.
Una supuesta restricción resulta ser una costumbre	No se comprobó si era contractual, legal o de fecha real	Pregunta qué ocurre si se incumple; si la respuesta es «nada en concreto», no era una restricción.
El diseño exige intervención humana de madrugada y no hay quien la haga	Se ignoraron las restricciones no declaradas: plantilla, guardia, heredado	Levanta explícitamente equipo, turnos y sistemas intocables antes de diseñar.
A los dos años cada cambio pequeño cuesta semanas	No se escribió ningún escenario de modificabilidad	Escribe uno con verbo, plazo y sin coordinación entre equipos, y prioriza por coste de cambio.
En el mes seis aparece un alcance que nadie había acordado	No se escribió lo que el sistema no iba a hacer	Incluye la lista de exclusiones en el documento de requisitos.

■ 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 los requisitos funcionales casi nunca deciden la arquitectura?
2. ¿Cómo se comprueba que una restricción es real?
3. ¿Qué seis partes tiene un escenario de calidad?
4. ¿Qué atributo se paga todos los meses y casi nunca se escribe?
5. ¿Por qué se prioriza por coste de cambio y no por importancia declarada?

Referencias

- Bass, L., Clements, P. y Kazman, R. (2021). Software Architecture in Practice, 4.ª ed. — escenarios de atributos de calidad. <https://www.oreilly.com/library/view/software-architecture-in/9780136886051/>
- SEI (2006). Quality Attribute Workshops, 3.ª ed. <https://insights.sei.cmu.edu/library/quality-attribute-workshops-qaws-third-edition/>
- Ford, N., Parsons, R. y Kua, P. (2017). Building Evolutionary Architectures — características arquitectónicas y su priorización. <https://www.oreilly.com/library/view/building-evolutionary-architectures/9781491986356/>

- ISO/IEC 25010 — modelo de calidad de producto software. <https://www.iso.org/standard/78176.html>
- AWS (2025). Well-Architected Framework — atributos y compromisos entre pilares. <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 15 en PDF · Manual integral

Anterior	Índice	Siguiente
← 180 · Capstone: defensa, portafolio y plan profesional	Parte 15 · Programa	182 · Contexto, contenedores, componentes y código con C4 →

Evaluación — 181 Requisitos funcionales, 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.

182 — Contexto, contenedores, componentes y código con C4

Parte: 15 — Arquitectura de sistemas e ingeniería de requisitos Nivel: intermedio-avanzado · Horas estimadas: 4
Laboratorio: architecture · Estado: EXECUTABLECORE

Propósito

Dibujar un sistema de forma que sirva para discutir decisiones y no para decorar una presentación. El modelo C4 da cuatro niveles de zoom —contexto, contenedores, componentes y código— y su valor no está en las cajas sino en dos disciplinas: un nivel por conversación y cada flecha dice quién llama a quién, con qué y por qué. La clase enseña además qué dibujar en cada nivel, qué no, y cómo evitar que el diagrama envejezca mal.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir el nivel de zoom adecuado a la conversación que se está teniendo.
2. Dibujar contexto y contenedores con el detalle justo y ninguno más.
3. Anotar las flechas con protocolo, dirección y motivo.
4. Detectar los diagramas que mienten: los que envejecen y los que ocultan.
5. Mantener los diagramas vivos con el mínimo esfuerzo posible.

Conceptos centrales

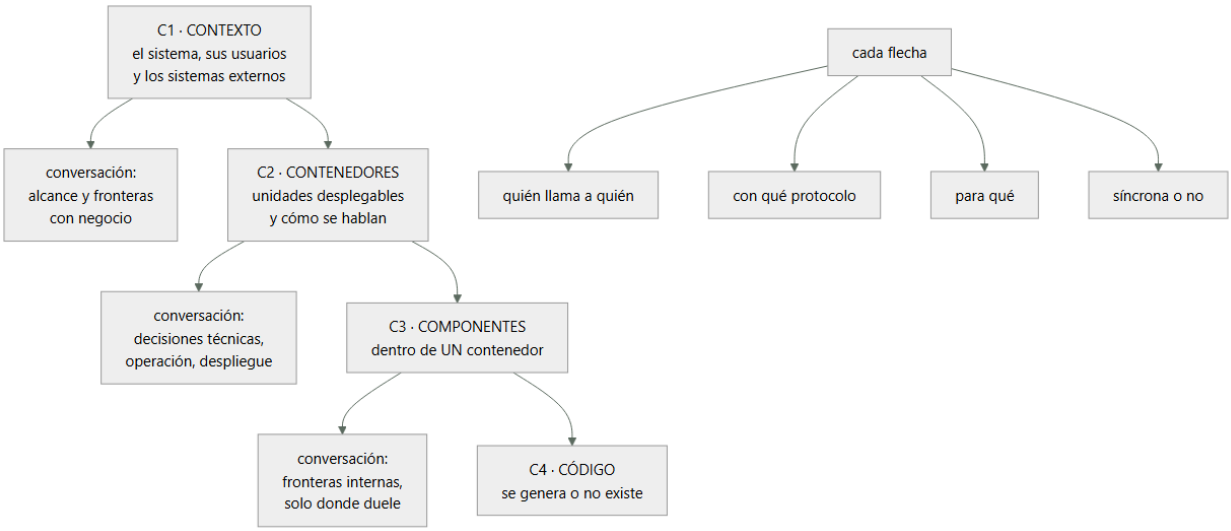
Concepto	Comprensión verificable
contexto (C1)	El sistema como una caja, sus usuarios y los sistemas externos con los que habla.
contenedor (C2)	Unidad desplegable y ejecutable por separado: aplicación, servicio, base de datos, cola, función.
componente (C3)	Agrupación lógica dentro de un contenedor. Se dibuja solo cuando hace falta discutirla.
código (C4)	Clases y relaciones. Casi nunca se dibuja a mano: se genera o no existe.
flecha anotada	Relación con dirección, protocolo, motivo y, si importa, sincronía y volumen.
diagrama vivo	El que se actualiza porque alguien lo usa. El resto miente en cuanto cambia algo.

Modelo mental

Una arquitectura es un conjunto de decisiones sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

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

■ Flujo de razonamiento



Desarrollo

1. Cuatro niveles, una conversación cada uno

El error que arruina la mayoría de los diagramas es mezclar niveles: un dibujo con usuarios, servicios, tablas y clases a la vez no sirve para ninguna conversación.

```
C1  CONTEXTO        ¿qué hay dentro y qué hay fuera?
    audiencia       negocio, dirección, equipos vecinos
    contiene        el sistema como UNA caja
                    personas que lo usan, con su papel
                    sistemas externos, con quién los opera
    NO contiene     tecnologías, bases de datos, colas

C2  CONTENEDORES    ¿de qué piezas desplegables consta?
    audiencia       el equipo, operación, seguridad
    contiene        aplicaciones, servicios, almacenes, colas
                    la tecnología de cada uno
                    las relaciones, anotadas
    NO contiene     clases, capas internas, detalles de esquema

C3  COMPONENTES     ¿cómo está organizado ESTE contenedor?
    audiencia       quien va a tocarlo
    se dibuja       solo cuando hay una discusión de frontera
    NO se dibuja    «por completitud», para todos los servicios

C4  CÓDIGO          clases y relaciones
    se genera desde el código, o no se hace
```

Y la regla que ahorra la mitad del trabajo:

```
casi todo el valor está en C1 y C2
C3 solo donde hay una decisión que discutir
C4 casi nunca
```

Qué es un contenedor y qué no, porque es donde más se confunde:

```
SÍ es contenedor          se despliega y ejecuta por separado
  una aplicación móvil
  una API
  un trabajo programado
  una base de datos
  una cola o un tema
  un almacén de objetos con lógica asociada

NO es contenedor
  una biblioteca compartida
  un módulo interno
  una capa
  un «dominio»
```

Y una prueba práctica: si no se puede reiniciar por separado, no es un contenedor.

2. Las flechas son el contenido

Las cajas se adivinan; las flechas no. Un diagrama con cajas correctas y flechas sin anotar no dice nada que no supiéramos.

Cada relación lleva, como mínimo:

```
DIRECCIÓN    quién inicia la llamada
             ← no «se comunican», que oculta el acoplamiento
PROTOCOLO    HTTPS/JSON, gRPC, SQL, mensaje en cola
MOTIVO       «para consultar disponibilidad», no «datos»
```

Y cuando importa, tres anotaciones más que cambian decisiones:

```
SÍNCRONA O NO      una llamada síncrona hereda la disponibilidad
                   del llamado                          clase 185
DURA O BLANDA      ¿el llamante sigue funcionando sin ella?
VOLUMEN            peticiones por segundo, o por operación
                   ← revela las consultas por elemento   clase 124
```

Y una anotación que este programa ha demostrado que vale más que todas:

```
¿QUIÉN ESCRIBE?
  marcar en el diagrama qué contenedor ESCRIBE cada almacén
  → si hay dos flechas de escritura al mismo almacén, hay
    acoplamiento oculto y el diagrama acaba de encontrarlo  ley 21
```

Y el error de dirección más caro:

```
dibujar «A ↔ B» porque «se hablan»
→ esconde quién depende de quién
→ y por tanto esconde qué se cae cuando cae uno
```

Lo que un diagrama de contenedores debe permitir responder sin preguntar a nadie:

```
¿qué se cae si cae esto?
¿por dónde entra el tráfico externo?
¿dónde vive cada dato y quién lo escribe?
¿qué cruza una frontera de red o de cuenta?
¿qué es síncrono en el camino crítico?
¿qué se despliega junto y qué por separado?
```

Y si no las responde, el diagrama es decorativo por muy bonito que sea.

3. Los diagramas que mienten

Hay tres formas de mentir con un diagrama correcto, y todas son frecuentes:

```
1. EL QUE ENVEJECIÓ
   se dibujó al empezar y nadie lo tocó
   → miente en cuanto se añade un servicio
   señal   no coincide con lo que hay desplegado

2. EL QUE OCULTA
   omite lo feo: el trabajo programado sin dueño, la conexión
   directa a la base del heredado, el acceso manual
   → y esas son precisamente las piezas que causan incidentes
   señal   el diagrama es más limpio que la realidad

3. EL QUE DIBUJA EL DESEO
   representa la arquitectura objetivo como si fuese la actual
   → y las decisiones se toman sobre algo que no existe
   señal   nadie sabe decir qué parte ya está
```

Y la disciplina que los evita:

```
marcar SIEMPRE el estado de cada elemento
  existe · en construcción · previsto · a retirar

y tener dos diagramas cuando haga falta, nunca uno mezclado
  ACTUAL   lo que hay hoy, con lo feo incluido
  OBJETIVO lo que se quiere, con fecha
```

Cómo mantenerlos vivos sin que se convierta en una tarea que nadie hace:

```
el diagrama vive en el repositorio, como texto
  → mermaid, PlantUML o similar; se revisa en el mismo cambio

se actualiza en el cambio que lo invalida, no «luego»
  → si un cambio añade un contenedor y no toca el diagrama,
    la revisión lo pide

se contrasta contra la realidad de vez en cuando
  → el inventario de recursos y el mapa de dependencias
    observadas dicen la verdad                          clase 122
  → y lo que aparezca en el mapa y no en el diagrama es
    exactamente lo que hay que mirar                    ley 15
```

Y el contraste automático, que es barato y encuentra mucho:

```
lista de servicios desplegados      vs   contenedores del C2
dependencias observadas             vs   flechas del C2
almacenes con más de un escritor    vs   flechas de escritura
→ las diferencias son hallazgos, no errores del diagrama
```

4. Cuándo dibujar y cuándo no

Dibujar tiene coste, y hacerlo por costumbre produce documentación que nadie lee.

```
DIBUJA CUANDO
  hay que decidir una frontera                          clase 183
  hay que explicar el alcance a alguien de fuera
  entra gente nueva y hay que orientarla
  se discute qué se cae si cae algo                     clase 185
  se revisa la seguridad y hace falta ver el alcance    clase 189
  se prepara una migración

NO DIBUJES
  «para tener documentación»
  el C3 de todos los servicios
  el C4 a mano
  lo que el código ya dice mejor
```

Y una forma de usar los niveles que funciona bien en reuniones:

```
empieza SIEMPRE en C1, aunque todos lo conozcan
  → alinea el alcance en dos minutos y evita media hora de
    malentendido
baja a C2 y quédate ahí
  → es el nivel donde se toman casi todas las decisiones
baja a C3 solo si la discusión lo pide, y solo del contenedor
  en cuestión
```

Y dos vistas complementarias que no son C4 y suelen faltar:

```
VISTA DE DESPLIEGUE   dónde corre cada contenedor: cuenta,
                      región, zona, red                 clase 169
                      → responde «¿qué cae si cae una zona?»

VISTA DE DATOS        quién escribe y quién lee cada almacén
                      → responde «¿dónde está el acoplamiento?»
```

Y la lista de comprobación de la clase:

- ■ cada diagrama tiene un solo nivel de zoom
- ■ el C1 no contiene tecnologías
- ■ cada elemento del C2 se despliega por separado
- ■ toda flecha tiene dirección, protocolo y motivo
- ■ las llamadas síncronas del camino crítico están marcadas
- ■ está marcado quién ESCRIBE cada almacén
- ■ no hay ninguna flecha bidireccional sin explicar
- ■ cada elemento tiene estado: existe, previsto o a retirar
- ■ actual y objetivo están separados
- ■ el diagrama está en el repositorio, como texto
- ■ se ha contrastado con las dependencias observadas

Y el cierre que enlaza con la clase siguiente: el diagrama de contenedores hace visibles las fronteras, pero no dice si están bien puestas. Decidir dónde va cada frontera —y por qué el criterio no es funcional— es la materia de la clase 183.

Ejemplo trabajado

El equipo de reservas de la clase 181 dibuja su sistema actual antes de rediseñarlo. Lo que sigue es el C1, el C2 con las flechas anotadas, el contraste contra la realidad —que encontró cinco cosas que no estaban en el dibujo— y el único C3 que se dibujó.

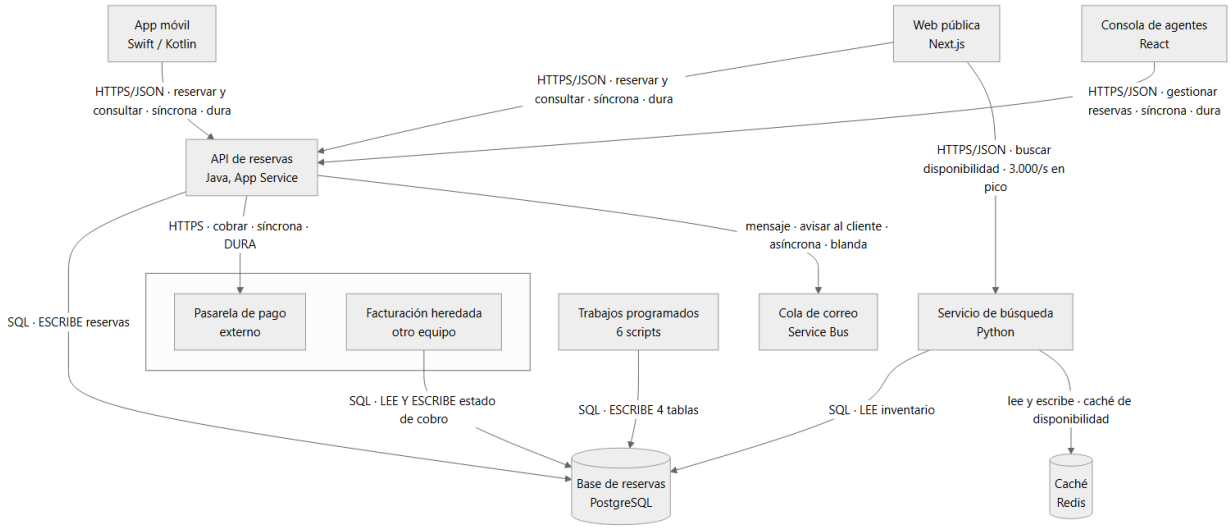
C1 · Contexto. Una caja, cuatro actores, tres sistemas externos:

```
PERSONAS
  Cliente final           reserva y modifica
  Agente de call center   reserva por teléfono y resuelve
  Gestor de hotel         actualiza inventario y precios

SISTEMA
  [Plataforma de reservas]

EXTERNOS
  Pasarela de pago        externa, contrato con SLA de 99,95 %
  Facturación heredada    interna, otro equipo, no se toca
  Proveedor de correo     externo
```

Y una decisión de alcance que el C1 dejó clara en dos minutos:

```
facturación está FUERA de la caja
→ y por tanto no se rediseña, se integra
→ negocio creía que entraba en el proyecto          clase 181
```

C2 · Contenedores, tal como estaba antes del rediseño.



Y lo que el diagrama hizo evidente en cuanto se anotó quién escribe:

```
tres contenedores ESCRIBEN la base de reservas
  la API, los trabajos programados y facturación heredada
→ acoplamiento por escritura, invisible hasta dibujarlo   ley 21
→ y explica por qué cualquier cambio de esquema requería
  coordinar con un equipo que no participa en el proyecto

el pago es una dependencia DURA y síncrona en el camino de
reserva
→ el techo de disponibilidad del flujo no puede superar el
  99,95 % de la pasarela                                  clase 185
```

El contraste contra la realidad, hecho con el inventario de recursos y el mapa de dependencias observadas:

```
EN LA REALIDAD Y NO EN EL DIBUJO
  1  un segundo caché, en una máquina virtual sin etiquetas,
     usado por la consola de agentes
  2  un trabajo programado nº 7 que nadie recordaba: exportaba
     un CSV a un almacén compartido con marketing
  3  una conexión directa de la herramienta de informes a la
     réplica de lectura
```

```
  4  la app móvil llamaba también a un endpoint antiguo
     (api-v1) que se creía retirado
  5  la pasarela de pago llamaba de vuelta a un webhook que
     no estaba dibujado

EN EL DIBUJO Y NO EN LA REALIDAD
  0
```

Y el hallazgo más caro fue el 5, por lo que implicaba:

```
el webhook de la pasarela era un punto de ENTRADA externo
no dibujado, y por tanto
  → no estaba en el modelo de amenazas               clase 189
  → no tenía alerta de ausencia                       ley 13
  → y cuando falló en marzo, 41 pagos quedaron cobrados
    y sin reserva confirmada durante 3 días
```

El único C3 que se dibujó, y por qué se dibujó ese:

```
motivo   se discutía si el cálculo de precios debía salir de
         la API de reservas a su propio contenedor

C3 de la API de reservas
  Controlador de reservas       entrada HTTP
  Motor de disponibilidad       consulta inventario
  Calculadora de precios        reglas, promociones, impuestos
  Coordinador de pago           habla con la pasarela
  Repositorio de reservas       único que escribe la tabla
  Publicador de eventos         escribe en la cola

lo que la discusión necesitaba ver
  la calculadora de precios NO tocaba el repositorio
  la llamaban 3 de los 6 componentes
  y cambiaba 4 veces al mes, frente a 1 vez al trimestre
  el resto
```

Y la decisión que salió de ese C3, con el criterio de la clase siguiente:

```
sale a su propio contenedor
motivo   ritmo de cambio muy distinto y sin escritura compartida
coste de cambio si nos equivocamos   medio: una llamada más
         y volver a juntarlo es mecánico
```

El diagrama objetivo, marcado con estado para no mentir:

```
API de reservas               existe
Servicio de precios           previsto        Q4
Servicio de catálogo          en construcción
Trabajos programados          a retirar       4 de 7
endpoint api-v1               a retirar       tras migrar la app
acceso de facturación a la BD a retirar       sustituir por evento
segundo caché sin etiquetas   a retirar       inmediato
```

La lección que esta clase deja: el diagrama que el equipo tenía era correcto y llevaba dos años sin tocarse. Al contrastarlo contra las dependencias observadas aparecieron cinco elementos reales que no estaban dibujados y ninguno dibujado que no existiera, y el más peligroso —un punto de entrada externo— llevaba tres años funcionando sin figurar en ningún sitio. El diagrama no era falso: era incompleto en la dirección que importa, que siempre es la misma.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-15-systems-architecture-engineering/182-contexto-contenedores-componentes-y-codigo-con-c4/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 c4-model 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 c4-model 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 diagrama no sirve para ninguna conversación	Mezcla niveles: usuarios, servicios, tablas y clases a la vez	Un nivel por diagrama; empieza en C1, decide en C2 y baja a C3 solo donde haya discusión.
El dibujo no dice qué se cae si cae algo	Las flechas no tienen dirección, sincronía ni dureza	Anota quién inicia, con qué protocolo, para qué, si es síncrona y si es dura o blanda.
Un cambio de esquema requiere coordinar con equipos ajenos y nadie sabía por qué	El diagrama no marcaba quién escribe cada almacén	Marca las flechas de escritura; dos escritores en un almacén son acoplamiento oculto.
El diagrama es más limpio que la realidad	Omite lo feo: trabajos sin dueño, accesos directos, endpoints antiguos	Contrástalo con el inventario y las dependencias observadas; lo que sobre en la realidad es el hallazgo.
Se toman decisiones sobre una arquitectura que no existe	El diagrama objetivo se presenta como el actual	Separa actual y objetivo, y marca el estado de cada elemento: existe, previsto o a retirar.
Los diagramas quedan obsoletos a las semanas	Viven en una herramienta aparte y actualizarlos es una tarea suelta	Guárdalos como texto en el repositorio y exige actualizarlos en el mismo cambio que los invalida.

■ 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 prueba decide si algo es un contenedor?
2. ¿Qué debe llevar como mínimo cada flecha, y qué tres anotaciones más cambian decisiones?
3. ¿Cuáles son las tres formas de mentir con un diagrama correcto?

4. ¿Cómo se contrasta un diagrama contra la realidad y qué significan las diferencias?
5. ¿Cuándo merece la pena dibujar un C3?

Referencias

- Brown, S. (2025). The C4 model for visualising software architecture. <https://c4model.com/>
- Brown, S. (2019). Software Architecture for Developers — niveles de abstracción y documentación mínima útil. <https://leanpub.com/software-architecture-for-developers>
- Clements, P. y otros (2010). Documenting Software Architectures, 2.ª ed. — vistas y su audiencia. <https://www.oreilly.com/library/view/documenting-software-architectures/9780132488617/>
- Structurizr (2025). Diagrams as code — diagramas versionados junto al código. <https://structurizr.com/>
- Mermaid (2025). Flowchart syntax — diagramas como texto en el repositorio. <https://mermaid.js.org/syntax/flowchart.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 15 en PDF · Manual integral

Anterior	Índice	Siguiente
← 181 · Requisitos funcionales, restricciones y atributos de calidad	Parte 15 · Programa	183 · Acoplamiento, cohesión, modularidad y fronteras →

Evaluación — 182 Contexto, contenedores, componentes y código con C4

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 c4-model 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.

183 — Acoplamiento, cohesión, modularidad y fronteras

Parte: 15 — Arquitectura de sistemas e ingeniería de requisitos Nivel: intermedio-avanzado · Horas estimadas: 4
Laboratorio: architecture · Estado: EXECUTABLECORE

Propósito

Decidir dónde va cada frontera, que es la decisión más cara de cambiar de todas las que se toman. La clase define acoplamiento y cohesión con precisión suficiente para discutirlos, enumera las siete formas de acoplamiento por orden de coste, y sostiene con la evidencia del programa que la frontera correcta la marca quién escribe cada dato y a qué ritmo cambia cada cosa, no la descomposición funcional que sale sola al mirar el organigrama.

Resultados de aprendizaje

Al finalizar podrás:

1. Distinguir los siete tipos de acoplamiento y ordenarlos por coste.
2. Medir la cohesión de un módulo con criterios comprobables.
3. Situar fronteras usando escritura de datos y ritmo de cambio.
4. Reconocer las fronteras mal puestas por sus síntomas.
5. Decidir cuándo mover una frontera y cuánto cuesta hacerlo.

Conceptos centrales

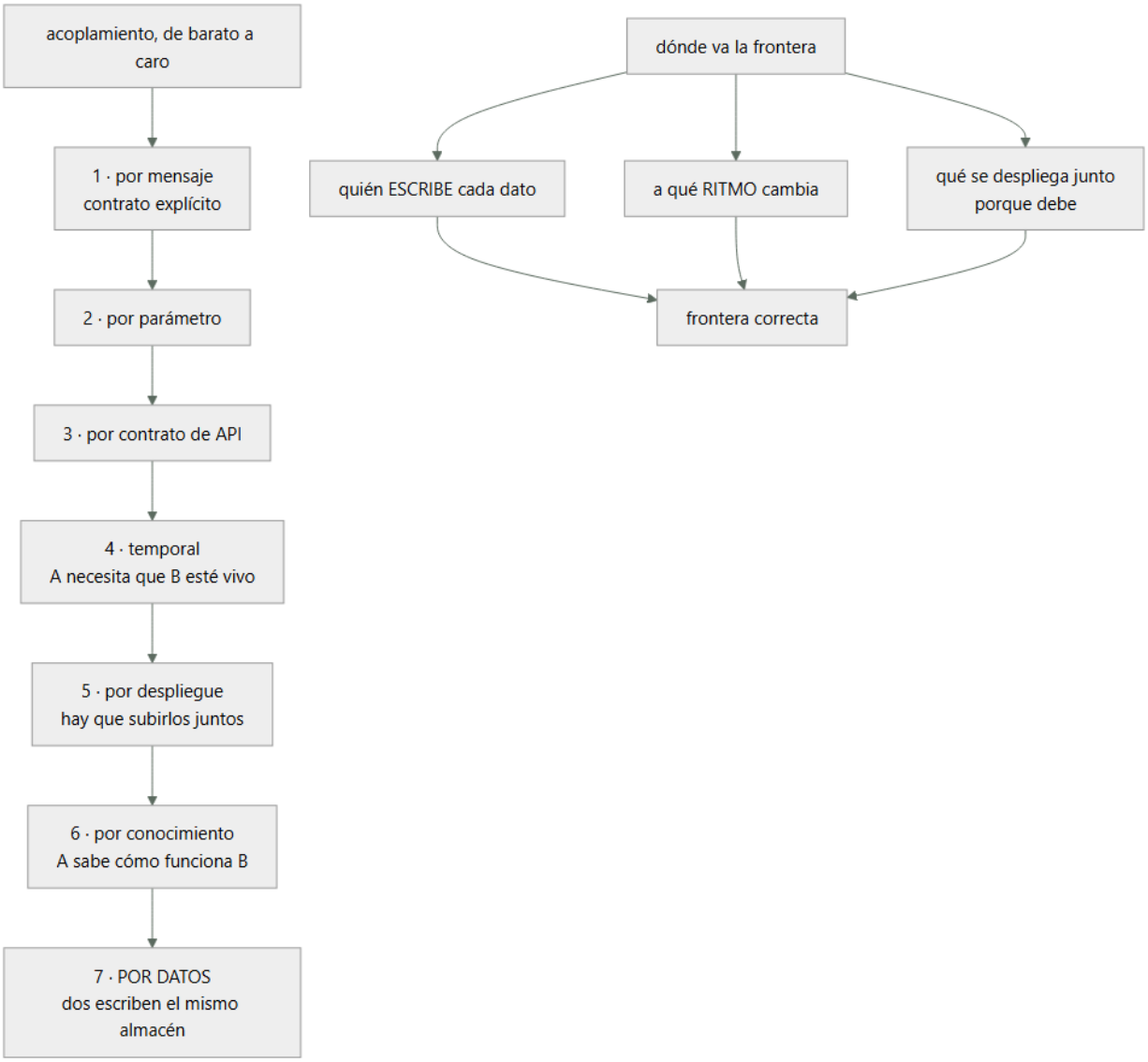
Concepto	Comprensión verificable
acoplamiento	Medida de cuánto obliga un cambio en A a cambiar B. No es malo por sí mismo; lo caro es el tipo equivocado.
cohesión	Medida de cuánto pertenecen juntas las cosas de un módulo. Alta cohesión: cambian por el mismo motivo.
acoplamiento por datos	Dos módulos escriben el mismo almacén. El más caro y el más invisible.
ritmo de cambio	Frecuencia con que se modifica una parte. Partes con ritmos muy distintos no pertenecen al mismo módulo.
frontera	Línea que separa lo que se despliega, versiona y decide por separado.
coste de mover una frontera	Lo que cuesta cambiarla más tarde. Determina cuánto cuidado merece la decisión inicial.

Modelo mental

Una arquitectura es un conjunto de decisiones sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

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. Acoplamiento: siete tipos, por coste

Decir «hay que reducir el acoplamiento» no ayuda: todo sistema útil está acoplado. Lo que importa es de qué tipo, porque el coste varía en dos órdenes de magnitud.

```
1  POR MENSAJE            A publica un evento; B lo consume
   coste de cambio        bajo; el contrato es explícito
   se rompe con           cambiar el esquema del evento  clase 148

2  POR PARÁMETRO          A llama a B pasando datos
   coste                   bajo

3  POR CONTRATO DE API    A depende de la forma de la API de B
   coste                   medio; gestionable con versiones

4  TEMPORAL               A necesita que B esté vivo AHORA
   coste                   medio-alto; hereda disponibilidad
   síntoma                 una caída de B tira A              clase 185

5  POR DESPLIEGUE         hay que subir A y B a la vez
   coste                   alto; elimina la autonomía
   síntoma                 «no puedo desplegar hasta que ellos»

6  POR CONOCIMIENTO       A sabe cómo funciona B por dentro
   coste                   alto; se rompe sin avisar
```

```
   síntoma                un cambio interno de B rompe A

7  POR DATOS             A y B escriben el mismo almacén
   coste                  el más alto de todos
   síntoma                nadie puede cambiar el esquema
   invisible              no aparece en el código de ninguno
```

Y la observación que este programa ha repetido desde la clase 147:

```
los seis primeros son visibles: están en el código
el séptimo NO, y por eso sobrevive años                 ley 21
→ dos servicios «independientes» que escriben la misma
  tabla no son dos servicios: son uno mal repartido
```

Y una consecuencia práctica que se usa como regla de diseño:

```
se puede convertir acoplamiento caro en barato
  7 → 1   un solo escritor, y los demás reciben eventos
  5 → 3   contrato versionado en vez de despliegue conjunto
  6 → 3   API explícita en vez de conocimiento interno
  4 → 1   asíncrono donde la operación lo permita   clase 118

y cada conversión cuesta latencia, consistencia o complejidad
→ no se hacen todas: se hacen donde el cambio duele
```

2. Cohesión: qué pertenece junto

La cohesión se explica mal con definiciones abstractas y bien con una pregunta:

```
¿estas dos cosas cambian por el mismo motivo?

sí   pertenecen juntas
no   no, aunque traten del mismo sustantivo
```

Y el error clásico es agrupar por sustantivo:

```
«todo lo de Cliente va en el servicio de Cliente»
  → datos de contacto        cambian por marketing
  → preferencias             cambian por producto
  → historial de pagos       cambian por finanzas
  → consentimientos          cambian por legal

cuatro motivos de cambio, cuatro equipos, un solo módulo
→ cada despliegue toca lo de los otros tres
```

Los criterios de cohesión que se pueden comprobar, no opinar:

```
1. MOTIVO DE CAMBIO
   mira los últimos 50 cambios del módulo
   → si vienen de tres áreas distintas de negocio, no hay cohesión

2. RITMO DE CAMBIO
   mide cuántas veces al mes cambia cada parte
   → si una cambia 8 veces y otra 1 al trimestre, sepáralas

3. QUIÉN LOS PIDE
   si los cambios vienen de equipos distintos, la frontera
   está en el sitio equivocado                          ley 21

4. QUÉ SE ROMPE JUNTO
   si un fallo en una parte no afecta a la otra, no comparten
   destino y podrían separarse
```

Y una medida directa que se saca del historial y casi nadie mira:

```
CAMBIOS ACOPLADOS
  ¿qué porcentaje de los cambios a A tocan también a B?

  > 60 %   A y B deberían estar juntos
  10-60 %  frontera dudosa, mírala
  < 10 %   frontera bien puesta

se calcula del historial del repositorio, en minutos
```

Y la asimetría que hay que recordar:

```
demasiados módulos   coste de coordinación, latencia,
```

```
                    operación                        clase 184
demasiado pocos     coste de coordinación humana
                    y despliegues bloqueados
→ el óptimo no es «más pequeño»; es «cambia junto, va junto»
```

3. Dónde va la frontera de verdad

La descomposición funcional —reservas, pagos, catálogo— sale sola, y casi siempre está a medio camino de la correcta. Los tres criterios que la corrigen, por orden de fuerza:

```
1. QUIÉN ESCRIBE CADA DATO                              ley 21
   un dato, un escritor
   → si dos módulos necesitan escribir lo mismo, o son uno
     solo o el dato está mal definido
   → este criterio, solo, coloca la mayoría de las fronteras

2. RITMO DE CAMBIO
   lo que cambia 8 veces al mes no vive con lo que cambia
   una vez al trimestre
   → aunque traten del mismo sustantivo

3. LO QUE DEBE FALLAR JUNTO
   si A no tiene sentido sin B, sepáralos con cuidado:
   estás creando acoplamiento temporal                  clase 185
```

Y tres criterios que se usan mucho y funcionan mal:

```
EL ORGANIGRAMA
  produce fronteras que reflejan la empresa de hoy
  → y la empresa se reorganiza antes que el software
  → aunque la ley de Conway es real: si el organigrama y la
    frontera se pelean, gana el organigrama                ley 21

EL SUSTANTIVO DE NEGOCIO
  «Cliente», «Producto», «Pedido»
  → agrupa cosas que cambian por motivos distintos

EL TAMAÑO
  «que ningún servicio pase de N líneas»
  → optimiza una medida sin relación con el coste    ley 17
```

Los síntomas de una frontera mal puesta, que se detectan sin discutir:

```
hay que desplegar dos cosas juntas para que funcione
un cambio pequeño requiere tocar tres repositorios
la mayoría de los cambios cruzan la frontera
dos equipos se bloquean en el mismo fichero
hay que preguntar a otro equipo qué significa un campo
dos servicios escriben el mismo almacén
un servicio hace una llamada por elemento a otro   clase 124
la frontera exige transacción distribuida          clase 149
```

Y el último es el más informativo:

```
si mantener la corrección exige una transacción a través de
la frontera, la frontera está en el sitio equivocado
→ mueve la frontera, no añadas coordinación distribuida
```

Cuándo mover una frontera y cuánto cuesta:

```
MOVER HACIA DENTRO (juntar dos módulos)   barato
  suele ser mecánico; se pierde autonomía

MOVER HACIA FUERA (separar uno en dos)    caro si hay datos
  si comparten almacén, hay que dividir datos, y eso es
  lo más caro que existe                              clase 147

REGLA PRÁCTICA
  ante la duda, EMPIEZA JUNTO y separa cuando el ritmo de
  cambio o el escritor lo exijan
  → juntar lo separado es más barato que separar lo junto
```

Y la lista de comprobación de la clase:

- ■ está identificado el tipo de acoplamiento de cada relación
- ■ ningún almacén tiene más de un escritor

- se ha medido el porcentaje de cambios acoplados
- se ha medido el ritmo de cambio de cada parte
- ninguna frontera exige transacción distribuida
- ninguna frontera obliga a desplegar dos cosas juntas
- las fronteras no se copiaron del organigrama sin pensar
- se sabe el coste de mover cada frontera
- ante la duda, se empezó junto

Y el cierre que enlaza con la clase siguiente: sabiendo dónde van las fronteras, queda decidir cuántas unidades desplegables las materializan —una, unas pocas o muchas— y esa es una decisión distinta y con su propio coste. Es la materia de la clase 184.

4. Modularidad sin repartir en servicios

Una confusión frecuente y cara: frontera no significa servicio. Se puede tener modularidad estricta dentro de un solo despliegue, y suele ser la mejor opción durante años.

```
LO QUE DA LA FRONTERA MODULAR (mismo despliegue)
  claridad de propiedad
  un escritor por dato
  contratos internos explícitos
  posibilidad de separar después

LO QUE AÑADE SEPARAR EN SERVICIOS
  despliegue independiente
  escalado independiente
  aislamiento de fallo
  y a cambio: red, latencia, consistencia eventual,
  operación, y coordinación                          clase 184
```

Y el modo de mantener fronteras dentro de un mismo despliegue, que es lo que las hace reales:

```
cada módulo con su propio esquema o su propio conjunto de tablas
prohibida la consulta cruzada; se pide por la interfaz del módulo
el acceso a datos de otro módulo NO compila
  → paquetes internos, reglas de dependencia, análisis estático
cada módulo tiene dueño escrito
y sus cambios se revisan por ese dueño
```

Y la comprobación que dice si la frontera modular es real o decorativa:

```
¿se podría sacar este módulo a su propio despliegue en una
semana, sin tocar los demás?

  sí   la frontera es real
  no   es una carpeta con un nombre bonito
```

Y una advertencia con evidencia de este programa:

```
las fronteras que solo existen por convención se erosionan
→ alguien hace una consulta cruzada «solo esta vez»
→ y a los dos años hay 7                             clase 179
→ si no lo impide el compilador o la revisión, ocurrirá
```

Ejemplo trabajado

El equipo de reservas decide dónde van sus fronteras. Lo que sigue es el análisis de acoplamiento con datos del repositorio, el rechazo de la descomposición que salía sola, y las cuatro fronteras que resultaron.

Punto de partida: la descomposición funcional obvia.

```
Reservas · Pagos · Catálogo · Precios · Clientes · Notificaciones
```

Y parecía correcta hasta que se miraron dos datos que están en el repositorio y no requieren opinión.

Dato 1: cambios acoplados, últimos 12 meses.

```
pares que cambian juntos                    % de cambios
  Reservas × Pagos                                71 %
  Catálogo × Precios                              64 %
  Reservas × Notificaciones                        6 %
  Catálogo × Clientes                              3 %
  Precios × Reservas                              38 %
```

Y la lectura:

```
Reservas y Pagos cambian juntos el 71 % de las veces
→ separarlos crea dos repositorios que hay que tocar a la vez
→ NO son dos módulos: son uno

Catálogo y Precios, 64 %
→ lo mismo

Precios × Reservas, 38 %
→ zona dudosa; hay que mirar por qué
```

Dato 2: ritmo de cambio y origen de los cambios.

```
parte                cambios/mes   los pide
reglas de precios          9       producto y revenue
disponibilidad             1       operaciones
flujo de reserva           2       producto
coordinación de pago       1       finanzas
plantillas de aviso        6       marketing
datos de contacto          4       marketing
consentimientos            1       legal
```

Y aquí apareció la contradicción con el dato 1:

```
Precios cambia 9 veces al mes; Catálogo, 1
→ el 64 % de cambios acoplados NO significa que pertenezcan
  juntos: significa que cada cambio de precio obligaba a tocar
  el catálogo
→ y al mirar por qué: los precios estaban GUARDADOS en la tabla
  de catálogo

el acoplamiento era del tipo 7, por datos, disfrazado
```

Dato 3: quién escribe cada almacén, del diagrama de la clase 182.

```
tabla                    escritores
reservas                 API, trabajos programados, facturación
catálogo                 catálogo, precios
clientes                 API, importación de marketing
inventario               gestor de hoteles, trabajos programados
```

Y cuatro almacenes con más de un escritor, que es donde estaba el problema real.

Las decisiones, con el criterio de la clase.

```
DECISIÓN 1   Reservas y Pagos van JUNTOS
  motivo     71 % de cambios acoplados; y la corrección exige
             transacción entre ambos
  alternativa descartada   separarlos con saga
  por qué    la compensación habría hecho invisible el fallo
             cuando el pago se confirma y la reserva no  ley 19
  coste de cambio si nos equivocamos   medio

DECISIÓN 2   Precios SALE de Catálogo
  motivo     9 cambios/mes frente a 1; equipos distintos
  cómo       catálogo deja de guardar el precio; precios tiene
             su propio almacén y publica un evento de cambio
  convierte  acoplamiento 7 → 1
  coste      alto: hay que migrar el dato       ← se hace pronto

DECISIÓN 3   Clientes NO se parte por sustantivo
  motivo     contacto (4/mes, marketing), consentimientos
             (1/mes, legal) y preferencias cambian por motivos
             distintos
  cómo       tres módulos con esquemas separados DENTRO del
             mismo despliegue, cada uno con dueño
  por qué no tres servicios   el tráfico no lo justifica y la
             operación costaría más que la autonomía que da
  coste de separarlos después   bajo: ya tienen esquema propio

DECISIÓN 4   Notificaciones queda separada
  motivo     6 % de cambios acoplados, y ya era asíncrona
  confirma   la frontera existente estaba bien puesta
```

Los escritores, después:

```
tabla                    escritor único
```

```
reservas                  servicio de reservas y pagos
  facturación             → recibe evento, ya no escribe
  trabajos programados    → 4 retirados; 3 pasan por la API
catálogo                  servicio de catálogo
precios                   servicio de precios      ← nuevo
clientes.contacto         módulo de contacto
clientes.consentimientos  módulo de consentimientos
inventario                servicio de catálogo
  gestor de hoteles       → escribe por la API, no directo
```

Cómo se hicieron reales las fronteras dentro del despliegue de clientes, que es lo que evita que se erosionen:

```
tres esquemas separados en la misma base
nadie consulta el esquema de otro módulo
  → usuario de base de datos por módulo, con permisos solo
    sobre su esquema; la consulta cruzada FALLA, no se detecta
    en revisión
reglas de dependencia en el análisis estático
cada módulo con dueño en el fichero de propiedad
```

Y la comprobación de la clase, aplicada:

```
¿se podría sacar contacto a su propio despliegue en una semana?
  → sí: esquema propio, sin consultas cruzadas, contrato claro
  → la frontera es real
```

Lo que costó equivocarse una vez. La primera propuesta separaba Reservas y Pagos:

```
se llegó a implementar la saga durante 3 semanas
se abandonó al hacer la prueba negativa: al fallar el paso de
confirmación, la compensación cancelaba el pago y el cliente
recibía dos correos contradictorios
coste del error                          3 semanas
cómo se detectó                          prueba negativa
qué lo habría evitado                    mirar el 71 % de
                                         cambios acoplados
                                         ANTES de diseñar
```

La lección que esta clase deja: dos datos que ya estaban en el repositorio —qué cambia junto y a qué ritmo— corrigieron una descomposición funcional que parecía obvia, y el acoplamiento más caro del sistema no estaba en ninguna llamada: estaba en que el precio vivía en la tabla del catálogo. Ningún diagrama de servicios lo habría mostrado.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-15-systems-architecture-engineering/183-acoplamiento-cohesion-modularidad-y-
fronteras/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 module-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 module-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
Dos servicios «independientes» no pueden cambiar de esquema sin coordinarse	Acoplamiento por datos: ambos escriben el mismo almacén	Un escritor por dato; los demás reciben eventos. Convierte el acoplamiento tipo 7 en tipo 1.
Un cambio pequeño obliga a tocar tres repositorios	La frontera cruza justo por donde ocurren los cambios	Calcula el porcentaje de cambios acoplados del historial; por encima del 60 % esas partes van juntas.
Un módulo se despliega constantemente por culpa de una parte que cambia mucho	Se agrupó por sustantivo y no por motivo ni ritmo de cambio	Separa lo que cambia 8 veces al mes de lo que cambia una vez al trimestre, aunque compartan nombre.
Mantener la corrección exige una transacción distribuida	La frontera está en el sitio equivocado	Mueve la frontera en vez de añadir coordinación distribuida y compensaciones.
Las fronteras modulares se erosionan y aparecen consultas cruzadas	Solo existían por convención	Haz que la consulta cruzada falle: esquemas y usuarios de base separados, reglas de dependencia en el análisis estático.
Separar un módulo resulta muchísimo más caro de lo previsto	Compartía almacén y hay que dividir datos	Ante la duda empieza junto: juntar lo separado es más barato que separar lo junto.

■ 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 de los siete tipos de acoplamiento es el más caro y por qué es invisible?
2. ¿Qué mide el porcentaje de cambios acoplados y cómo se obtiene?
3. ¿Por qué agrupar por sustantivo de negocio suele dar mala cohesión?
4. ¿Qué indica que una frontera está mal puesta si exige transacción distribuida?
5. ¿Qué comprobación dice si una frontera modular es real o decorativa?

Referencias

- Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. <https://dl.acm.org/doi/10.1145/361598.361623>

- Newman, S. (2019). Monolith to Microservices — acoplamiento por datos y cómo dividirlo. <https://www.oreilly.com/library/view/monolith-to-microservices/9781492047834/>
- Tornhill, A. (2018). Software Design X-Rays — cambios acoplados medidos sobre el historial. <https://pragprog.com/titles/atevol/software-design-x-rays/>
- Evans, E. (2003). Domain-Driven Design — contextos delimitados y sus fronteras. <https://www.oreilly.com/library/view/domain-driven-design-tackling/0321125215/>
- Conway, M. (1968). How do committees invent? — la frontera y el organigrama. <https://www.melconway.com/Home/CommitteesPaper.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 15 en PDF · Manual integral

Anterior	Índice	Siguiente
← 182 · Contexto, contenedores, componentes y código con C4	Parte 15 · Programa	184 · Arquitectura monolítica, modular y de microservicios →

Evaluación — 183 Acoplamiento, cohesión, modularidad y fronteras

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 module-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.

184 — Arquitectura monolítica, modular y de microservicios

Parte: 15 — Arquitectura de sistemas e ingeniería de requisitos Nivel: intermedio-avanzado · Horas estimadas: 4
Laboratorio: decision · Estado: EXECUTABLECORE

Propósito

Elegir cuántas unidades desplegables materializan las fronteras de la clase 183: una, unas pocas o muchas. La clase trata el monolito modular como opción por defecto legítima y no como fracaso, enumera los cinco motivos que de verdad justifican separar en servicios, cuantifica lo que cuesta cada separación, y da el orden de migración que evita los dos desastres habituales: el monolito distribuido y la reescritura completa.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir entre monolito modular, unos pocos servicios y muchos con criterios comprobables.
2. Aplicar los cinco motivos válidos para separar, y descartar los inválidos.
3. Cuantificar el coste operativo de cada servicio adicional.
4. Reconocer un monolito distribuido antes de haberlo construido.
5. Migrar por partes, empezando por la que más lo justifica.

Conceptos centrales

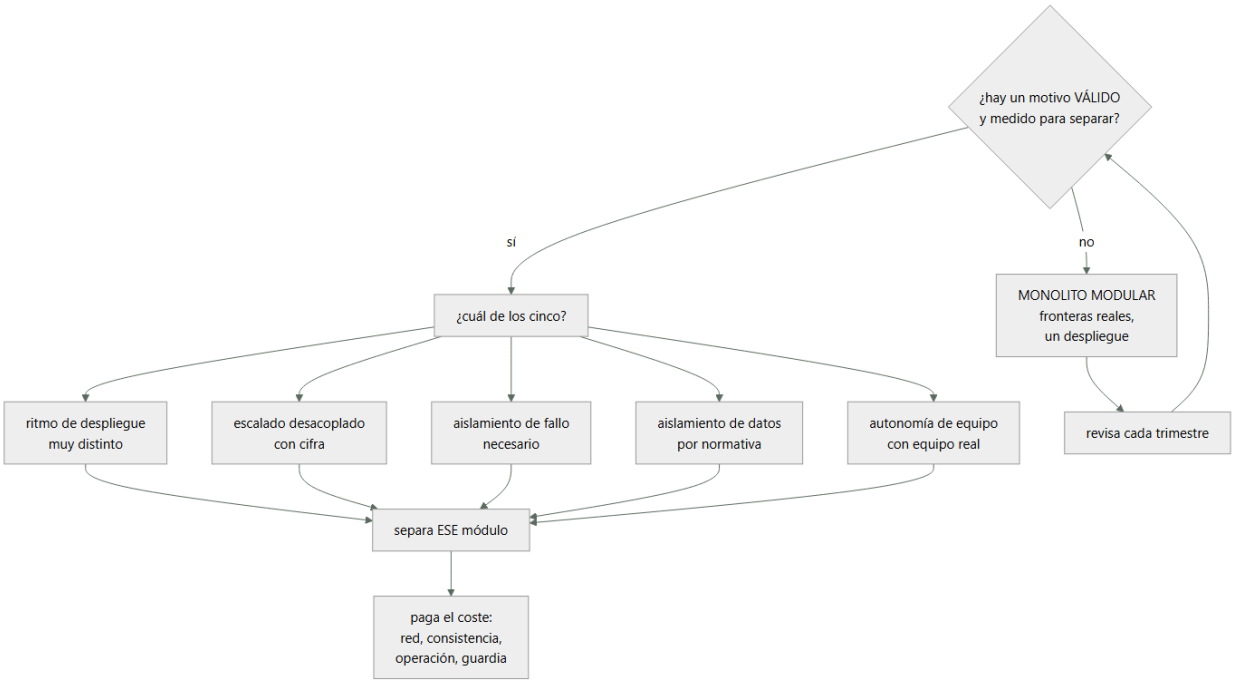
Concepto	Comprensión verificable
monolito modular	Un despliegue con fronteras internas reales: esquemas separados, un escritor por dato, dependencias controladas.
monolito distribuido	Muchos servicios que hay que desplegar juntos y comparten datos. Suma los costes de ambos modelos y no da las ventajas de ninguno.
motivo válido de separación	Razón medible: ritmo de despliegue, escalado desacoplado, aislamiento de fallo, aislamiento de datos o autonomía de equipo.
coste marginal por servicio	Lo que cuesta al mes tener un servicio más: canalización, telemetría, guardia, cuota, seguridad.
estrangulamiento	Migración por partes en la que lo nuevo intercepta tráfico y lo viejo se retira poco a poco.
latencia de frontera	Lo que añade cada salto de red: milisegundos, fallos parciales y consistencia eventual.

Modelo mental

Una arquitectura es un conjunto de decisiones sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

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 monolito modular como opción por defecto

La discusión suele plantearse como «monolito o microservicios», y el planteamiento ya es el error: la frontera y el despliegue son decisiones distintas.

```
FRONTERA          dónde está la línea           clase 183
DESPLIEGUE        cuántas unidades hay          esta clase

→ se puede tener 12 fronteras y 1 despliegue
→ y se puede tener 12 despliegues y 0 fronteras reales
   (que es el peor caso de todos)
```

Y el monolito modular no es un paso intermedio ni una derrota:

```
QUÉ DA
  fronteras reales, propiedad clara, un escritor por dato
  transacciones locales, sin coordinación distribuida
  una canalización, una telemetría, una guardia
  latencia de llamada en nanosegundos
  refactorizar fronteras es barato

QUÉ NO DA
  despliegue independiente
  escalado por parte
  aislamiento de fallo
  autonomía de equipos grandes
```

Y el dato que más pesa en la decisión, con el tamaño de equipo real:

```
hasta ~8 personas   el coste de coordinación humana es bajo
                    → un despliegue casi siempre gana
8 a 25              zona de decisión; separa lo que lo pida
> 25                la coordinación de un solo despliegue empieza
                    a dominar; separar por equipo tiene sentido
```

Y una advertencia sobre la comparación habitual:

```
se compara «monolito heredado sin fronteras» contra
«microservicios bien hechos»
→ la comparación honesta es monolito MODULAR contra servicios,
  y entonces la ventaja se reduce mucho
```

2. Los cinco motivos válidos, y los inválidos

Separar un módulo en su propio servicio tiene que justificarse con uno de estos cinco, con una cifra:

```
1. RITMO DE DESPLIEGUE MUY DISTINTO
   este módulo cambia 9 veces al mes y el resto 1
   → y los despliegues del resto lo bloquean, o al revés
   cifra   despliegues/mes de cada parte            clase 183

2. ESCALADO DESACOPLADO
   este módulo consume 8× los recursos del resto, o su carga
   sube 20× en campaña mientras el resto no se mueve
   cifra   uso de CPU/memoria por módulo y su perfil temporal
   ojo     si el ahorro es de 200 €/mes, no compensa

3. AISLAMIENTO DE FALLO
   un fallo aquí no debe poder tirar lo demás
   cifra   incidentes originados en este módulo que afectaron
           al resto
   ojo     dentro de un proceso también se puede aislar
           (límites, mamparos, tiempos de espera)   clase 153

4. AISLAMIENTO DE DATOS POR NORMATIVA
   estos datos deben vivir en otra región, otra cuenta u otro
   régimen de acceso                                clase 177
   → es el motivo menos discutible de los cinco

5. AUTONOMÍA DE EQUIPO
   hay un equipo REAL, con personas, que lo posee y lo opera
   ojo     «vamos a contratar» no cuenta; un servicio sin
           equipo es un servicio sin dueño            ley 20
```

Y los motivos que se usan mucho y no aguantan:

```
«ES LA BUENA PRÁCTICA»                          ley 17
«ASÍ ESCALAMOS EN EL FUTURO»
  → el futuro rara vez llega con la forma prevista, y mientras
    tanto se paga todos los meses
«CADA SERVICIO EN SU LENGUAJE»
  → multiplica la operación por el número de lenguajes
«EL MONOLITO ESTÁ HECHO UN DESASTRE»
  → el desastre es la falta de fronteras; separarlo sin
    arreglarlas produce un monolito distribuido
«PARA QUE LOS EQUIPOS NO SE PISEN»
  → si el problema es el fichero compartido, la frontera
    modular ya lo resuelve                          clase 183
```

Y el criterio que resume todo:

```
separa UN módulo cuando UNO de los cinco lo pide, con cifra
no separes «el sistema» de golpe
```

3. Lo que cuesta cada servicio

El coste de un servicio adicional se subestima siempre porque es difuso: no aparece en ninguna factura con su nombre.

```
POR CADA SERVICIO, AL MES
  canalización y su mantenimiento
  telemetría: paneles, alertas, objetivos          clase 125
  procedimientos y turno de guardia                clase 127
  actualizaciones de base y de dependencias
  identidad, permisos, secretos                    clase 134
  presupuesto y etiquetas                          clase 142
  cuota y límites
  documentación y contrato                         clase 188

y los costes que no se ven hasta el incidente
  un salto de red más en el camino crítico
  un modo de fallo parcial más
  consistencia eventual donde antes había transacción
  una correlación más que mantener                 clase 121
```

Y el orden de magnitud, medido en organizaciones reales:

```
un servicio bien operado cuesta entre 2 y 6 días-persona al
mes de mantenimiento, sin contar el desarrollo
→ 20 servicios ≈ una persona entera solo en mantenerlos
→ y ese es el mecanismo de la ley 23
```

La latencia de frontera, que se paga en cada llamada:

```
llamada dentro del proceso            ~100 ns
llamada de red en la misma zona       0,5 - 2 ms
llamada entre zonas                   1 - 4 ms
con serialización y TLS               +0,3 - 1 ms

→ una operación que antes hacía 6 llamadas internas y ahora
  cruza 4 fronteras pasa de microsegundos a ~8 ms
→ y si alguna es una llamada por elemento, a segundos  clase 124
```

Y el efecto sobre la disponibilidad, que es el más olvidado:

```
cada dependencia síncrona dura multiplica               clase 185
  6 servicios al 99,9 % en serie → 99,4 %
→ separar sin convertir las llamadas en asíncronas BAJA la
  disponibilidad
```

El monolito distribuido, que es el resultado de separar mal, se reconoce por estas señales:

```
hay que desplegar varios servicios a la vez, en orden
varios servicios comparten base de datos
un cambio de esquema requiere coordinar equipos
las pruebas necesitan todo el sistema levantado
cada operación de usuario cruza más de 4 servicios
hay una transacción distribuida o una saga por operación normal
nadie puede desplegar en viernes
```

Y el diagnóstico:

```
si tres o más de esas señales están presentes, se pagan los
costes de los servicios y los del monolito a la vez
→ y la salida suele ser JUNTAR, no separar más
```

4. Cómo migrar sin reescribir

Cuando la separación está justificada, el modo importa tanto como la decisión.

Lo que no funciona:

```
LA REESCRITURA COMPLETA
  se congela el sistema viejo, se construye el nuevo
  → el viejo sigue cambiando porque el negocio no se para
  → el nuevo persigue un objetivo móvil durante 18 meses
  → y la mitad de estos proyectos se abandonan
```

Lo que funciona: estrangulamiento.

```
1. poner una fachada delante del sistema actual
   → todo el tráfico pasa por ahí; nada cambia todavía

2. elegir UN módulo, el que más lo justifique
   → normalmente el de ritmo de cambio más alto

3. hacer que el dato tenga un solo escritor
   → esto es la parte cara, y se hace ANTES de mover código

4. desviar el tráfico de ese módulo al servicio nuevo
   → con despliegue escalonado y vuelta atrás    clase 102

5. RETIRAR el código y los datos viejos
   → sin este paso, la migración solo añade      clase 171

6. medir, y decidir si el siguiente módulo lo merece
```

Y el paso 3 es el que decide el resultado:

```
mover el código sin dividir los datos produce dos servicios
que escriben la misma tabla
→ acoplamiento tipo 7, el más caro                 clase 183
→ y es exactamente el monolito distribuido
```

Y el paso 5 es el que más se salta:

```
migración sin retirada = dos sistemas que mantener
→ y la capacidad del equipo se consume manteniendo      ley 23
→ la migración no termina cuando lo nuevo funciona:
  termina cuando lo viejo ya no existe
```

Y la lista de comprobación de la clase:

- las fronteras están decididas antes que el despliegue
- cada separación tiene uno de los cinco motivos, con cifra
- hay un equipo real para cada servicio propuesto
- está calculado el coste marginal mensual por servicio
- está calculado el efecto en latencia y en disponibilidad
- ninguna operación normal exige transacción distribuida
- ningún par de servicios comparte almacén
- no hay servicios que deban desplegarse en orden
- la migración es por partes, no reescritura
- los datos se dividen antes de mover el código
- cada paso termina con la retirada de lo viejo

Y el cierre que enlaza con la clase siguiente: cada frontera que se materializa en un servicio añade un punto que puede fallar y una dependencia que se hereda. Calcular a cuánto asciende eso —y dónde está el techo real de disponibilidad— es la materia de la clase 185.

Ejemplo trabajado

El equipo de reservas decide cuántos despliegues materializan las cuatro fronteras de la clase 183. Lo que sigue es la evaluación de los cinco motivos módulo por módulo, el cálculo del coste, y la migración de los dos que se separaron.

Punto de partida: 6 personas, un monolito sin fronteras internas, 214 despliegues al año.

Evaluación módulo por módulo, con cifras.

```
PRECIOS
  1 ritmo         9 despliegues/mes frente a 2 del resto     ✓
                  y 4 veces el mes pasado un despliegue de
                  precios se retrasó por esperar a reservas
  2 escalado      no; consume el 6 % de la CPU               ✗
  3 fallo         un error de reglas tiró el sistema entero
                  2 veces en el año                          ✓
  4 datos         no                                         ✗
  5 equipo        lo lleva 1 persona con revenue             ~
  DECISIÓN        SEPARAR — motivos 1 y 3, ambos con cifra

BÚSQUEDA
  1 ritmo         1 despliegue/mes                           ✗
  2 escalado      3.000/s en pico frente a 40/s del resto;
                  perfil temporal completamente distinto;
                  ahorro estimado 3.100 €/mes                ✓
  3 fallo         una búsqueda lenta agotaba el grupo de
                  conexiones y afectaba a reservas           ✓
  4 datos         no                                         ✗
  5 equipo        no hay equipo propio                       ✗
  DECISIÓN        SEPARAR — motivo 2 con cifra clara

RESERVAS + PAGOS
  1 ritmo         2/mes                                      ✗
  2 escalado      es el resto                                 ✗
  3 fallo         es el núcleo; aislarlo de qué                 ✗
  4 datos         los datos de tarjeta no se guardan, se
                  tokenizan en la pasarela                    ✗
  5 equipo        4 personas, el mismo equipo                 ✗
  DECISIÓN        NO SEPARAR — y además el 71 % de cambios
                  acoplados lo desaconseja              clase 183

CATÁLOGO
  1 ritmo         1/mes                                      ✗
  2 escalado      no                                         ✗
  3 fallo         no ha causado incidentes                   ✗
  4 datos         no                                         ✗
  5 equipo        no                                         ✗
  DECISIÓN        NO SEPARAR — queda como módulo
```

```
CLIENTES (contacto, preferencias, consentimientos)
  4 datos         los consentimientos deben ser auditables y
                  con acceso restringido a legal              ~
  DECISIÓN        NO SEPARAR todavía — el aislamiento se
                  consigue con esquema y permisos propios,
                  sin un despliegue más

NOTIFICACIONES
  ya estaba separada; 6 % de cambios acoplados; se mantiene
```

Resultado: 4 despliegues, no 12.

```
monolito modular   reservas+pagos, catálogo, clientes (3 módulos)
precios            servicio
búsqueda           servicio
notificaciones     servicio
```

El coste, calculado antes de decidir.

```
coste marginal por servicio, estimado con el histórico
  canalización y mantenimiento         0,8 día-persona/mes
  telemetría y alertas                 0,5
  actualizaciones y dependencias       0,7
  identidad, secretos, permisos        0,3
  documentación y contrato             0,4
  TOTAL                                2,7 días-persona/mes

3 servicios además del monolito       8,1 días-persona/mes
equipo de 6 personas ≈ 120 días/mes   → 6,8 % de la capacidad

contraste: si se hubieran separado los 12 módulos
  12 × 2,7 = 32,4 días/mes            → 27 % de la capacidad
  y el equipo no habría podido construir nada         ley 23
```

Efecto en latencia y disponibilidad, calculado antes:

```
el flujo de reserva cruza ahora 1 frontera nueva (precios)
  +1,4 ms de p50, +6 ms de p99
  aceptable frente al QA-1 de 500 ms                clase 181

disponibilidad
  precios era dependencia DURA en el flujo de reserva
  → 99,9 × 99,9 = 99,8 %, por debajo del QA-2 de 99,7 %  ✓ justo
  → decisión: el precio se CACHEA con validez de 10 min y
    si precios no responde se usa el último válido
  → pasa a dependencia BLANDA, y el techo vuelve a 99,9 %
```

Y esa decisión —convertir la dependencia dura en blanda— fue la que hizo viable la separación, y no se habría visto sin el cálculo.

La migración de precios, por estrangulamiento.

```
semana 1   fachada delante del monolito; nada cambia
semanas 2-5  DIVIDIR EL DATO   ← la parte cara
  el precio vivía en la tabla de catálogo
  se crea el almacén de precios y se escribe en los dos
  se compara durante 2 semanas: 3 discrepancias, todas por
    un trabajo programado olvidado que escribía precios
  catálogo deja de escribir el precio
semana 6   el servicio de precios lee de su almacén
semana 7   la fachada desvía el 5 %, luego 25 %, luego 100 %
semana 8   se publica el evento de cambio de precio
semanas 9-10  RETIRADA
  columna de precio eliminada del catálogo
  código viejo borrado
  trabajo programado retirado
```

Y lo que costó y lo que se descubrió:

```
dividir el dato                   4 semanas de las 10
mover el código                   1 semana
hallazgo   un trabajo programado escribía precios desde 2021
           y nadie lo sabía; explicaba 3 incidentes de
           «precio incorrecto» sin causa conocida
```

El resultado, seis meses después.

```
despliegues de precios bloqueados/mes            4         0
incidentes originados en reglas de precios       2/año     0
coste de cómputo en pico                    -3.100 €/mes
p99 del flujo de reserva                    412 ms    438 ms
disponibilidad del flujo                    99,71 %   99,74 %
capacidad consumida en mantenimiento          n/d       6,8 %
servicios que comparten almacén                 4         0
```

Y la decisión que no se tomó, registrada para no rediscutirla cada trimestre:

```
no separar reservas y pagos
revisar si   el equipo pasa de 12 personas
             o el 71 % de cambios acoplados baja del 30 %
```

La lección que esta clase deja: de doce módulos, solo tres cumplían alguno de los cinco motivos con una cifra detrás, y separar los doce habría consumido el 27 % de la capacidad del equipo en mantenimiento. Y la parte cara de separar un servicio no fue mover el código —una semana— sino dividir el dato: cuatro. Quien separa código sin dividir datos no ha separado nada.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-15-systems-architecture-engineering/184-arquitectura-monolitica-modular-y-de-microservicios/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 architecture-style-adr 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 architecture-style-adr 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
Hay muchos servicios pero hay que desplegarlos en orden y comparten base	Monolito distribuido: se separó el código sin dividir los datos	Divide primero los datos con un escritor único; si ya está hecho al revés, plantea juntar en vez de separar más.
El equipo apenas construye nada nuevo	El coste de mantener los servicios consume la capacidad	Calcula el coste marginal por servicio y retira; separa solo lo que cumpla uno de los cinco motivos con cifra.
La disponibilidad bajó tras separar en servicios	Las llamadas nuevas son síncronas y duras, y se multiplican	Calcula el techo antes de separar y convierte las dependencias del camino crítico en blandas con caché o valor por defecto.
La reescritura completa lleva año y medio y no sale	El sistema viejo sigue cambiando mientras se construye el nuevo	Migra por estrangulamiento: fachada, un módulo cada vez, y retirada al final de cada paso.
Tras la migración hay que mantener el sistema nuevo y el viejo	No hubo paso de retirada	La migración termina cuando lo viejo deja de existir, no cuando lo nuevo funciona.
Se separa un servicio y acaba sin quien lo atienda	Se contó con un equipo que no existía	Exige equipo real y dueño escrito antes de crear un despliegue nuevo.

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 frontera y despliegue son decisiones distintas?
2. ¿Cuáles son los cinco motivos válidos para separar y qué cifra respalda cada uno?
3. ¿Cuánto cuesta al mes un servicio adicional y por qué no aparece en ninguna factura?
4. ¿Por qué separar sin convertir llamadas en asíncronas baja la disponibilidad?
5. ¿Cuál es el paso caro del estrangulamiento y cuál el que más se salta?

Referencias

- Newman, S. (2021). Building Microservices, 2.ª ed. — cuándo separar y qué cuesta. <https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/>
- Fowler, M. (2015). MonolithFirst — empezar junto y separar con evidencia. <https://martinfowler.com/bliki/MonolithFirst.html>
- Fowler, M. (2004). StranglerFigApplication — migración por estrangulamiento. <https://martinfowler.com/bliki/StranglerFigApplication.html>
- Shopify Engineering (2019). Deconstructing the monolith — monolito modular a escala. <https://shopify.engineering/deconstructing-monolith-designing-software-maximizes-developer-productivity>
- Skelton, M. y Pais, M. (2019). Team Topologies — servicios, equipos y carga cognitiva. <https://teamtopologies.com/book>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 15 en PDF · Manual integral

Evaluación — 184 Arquitectura monolítica, modular y de microservicios

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-style-adr 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.

185 — Disponibilidad, confiabilidad y análisis de puntos de fallo

Parte: 15 — Arquitectura de sistemas e ingeniería de requisitos Nivel: intermedio-avanzado · Horas estimadas: 4
Laboratorio: reliability · Estado: EXECUTABLECORE

Propósito

Calcular a cuánto asciende la disponibilidad que se puede prometer, y descubrir por qué casi siempre es menor de lo que se cree. La clase da la aritmética de dependencias en serie y en paralelo, el análisis de puntos de fallo hecho como se hace de verdad —no listando componentes sino preguntando qué pasa si cada uno desaparece—, y sostiene con la evidencia del programa que la mayoría de los puntos únicos de fallo no son de infraestructura, sino de conocimiento y de procedimiento.

Resultados de aprendizaje

Al finalizar podrás:

1. Calcular el techo de disponibilidad a partir de las dependencias.
2. Distinguir dependencia dura de blanda y convertir unas en otras.
3. Ejecutar un análisis de puntos de fallo que encuentre los que no son técnicos.
4. Cuantificar qué aporta cada redundancia y cuánto cuesta.
5. Decidir dónde no invertir en disponibilidad, con cifras.

Conceptos centrales

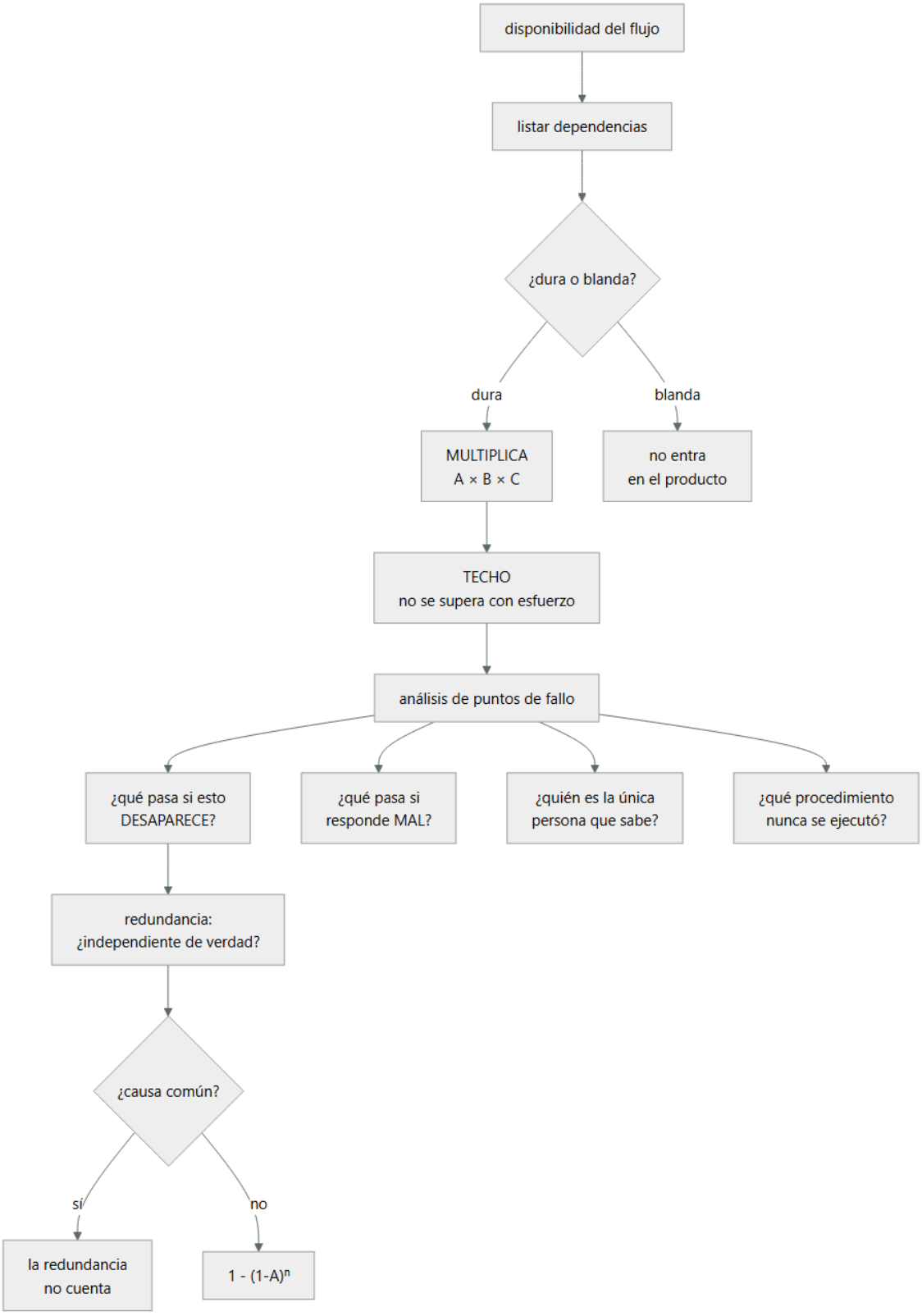
Concepto	Comprensión verificable
dependencia dura	Si falla, la operación falla. Su disponibilidad se multiplica con la del sistema.
dependencia blanda	Si falla, la operación continúa degradada. No entra en el producto.
techo de disponibilidad	Máximo que se puede prometer dadas las dependencias duras. No se supera con esfuerzo.
punto único de fallo	Elemento cuya desaparición interrumpe el servicio. Puede ser una máquina, un dato, una persona o un procedimiento.
fallo correlacionado	Dos réplicas que caen a la vez por una causa común. Anula la redundancia calculada.
modo de fallo gris	El componente no cae: responde mal o lento. La redundancia clásica no lo cubre.

Modelo mental

Una arquitectura es un conjunto de decisiones sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

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 aritmética que casi nadie hace

La disponibilidad prometida se calcula, no se declara. Y el cálculo es elemental, lo que hace más llamativo que se omita.

En serie, que es el caso normal de una llamada síncrona dura:

```
A depende de B, B de C, todas duras
  disponibilidad = A × B × C

ejemplo real de un flujo de reserva
  balanceador          99,99 %
  API de reservas      99,95 %
  base de datos        99,99 %
  servicio de precios  99,90 %
  pasarela de pago     99,95 %
  identidad            99,99 %
  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  techo                99,77 %      ← ~100 min/mes de parada

y la promesa que había en el contrato   99,9 %
→ imposible de cumplir por aritmética, no por calidad
```

Y el efecto acumulativo es lo que sorprende:

```
 5 dependencias al 99,9 %  →  99,50 %
10 dependencias al 99,9 %  →  99,00 %
20 dependencias al 99,9 %  →  98,02 %

→ y por eso separar en servicios sin convertir llamadas en
  asíncronas baja la disponibilidad                clase 184
```

En paralelo, que es lo que aporta la redundancia:

```
n réplicas independientes, cualquiera sirve
  disponibilidad = 1 - (1 - A)■

2 réplicas al 99 %    →  99,99 %
2 réplicas al 99,9 %  →  99,9999 %

→ la redundancia da mucho MÁS de lo que quita la serie
→ siempre que las réplicas sean INDEPENDIENTES
```

Y aquí está el engaño más frecuente de todo el cálculo:

```
FALLO CORRELACIONADO
  dos réplicas en la misma zona          ← comparten energía
  dos réplicas con el mismo despliegue   ← comparten el error
  dos réplicas con el mismo certificado  ← caduca a la vez
  dos réplicas con la misma configuración errónea
  dos réplicas que dependen del mismo servicio de identidad

→ en todos esos casos la fórmula del paralelo NO aplica
→ y la disponibilidad real es la de UNA réplica
```

Y la comprobación práctica:

```
por cada redundancia, preguntar
  ¿qué causa haría caer a las dos a la vez?
  → si existe alguna, la redundancia solo cubre el resto
```

2. Duras, blandas y cómo convertir

La palanca más barata para subir el techo no es añadir réplicas: es reducir el número de dependencias duras.

```
DURA     si falla, la operación falla
BLANDA   si falla, la operación continúa degradada

y la mayoría de las dependencias que se creen duras no lo son
```

Las cuatro formas de convertir dura en blanda:

```
1. VALOR POR DEFECTO
   si recomendaciones no responde, no hay recomendaciones
   coste   ninguno; solo decidirlo

2. ÚLTIMO VALOR VÁLIDO
   si precios no responde, se usa el precio cacheado con
   validez declarada
   coste   posible precio desactualizado unos minutos
   ojo     hay que decidir CUÁNTO tiempo es aceptable

3. DIFERIR
   si el correo no sale, se encola y sale luego     clase 118
```

```
   coste   complejidad de cola y reintentos

4. ACEPTAR Y RECONCILIAR
   se acepta la operación y se verifica después
   coste   hay que manejar el caso de rechazo posterior
   ojo     la compensación hace invisible el fallo    ley 19
```

Y las que no se pueden convertir, y hay que aceptar como techo:

```
el almacén donde vive el dato que se escribe
la identidad, si cada petición la necesita
el cobro, si el negocio no acepta reservar sin cobrar
→ y sobre estas se decide con redundancia o con contrato
```

Y el ejemplo del efecto, con el mismo flujo de antes:

```
antes    6 dependencias duras         techo 99,77 %

conversiones
  precios      → último valor válido, 10 min
  identidad    → validación local del testigo, sin llamada

después  4 dependencias duras         techo 99,88 %
coste    dos decisiones y un caché; cero euros de infraestructura
```

Y una advertencia que este programa ha demostrado dos veces:

```
una dependencia declarada blanda que nunca se ha probado
suele ser dura                                    ley 22
→ la prueba: apagarla y ver si la operación sigue
→ en la clase 179, el catálogo estaba declarado blando
  y no lo era
```

3. Análisis de puntos de fallo, hecho de verdad

El análisis que se hace habitualmente lista componentes y marca los que tienen redundancia. Encuentra poco porque hace la pregunta fácil.

Las cuatro preguntas que sí encuentran:

```
1. ¿QUÉ PASA SI ESTO DESAPARECE?
   la pregunta clásica; encuentra los puntos únicos técnicos

2. ¿QUÉ PASA SI RESPONDE MAL O LENTO?
   el fallo gris; la redundancia no lo cubre porque el
   elemento sigue «vivo»                          clase 130
   → una réplica lenta puede ser peor que una caída

3. ¿QUIÉN ES LA ÚNICA PERSONA QUE SABE ESTO?
   punto único de conocimiento
   → no aparece en ningún diagrama                clase 180

4. ¿QUÉ PROCEDIMIENTO NUNCA SE HA EJECUTADO?
   punto único de procedimiento                    ley 22
   → el que se descubre cuando hace falta
```

Y la evidencia de este programa sobre el reparto:

```
en la clase 179, de los puntos únicos encontrados
  de infraestructura                               2
  de datos (un escritor sin réplica)               1
  de conocimiento (una sola persona sabía)         1
  de procedimiento (nunca ejecutado)               5

→ la mayoría NO eran de infraestructura
```

Los puntos únicos que se olvidan siempre, por categoría:

```
TÉCNICOS NO EVIDENTES
  el registro de artefactos: si cae, no hay despliegues
  el proveedor de identidad: si cae, no entra nadie
  el DNS y los certificados: caducan a la vez en todas partes
  el servicio de secretos
  la cuenta de un proveedor externo con un solo pagador

DE DATOS
```

```
  la única copia, aunque haya réplicas             clase 166
  el único escritor sin plan de sustitución

HUMANOS Y DE PROCESO
  la persona que conoce el despliegue manual
  la aprobación que solo puede dar una persona
  el procedimiento de recuperación nunca ensayado
  el acceso de emergencia nunca usado              clase 179
  el contrato con un proveedor sin alternativa
```

Y la forma práctica de hacer el análisis, que cabe en una tabla:

```
elemento ■ ¿qué pasa si cae? ■ ¿si responde mal? ■ detección ■
         ■ ¿cuánto tarda en  ■ ¿quién sabe?      ■ ¿probado? ■
         ■ notarse?          ■                   ■           ■
```

Y la columna que más encuentra es la última:

```
¿PROBADO?
  «sí, en 2023»     → no está probado              ley 22
  «está documentado» → no está probado
  «debería funcionar» → no está probado
```

4. Dónde no invertir

Subir la disponibilidad cuesta, y el coste crece más rápido que el beneficio. Decidir dónde parar es parte del diseño.

```
coste aproximado de cada nueve, misma carga
  99,0 %   →  99,9 %      ×1,3   (reintentos, escalado, alertas)
  99,9 %   →  99,95 %     ×1,6   (multizona real)
  99,95 %  →  99,99 %     ×2,5   (multirregión, ensayos)
  99,99 %  →  99,999 %    ×6+    (y suele ser inalcanzable por
                                  el techo de dependencias)
```

Y el criterio para decidir, que es el mismo de la clase 164:

```
coste de la indisponibilidad al mes
  = minutos esperados × pérdida por minuto

y se compara con el coste de la nueve siguiente
→ si la nueve cuesta 6.400 €/mes y evita 1.900 € de pérdida,
  no se compra
```

Y tres sitios donde casi nunca compensa invertir:

```
CAMINOS QUE NO SON CRÍTICOS
  el panel interno, los informes, la exportación nocturna
  → degradar es correcto

POR ENCIMA DEL TECHO DE DEPENDENCIAS
  si la pasarela de pago da 99,95 %, invertir en llegar al
  99,99 % propio no cambia nada                  clase 181

DONDE EL PROBLEMA ES LA DETECCIÓN
  si la mitad del tiempo caído es tiempo hasta enterarse,
  la inversión rentable es la alerta, no la réplica
  → en la clase 132, la detección era el 70 % del tiempo
```

Y una consecuencia que ordena las prioridades:

```
disponibilidad observada = f(frecuencia, detección, mitigación)

y reducir el tiempo de detección suele ser lo más barato
→ antes de duplicar infraestructura, mira cuánto se tarda
  en enterarse
```

Y la lista de comprobación de la clase:

- ■ están listadas todas las dependencias del flujo crítico
- ■ cada una está marcada como dura o blanda
- ■ las blandas se han probado apagándolas
- ■ el techo está calculado y comparado con lo prometido
- ■ cada redundancia tiene identificada su causa común
- ■ el análisis pregunta también por respuesta lenta o errónea
- ■ están listados los puntos únicos de conocimiento
- ■ están listados los procedimientos nunca ejecutados
- ■ el coste de la nueve siguiente está comparado con la pérdida

```
■ se sabe qué parte del tiempo caído es detección
```

Y el cierre que enlaza con la clase siguiente: la disponibilidad supone que el sistema, cuando está en pie, responde a tiempo. Cuánta carga aguanta antes de que eso deje de ser cierto —y por qué el punto de ruptura llega antes de lo que sugiere el uso de CPU— es la materia de la clase 186.

Ejemplo trabajado

El equipo de reservas calcula el techo de su flujo crítico y hace el análisis de puntos de fallo. Lo que sigue es el cálculo, las conversiones que hicieron, los once puntos únicos encontrados —de los cuales solo tres eran de infraestructura— y la decisión de dónde no invertir.

El cálculo inicial del flujo de reserva, con lo que había:

```
componente              disp.      dura   nota
CDN / balanceador       99,99 %     sí
API de reservas         99,95 %     sí    2 instancias, 1 zona
base de reservas        99,99 %     sí    zonal, sin réplica
servicio de precios     99,90 %     sí    nuevo, clase 184
pasarela de pago        99,95 %     sí    externa, contrato
proveedor de identidad  99,99 %     sí    valida cada petición
servicio de catálogo    99,90 %     sí    ← se creía blando
notificaciones          99,50 %     no    asíncrona
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
techo                   99,68 %
prometido en el QA-2    99,70 %           ← incumplido por
                                            aritmética
```

Y el hallazgo del cálculo:

```
catálogo estaba declarado como dependencia blanda
la prueba negativa de la clase 179 lo desmintió: al inyectar
latencia, el flujo de reserva se caía
→ el código lo llamaba de forma síncrona sin plazo ni
  alternativa                                        ley 22
```

Las conversiones, por orden de coste:

```
1  IDENTIDAD → sin llamada
   validación local del testigo firmado; la llamada solo en
   renovación
   coste   0 €; 3 días de trabajo
   efecto  sale del producto

2  PRECIOS → último valor válido
   caché con validez de 10 min; si no responde, precio anterior
   coste   0 €; riesgo de precio desactualizado 10 min
   acepta  revenue, por escrito
   efecto  sale del producto

3  CATÁLOGO → blando de verdad
   plazo de 300 ms y, si no responde, se sirve la ficha
   cacheada; si no hay caché, se muestra sin detalle
   coste   0 €; 5 días
   efecto  sale del producto

4  BASE DE RESERVAS → multizona
   réplica síncrona en otra zona, conmutación automática
   coste   +840 €/mes
   efecto  99,99 % → 99,995 %

5  API DE RESERVAS → dos zonas
   coste   +310 €/mes
   efecto  99,95 % → 99,98 %

NO CONVERTIBLE
   pasarela de pago: el negocio no acepta reservar sin cobrar
   → queda como techo de 99,95 %
```

El techo después:

```
CDN                   99,99 %
API (2 zonas)         99,98 %
base (multizona)      99,995 %
```

```
pasarela de pago      99,95 %
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
techo                 99,92 %      frente a 99,68 % antes
coste añadido         1.150 €/mes
y de las 5 mejoras, 3 costaron 0 €
```

El análisis de puntos de fallo: once encontrados.

```
INFRAESTRUCTURA (3)
  1  el registro de artefactos es de una sola región
     si cae   no hay despliegues ni escalado con imagen nueva
     probado  no
  2  los certificados caducan el mismo día en los 4 servicios
     si cae   caen los cuatro a la vez → fallo correlacionado
     probado  no
  3  el servicio de secretos es zonal
     si cae   los servicios que arrancan no obtienen credenciales
     probado  no

DATOS (1)
  4  la copia de seguridad está en la misma cuenta
     si cae   un borrado con credencial comprometida se lleva
              las dos                                clase 166
     probado  sí, y falló

RESPUESTA LENTA, NO CAÍDA (2)
  5  una réplica de lectura degradada seguía recibiendo tráfico
     el reparto era rotatorio                       clase 152
  6  la pasarela de pago responde en 8 s en vez de fallar
     el plazo estaba en 30 s → el hilo se ocupaba

CONOCIMIENTO (2)
  7  una sola persona sabe reconstruir el índice de búsqueda
     si no está   entre 4 y 8 h de indisponibilidad de búsqueda
  8  una sola persona conoce la relación entre el inventario
     de consumidores y los informes de negocio       clase 180

PROCEDIMIENTO (3)
  9  el acceso de emergencia nunca se había usado
     probado  sí, en la clase 179, y NO funcionaba
 10  la conmutación de región nunca se había cronometrado
     probado  sí, y tardaba 2 h 10 frente a 1 h declarada
 11  la restauración de la base de precios (nueva) no se
     había ensayado nunca
     probado  no
```

Y el reparto, que es la conclusión de esta clase:

```
de infraestructura               3 de 11        27 %
de datos                         1 de 11         9 %
de respuesta degradada           2 de 11        18 %
de conocimiento                  2 de 11        18 %
de procedimiento                 3 de 11        27 %

→ 8 de 11 no se habrían encontrado mirando el diagrama
```

Las correcciones y su coste:

```
punto   corrección                              coste
1       réplica del registro en 2ª región       120 €/mes
2       renovación escalonada, 3 fechas          0 €
3       secretos regionales                      0 €
4       copia en cuenta separada e inmutable     90 €/mes
5       reparto por menor número de peticiones   0 €
6       plazo a 3 s con reintento                0 €
7       procedimiento escrito + ejecutado por    2 días
        otra persona                             de trabajo
8       inventario de consumidores documentado   3 días
9       acceso de emergencia corregido y con     0 €
        prueba trimestral                        + calendario
10      conmutación optimizada a 38 min          clase 179
11      ensayo de restauración de precios        1 día
```

Y el detalle que resume la clase:

```
de las once correcciones, siete costaron 0 € de infraestructura
y las tres más peligrosas —acceso de emergencia, conocimiento
único y certificados simultáneos— no aparecían en ningún
diagrama de arquitectura
```

Dónde se decidió NO invertir:

```
1  segunda región activa para el flujo de reserva
   subiría de 99,92 % a 99,96 %
   coste                          6.400 €/mes
   pérdida evitada estimada         390 €/mes
   decisión   no; se mantiene la segunda región en frío

2  redundancia del servicio de búsqueda
   la búsqueda degradada permite reservar por referencia
   decisión   no; es dependencia blanda y ya está probado

3  alta disponibilidad del panel interno y los informes
   decisión   no; degradar es correcto

4  subir del 99,92 % con la pasarela de pago al 99,95 %
   decisión   imposible; es el techo. Se negocia en contrato
              o se acepta
```

La lección que esta clase deja: el techo real era 99,68 % frente al 99,70 % prometido, y se incumplía por aritmética antes de que nadie cometiera un error. Subirlo a 99,92 % costó 1.150 €/mes, y tres de las cinco mejoras costaron cero euros: consistieron en dejar de llamar a cosas de forma síncrona. Y de los once puntos únicos de fallo, ocho no eran de infraestructura.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-15-systems-architecture-engineering/185-disponibilidad-confiabilidad-y-analisis-de-puntos-de-fallo/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 failure-mode-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 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 failure-mode-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
Se promete una disponibilidad que se incumple sin que nadie cometa errores	No se calculó el techo que imponen las dependencias duras	Multiplica la disponibilidad de todas las dependencias duras del flujo y compara con lo prometido antes de firmarlo.
Hay redundancia y aun así el servicio cae entero	Fallo correlacionado: misma zona, mismo despliegue, mismo certificado o misma configuración	Por cada redundancia, identifica qué causa tiraría a las dos a la vez; si existe, la fórmula del paralelo no aplica.
Una dependencia declarada blanda tumba la operación	Nunca se probó apagándola	Apaga cada dependencia blanda en una prueba negativa; si la operación se cae, era dura.
Una réplica degradada hace más daño que una caída	El análisis solo preguntó qué pasa si el elemento desaparece	Pregunta también qué pasa si responde mal o lento, y reparte por menor número de peticiones en vuelo.
En un incidente resulta que solo una persona sabía hacer algo	El análisis se limitó a componentes técnicos	Añade las preguntas por punto único de conocimiento y por procedimiento nunca ejecutado.
Se gasta mucho en redundancia y la disponibilidad observada apenas mejora	La mayor parte del tiempo caído es tiempo hasta enterarse	Mide qué fracción del tiempo es detección; si domina, invierte en alertas antes que en réplicas.

■ Seguridad, ética y costo

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

■ Preguntas de comprobación

1. ¿Cómo se calcula el techo de disponibilidad de un flujo con dependencias duras en serie?
2. ¿Qué invalida la fórmula del paralelo y cómo se detecta?
3. ¿Cuáles son las cuatro formas de convertir una dependencia dura en blanda?
4. ¿Qué cuatro preguntas hace un análisis de puntos de fallo que encuentre los no técnicos?
5. ¿En qué tres situaciones no compensa invertir en más disponibilidad?

Referencias

- Beyer, B. y otros (2016). Site Reliability Engineering, cap. «Embracing risk» — aritmética de disponibilidad y coste de cada nueve. <https://sre.google/sre-book/embracing-risk/>
- AWS (2025). Reliability Pillar: failure management — modos de fallo y dependencias. <https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/welcome.html>
- Huang, P. y otros (2017). Gray failure: the Achilles' heel of cloud-scale systems. <https://www.microsoft.com/en-us/research/publication/gray-failure-the-achilles-heel-of-cloud-scale-systems/>
- Nygard, M. (2018). Release It!, 2.ª ed. — mamparos, plazos y fallo en cascada. <https://pragprog.com/titles/mnee2/release-it-second-edition/>
- Google Cloud (2025). Designing resilient systems — dependencias duras, blandas y degradación. <https://cloud.google.com/architecture/framework/reliability>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Anterior	Índice	Siguiente
← 184 · Arquitectura monolítica, modular y de microservicios	Parte 15 · Programa	186 · Capacidad, latencia, throughput y teoría de colas →

Evaluación — 185 Disponibilidad, confiabilidad y análisis de puntos de fallo

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 failure-mode-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.

186 — Capacidad, latencia, throughput y teoría de colas

Parte: 15 — Arquitectura de sistemas e ingeniería de requisitos Nivel: intermedio-avanzado · Horas estimadas: 4
Laboratorio: performance · Estado: EXECUTABLECORE

Propósito

Entender por qué un sistema al 70 % de uso responde bien y al 85 % deja de responder, y poder predecirlo antes de que ocurra. La clase da la teoría de colas mínima que hace falta —ley de Little, la curva de espera frente a utilización y por qué la variabilidad la empeora—, explica la diferencia entre modelo cerrado y abierto en las pruebas de carga, y convierte todo eso en decisiones de capacidad, plazos y concurrencia que se pueden escribir en un diseño.

Resultados de aprendizaje

Al finalizar podrás:

1. Aplicar la ley de Little para relacionar concurrencia, caudal y latencia.
2. Explicar por qué la latencia se dispara cerca de la saturación.
3. Medir el codo con una prueba de modelo abierto y datos realistas.
4. Dimensionar grupos de conexiones, hilos y plazos con criterio.
5. Decidir el margen de capacidad con una cifra y no con una costumbre.

Conceptos centrales

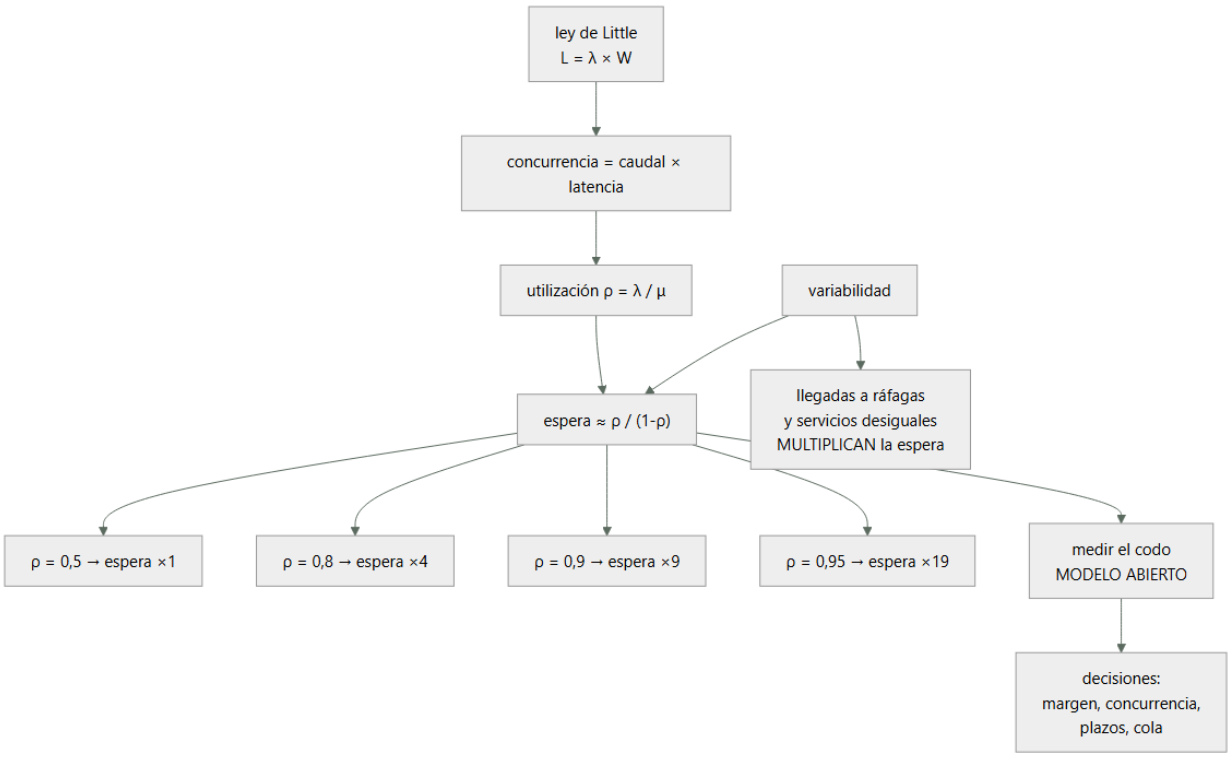
Concepto	Comprensión verificable
ley de Little	L = λ × W. La concurrencia media es el caudal por el tiempo medio en el sistema. Se cumple siempre, sin supuestos.
utilización (ρ)	Fracción del tiempo que el recurso está ocupado. La espera crece como 1/(1-ρ).
codo	Zona de utilización donde la latencia empieza a crecer sin control. Suele estar entre 0,6 y 0,8, no en 0,95.
variabilidad	Dispersión de llegadas y de servicio. Duplicar la variabilidad duplica la espera a la misma utilización.
modelo abierto	Prueba en la que las llegadas no dependen de las respuestas. Es la única que reproduce un pico real.
modelo cerrado	Prueba con N usuarios virtuales que esperan la respuesta. Se autolimita y oculta la saturación.

Modelo mental

Una arquitectura es un conjunto de decisiones sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

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. Ley de Little, que se cumple siempre

La única fórmula de esta clase que no tiene supuestos:

```
L = λ × W

  L   elementos en el sistema (concurrencia media)
  λ   caudal (llegadas por segundo, en régimen estable)
  W   tiempo medio en el sistema (latencia)
```

Y su utilidad práctica es que despeja incógnitas que nadie suele calcular:

```
¿cuántas conexiones a la base necesito?
  λ = 400 consultas/s, W = 25 ms
  L = 400 × 0,025 = 10 conexiones en uso de media
  → el grupo de 200 conexiones no hacía falta
  → y era la causa de la saturación de la base

¿cuántos hilos necesita el servicio?
  λ = 1.200 pet/s, W = 80 ms
  L = 96 peticiones simultáneas

¿qué latencia implica mi límite de concurrencia?
  L = 50 (límite del grupo), λ = 1.000/s
  W = 50 / 1.000 = 50 ms
  → si el trabajo real tarda 80 ms, el caudal máximo es 625/s
    y por encima se forma cola, haga lo que haga el escalado
```

Y la lectura que más decisiones cambia:

```
si la latencia sube y el caudal se mantiene, la concurrencia
SUBE proporcionalmente
→ y los grupos de conexiones, hilos y sockets se agotan
→ por eso una dependencia lenta agota recursos aguas arriba
  aunque no falle nada                                clase 185
```

Y una aplicación al revés, útil para dimensionar colas:

```
si el consumidor procesa 200 msg/s y llegan 250/s
  la cola crece 50/s, indefinidamente
  → el retraso no se estabiliza; no es un pico, es un déficit
  → y el tiempo de recuperación tras un pico de N mensajes
```

```
    es N / (capacidad - llegada)                    clase 117
```

2. Por qué la latencia se dispara

La intuición dice que un sistema al 90 % de uso va «un poco peor» que al 45 %. La realidad es otra.

```
utilización ρ = λ / μ    (llegadas / capacidad de servicio)

espera en cola ≈ (ρ / (1 - ρ)) × tiempo de servicio

  ρ = 0,50   espera ≈ 1,0 × servicio
  ρ = 0,70   espera ≈ 2,3 ×
  ρ = 0,80   espera ≈ 4,0 ×
  ρ = 0,90   espera ≈ 9,0 ×
  ρ = 0,95   espera ≈ 19,0 ×
  ρ = 0,99   espera ≈ 99,0 ×
```

Y la consecuencia operativa:

```
de 0,50 a 0,70 de utilización, la latencia poco más que dobla
de 0,90 a 0,95, se dobla otra vez
→ el sistema no se degrada linealmente: cae por un acantilado
→ y el panel de CPU al 85 % parece «margen» y no lo es
```

La variabilidad, que es lo que hace que el codo llegue antes de lo que dice la fórmula:

```
la fórmula anterior supone llegadas y servicios «medios»
en la realidad
  las llegadas vienen a ráfagas (no son uniformes)
  los tiempos de servicio son desiguales (una consulta pesada
    entre mil ligeras)

y la espera crece con la SUMA de las dos variabilidades
  → duplicar la variabilidad duplica la espera a la misma
    utilización
```

Y de ahí salen tres decisiones de diseño concretas:

```
1. SEPARAR TRABAJOS DESIGUALES
   una cola para lo corto y otra para lo largo
   → una petición de 2 s detrás de la cual esperan 200 de 20 ms
     arruina el percentil de todas                  clase 152

2. SUAVIZAR LAS LLEGADAS
   limitación de ritmo y encolado deliberado
   → una ráfaga admitida es una ráfaga que se paga en p99

3. NO APUNTAR A UTILIZACIÓN ALTA
   el objetivo razonable está en 0,6-0,7 para servicio
   interactivo, no en 0,9
   → 0,9 es para trabajo por lotes, donde la espera no importa
```

Y el efecto que explica los incidentes más confusos:

```
fallo en cascada por cola
  el servicio se ralentiza → sube la concurrencia (Little)
  → se agotan hilos o conexiones → suben los plazos vencidos
  → los clientes reintentan → sube λ
  → el sistema entra en una realimentación que no sale sola

→ y por eso el remedio es RECHAZAR pronto, no esperar más
```

3. Medir el codo: abierto frente a cerrado

La prueba de carga habitual miente, y miente siempre en la misma dirección: hacia el optimismo.

```
MODELO CERRADO
  N usuarios virtuales; cada uno envía, ESPERA la respuesta,
  y envía otra
  → si el sistema se ralentiza, el generador envía menos
  → la carga se autolimita
  → nunca se ve la saturación
  → sirve para simular usuarios con pensamiento, no picos

MODELO ABIERTO
  las llegadas ocurren a un ritmo dado, respondan o no
```

```
  → si el sistema se ralentiza, la cola crece, como en la vida
  → es el único que encuentra el codo
```

Y la diferencia medida, que suele ser de varios múltiplos:

```
en la clase 179, el codo medido con modelo abierto estaba en
1.850 peticiones/s; la prueba cerrada anterior decía 6.000
```

Cómo medir bien, en seis puntos:

```
1. modelo abierto, con ritmo creciente en escalones
2. datos realistas: el catálogo entero, no 1.000 filas
   → la clase 179 tuvo un incidente por probar con 1.000
3. mezcla de operaciones realista, con su proporción
4. con los cachés en estado realista (fríos y calientes)
5. midiendo en el BORDE, no en el servidor          clase 126
6. registrando percentiles, no medias
```

Y qué buscar en el resultado:

```
el punto donde el p99 empieza a crecer más rápido que el p50
  → ese es el codo, y suele estar bastante por debajo del
    punto donde suben los errores

el caudal máximo sostenible
  → no el pico instantáneo

qué recurso se agota primero
  → CPU, memoria, conexiones, hilos, ancho de banda, IOPS
  → casi nunca es la CPU
```

Y el error de interpretación más común:

```
«aguantó 3.000/s»
→ ¿con qué latencia? ¿durante cuánto tiempo? ¿con qué mezcla?
→ un caudal sin latencia asociada no es un dato
```

4. De la cola a las decisiones

Todo lo anterior sirve si se traduce en números que van al diseño.

Margen de capacidad:

```
objetivo de utilización en servicio interactivo   0,6 - 0,7
y el margen debe cubrir
  el tiempo que tarda el escalado en reaccionar     clase 129
  la pérdida de una zona (si son 3, cada una al 66 % máximo)
  el crecimiento hasta la próxima revisión

regla práctica con 3 zonas
  utilización máxima por zona = 0,66 × 0,7 ≈ 0,46
  → parece bajo, y es lo que permite perder una zona sin
    cruzar el codo
```

Concurrencia y grupos:

```
tamaño del grupo de conexiones = L calculado con Little,
  más un margen pequeño
→ un grupo demasiado GRANDE no protege: traslada la
  saturación a la base
→ un grupo pequeño con cola corta y rechazo rápido protege

límite de concurrencia por servicio
  → es la forma más simple de mamparo               clase 153
```

Plazos, que son decisiones de cola:

```
un plazo largo mantiene ocupada la concurrencia
  plazo de 30 s con W real de 200 ms = 150 veces el trabajo
  → en saturación, los hilos se llenan de peticiones muertas

regla   plazo ≈ p99 esperado × 3, no más
        y con presupuesto de plazo que se propaga     clase 121
        → si al llamado le quedan 40 ms, que no empiece
```

Rechazo y degradación:

```
en saturación, rechazar rápido es mejor que encolar
```

```
  → devuelve error o respuesta degradada en milisegundos
  → mantiene la latencia de los que sí se atienden

y la señal para rechazar no es la CPU: es la longitud de cola
y el tiempo de espera en ella
```

Y la lista de comprobación de la clase:

- ■ está calculada la concurrencia con la ley de Little
- ■ los grupos de conexiones se dimensionaron con ese cálculo
- ■ el objetivo de utilización está entre 0,6 y 0,7, no en 0,9
- ■ el margen cubre la pérdida de una zona
- ■ el codo se midió con modelo abierto
- ■ la prueba usó volumen y mezcla realistas
- ■ se midió en el borde y por percentiles
- ■ se sabe qué recurso se agota primero
- ■ los trabajos largos y cortos no comparten cola
- ■ los plazos son múltiplos pequeños del p99 esperado
- ■ hay rechazo rápido, y su señal es la cola, no la CPU

Y el cierre que enlaza con la clase siguiente: la teoría de colas describe un sistema que responde a tiempo. Qué significa que responda lo correcto cuando hay varias copias del dato, y qué hay que renunciar a cambio, es la materia de la clase 187.

Ejemplo trabajado

El servicio de búsqueda de la clase 184 se cae en cada campaña pese a tener CPU de sobra. Lo que sigue es el diagnóstico con la ley de Little, la medición del codo con modelo abierto —que dio la tercera parte de lo que decía la prueba anterior— y las cinco decisiones que salieron.

El síntoma:

```
en campaña, a partir de ~1.900 búsquedas/s
  p50 pasa de 60 ms a 90 ms          ← poco
  p99 pasa de 180 ms a 4.200 ms      ← acantilado
  CPU de las instancias                       48 %
  memoria                                     51 %
  errores                                     0,02 %

«hay CPU de sobra, no entiendo qué pasa»
```

Diagnóstico con la ley de Little.

```
en régimen normal
  λ = 900 búsquedas/s
  W = 60 ms
  L = 900 × 0,060 = 54 peticiones simultáneas

en campaña, antes del acantilado
  λ = 1.850/s
  W = 90 ms
  L = 1.850 × 0,090 = 166 simultáneas

y el límite real del servicio
  grupo de conexiones al almacén         200
  hilos de trabajo                       200
  → a 166 ya se está al 83 % del grupo
```

Y entonces la utilización del recurso que importa:

```
el recurso saturado NO era la CPU: era el grupo de conexiones

ρ del grupo a 1.850/s                 0,83
espera esperada ≈ 0,83/(1-0,83)       ×4,9 sobre el servicio
→ y a 2.100/s, ρ = 0,94 → ×15,7

→ eso explica el p99 de 4.200 ms con CPU al 48 %
```

La variabilidad, que era la mitad del problema.

```
al separar las búsquedas por tipo
  búsqueda por ciudad y fechas       92 % · 40 ms de media
  búsqueda con filtros múltiples      7 % · 210 ms
  búsqueda sin filtros («todo»)       1 % · 1.900 ms
```

```
el 1 % de consultas largas ocupaba
  1.850 × 0,01 × 1,9 = 35 conexiones de las 200
  → el 17,5 % del grupo para el 1 % del tráfico

y peor: bloqueaban la cola de las cortas
```

La medición del codo, hecha dos veces.

```
PRUEBA ANTERIOR (modelo cerrado, 500 usuarios virtuales)
  resultado   «aguanta 6.000/s»
  por qué mentía
    los usuarios virtuales esperaban la respuesta
    al ralentizarse el sistema, enviaban menos
    la carga real nunca superó 2.400/s efectivos
    y la latencia media reportada era 200 ms, sin percentiles

PRUEBA NUEVA (modelo abierto, escalones de 200/s cada 3 min)
  datos       catálogo completo, 4,1 M de registros
  mezcla      92/7/1 como en producción
  caché       precalentado al 60 %, como en un pico real
  medida      en el borde, p50/p95/p99

  1.200/s     p50 58 ms   p99 165 ms    ρ grupo 0,42
  1.600/s     p50 62 ms   p99 210 ms    ρ grupo 0,58
  1.850/s     p50 90 ms   p99 480 ms    ρ grupo 0,83   ← codo
  2.100/s     p50 140 ms  p99 4.200 ms  ρ grupo 0,94
  2.400/s     p50 900 ms  p99 timeout   ρ grupo 1,00

  codo real                    ~1.750/s
  lo que decía la prueba vieja  6.000/s
  factor de error               ×3,4
```

Las cinco decisiones.

```
1  SEPARAR COLAS POR TIPO DE TRABAJO
   grupo de 160 conexiones para búsquedas cortas
   grupo de 24 para filtros múltiples
   grupo de 6 para búsquedas sin filtros, con cola propia
   efecto   el 1 % pesado deja de arruinar el p99 del 92 %
   coste    0 €

2  REDIMENSIONAR EL GRUPO CON LITTLE, NO A OJO
   objetivo   ρ ≤ 0,65 a 2.500/s con W de 60 ms
   L = 2.500 × 0,060 = 150
   grupo = 150 / 0,65 ≈ 230
   pero el almacén solo admite 250 conexiones en total
   → decisión: 230 no cabe con 3 réplicas del servicio
   → se añade una réplica de lectura y se reparte

3  PLAZOS PROPORCIONALES
   antes    plazo único de 30 s
   después  cortas 400 ms, filtros 900 ms, sin filtros 5 s
   razón    30 s con W de 60 ms ocupaba una conexión 500 veces
            el trabajo útil durante la saturación
   efecto   en el pico, las conexiones dejan de llenarse de
            peticiones muertas

4  RECHAZO RÁPIDO POR LONGITUD DE COLA
   señal    espera en cola > 150 ms, no CPU
   acción   responder con resultados cacheados o error 503
            con reintento sugerido
   efecto   el p99 de los atendidos se mantiene bajo 500 ms

5  MARGEN DE CAPACIDAD CON CRITERIO
   pico observado                    1.850/s
   crecimiento previsto a 12 meses   +35 %  → 2.500/s
   objetivo de utilización           0,65
   pérdida de una zona de tres       capacidad × 0,66
   capacidad necesaria = 2.500 / (0,65 × 0,66) ≈ 5.830/s
   → 4 réplicas por zona en lugar de 3
   coste                             +680 €/mes
```

El resultado, en la campaña siguiente:

```
pico atendido                    1.850/s      3.100/s
```

```
p50 en el pico                     90 ms        61 ms
p99 en el pico                  4.200 ms       290 ms
CPU en el pico                     48 %         44 %
ρ del grupo de conexiones          0,94         0,61
peticiones rechazadas              0 %          0,4 %  ← a propósito
incidentes de búsqueda en campaña      3            0
```

Y la línea que resume el diagnóstico:

```
el sistema no tenía un problema de capacidad de cómputo
tenía un problema de COLA, y el panel de CPU no podía verlo
```

Lo que se descubrió por el camino y no se buscaba:

```
la búsqueda «sin filtros» la generaba un enlace de una campaña
de marketing de 2022 que seguía activo en un correo antiguo
→ 18 peticiones/s de un enlace que nadie mantenía    ley 20
→ al retirarlo, el 1 % pesado bajó al 0,3 %
```

La lección que esta clase deja: el servicio se caía con la CPU al 48 % porque el recurso saturado era el grupo de conexiones, y la relación entre concurrencia, caudal y latencia se podía calcular con una multiplicación antes de tocar nada. Y la prueba de carga que decía «aguanta 6.000/s» no estaba mal ejecutada: estaba midiendo otra cosa, porque el modelo cerrado se autolimita y nunca llega al codo.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-15-systems-architecture-engineering/186-capacidad-latencia-throughput-y-teoria-de-colas/lab.py
```

El laboratorio selecciona el motor de práctica performance 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 capacity-model 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 prueba de carga con baseline y cuello de botella. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye capacity-model 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 sistema se cae con la CPU al 50 %	El recurso saturado es otro: conexiones, hilos, IOPS o ancho de banda	Calcula la concurrencia con la ley de Little y mide la utilización del recurso que realmente limita.
La latencia se dispara sin que suba mucho el tráfico	El sistema entró en la zona de utilización alta, donde la espera crece como 1/(1-ρ)	Fija el objetivo de utilización entre 0,6 y 0,7 y dimensiona el margen para perder una zona.
El p99 es pésimo aunque el p50 esté bien	Trabajos largos y cortos comparten la misma cola	Separa colas y grupos por tipo de trabajo, con límites propios.
La prueba de carga dice que aguanta el triple de lo que aguanta	Modelo cerrado: los usuarios virtuales esperan la respuesta y la carga se autolimita	Mide con modelo abierto, volumen y mezcla realistas, en el borde y por percentiles.
En saturación los hilos se llenan de peticiones que ya no interesan a nadie	Plazos muy largos comparados con el trabajo útil	Plazo del orden de tres veces el p99 esperado, con presupuesto de plazo propagado.
Un pico deja al sistema en un bucle del que no sale solo	Realimentación: cola, plazos vencidos y reintentos suben el caudal	Rechaza rápido usando la longitud de cola como señal, y limita los reintentos con retroceso.

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 relaciona la ley de Little y para qué sirve al dimensionar un grupo de conexiones?
2. ¿Por qué la espera crece de forma no lineal con la utilización?
3. ¿Cómo afecta la variabilidad a la latencia a igual utilización?
4. ¿Por qué una prueba de modelo cerrado no encuentra el codo?
5. ¿Qué señal debe disparar el rechazo rápido y por qué no la CPU?

Referencias

- Little, J. D. C. (1961). A proof for the queuing formula L = λW. <https://pubsonline.informs.org/doi/10.1287/opre.9.3.383>
- Gunther, N. (2007). Guerrilla Capacity Planning — utilización, escalabilidad y sus límites. <https://link.springer.com/book/10.1007/978-3-540-31010-5>
- Schroeder, B., Wierman, A. y Harchol-Balter, M. (2006). Open versus closed: a cautionary tale. <https://www.usenix.org/legacy/event/nsdi06/tech/schroeder.html>
- Harchol-Balter, M. (2013). Performance Modeling and Design of Computer Systems — colas y variabilidad. <https://www.cambridge.org/core/books/performance-modeling-and-design-of-computer-systems/8E5B6F1D9E2E1E4B2A4C0B9B1B0E3E0D>
- Google (2016). SRE Book: addressing cascading failures — realimentación, rechazo y plazos. <https://sre.google/sre-book/addressing-cascading-failures/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 15 en PDF · Manual integral

Evaluación — 186 Capacidad, latencia, throughput y teoría de colas

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 capacity-model 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.

187 — Consistencia, particiones, relojes y consenso

Parte: 15 — Arquitectura de sistemas e ingeniería de requisitos Nivel: intermedio-avanzado · Horas estimadas: 4
Laboratorio: distributed · Estado: EXECUTABLECORE

Propósito

Decidir qué se garantiza cuando hay varias copias del dato y la red puede partirse. La clase pone en su sitio el teorema CAP —que se cita mucho y se aplica mal—, explica por qué los relojes no sirven para ordenar sucesos, presenta el consenso como la herramienta cara que resuelve el acuerdo, y sostiene lo que este programa lleva demostrando desde la clase 149: la consistencia se decide por operación, nunca para el sistema entero.

Resultados de aprendizaje

Al finalizar podrás:

1. Enunciar CAP correctamente y evitar los dos errores habituales al citarlo.
2. Elegir el modelo de consistencia por operación, con su justificación de negocio.
3. Explicar por qué los relojes de pared no ordenan sucesos distribuidos.
4. Reconocer cuándo hace falta consenso y cuánto cuesta.
5. Diseñar con consistencia eventual sin que el usuario vea incoherencias.

Conceptos centrales

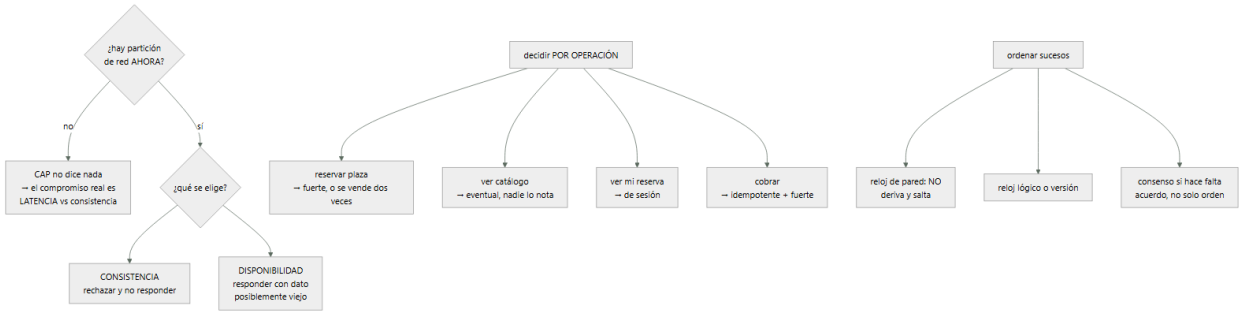
Concepto	Comprensión verificable
CAP	Bajo partición de red hay que elegir entre responder (disponibilidad) y garantizar la última escritura (consistencia). Solo aplica durante la partición.
linealizabilidad	Toda lectura ve la última escritura confirmada. La garantía más fuerte y la más cara.
consistencia eventual	Sin escrituras nuevas, todas las réplicas convergen. No dice cuándo.
consistencia de sesión	Un cliente ve sus propias escrituras y no retrocede en el tiempo. Suele ser lo que el usuario percibe como correcto.
reloj lógico	Contador que ordena sucesos por causalidad sin depender de la hora.
consenso	Acuerdo entre réplicas sobre un valor. Requiere mayoría y cuesta al menos un viaje de red.

Modelo mental

Una arquitectura es un conjunto de decisiones sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

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. CAP, bien enunciado

El teorema se cita constantemente y casi siempre mal. Su enunciado correcto es estrecho:

```
CUANDO LA RED SE PARTE y dos grupos de réplicas no pueden
hablarse, un sistema debe elegir entre
  responder a las peticiones (disponibilidad), aceptando que
    la respuesta pueda no reflejar la última escritura
  o garantizar la última escritura (consistencia), aceptando
    no responder
```

Y los dos errores habituales:

```
ERROR 1   «elegimos AP» o «somos CP» como propiedad del sistema
  → la elección se hace por operación, y solo importa DURANTE
    la partición
  → el mismo sistema puede ser fuerte al reservar y eventual
    al listar

ERROR 2   creer que sin partición hay que elegir
  → sin partición se puede tener las dos
  → pero se paga en LATENCIA, que es el compromiso que
    importa el 99,99 % del tiempo
```

Y por eso la formulación práctica es PACELC, que sí describe el día a día:

```
si hay Partición   →  elegir entre Availability y Consistency
Else (lo normal)   →  elegir entre Latency y Consistency

→ una escritura con confirmación en tres regiones es coherente
  y cuesta ~120 ms
→ la misma con confirmación local cuesta ~2 ms y puede perderse
```

Y la observación de este programa:

```
casi ninguna discusión de CAP en un proyecto real trata de
particiones: trata de si se acepta pagar latencia
→ y disfrazarlo de CAP impide medir el compromiso
```

El espectro de garantías, de más fuerte a más débil:

```
LINEALIZABLE          toda lectura ve la última escritura
  coste   mayoría en cada operación; latencia de la red

SERIALIZABLE          las transacciones equivalen a un orden
  coste   coordinación; conflictos y reintentos

DE SESIÓN             veo mis escrituras, no retrocedo
  coste   afinidad o testigo de versión; barato

MONÓTONA              nunca leo algo más viejo que lo ya leído
  coste   muy bajo

EVENTUAL              converge, sin plazo
  coste   el más bajo; y el que más sorprende al usuario
```

Y el punto que cambia diseños:

```
la mayoría de los sistemas que «necesitan consistencia fuerte»
necesitan CONSISTENCIA DE SESIÓN
```

```
→ el usuario no compara su vista con la de otro usuario:
  compara con lo que él mismo acaba de hacer
```

2. Los relojes no ordenan nada

Ordenar sucesos por la hora de la máquina es una fuente inagotable de errores sutiles.

```
PROBLEMAS DEL RELOJ DE PARED
  deriva entre máquinas: decenas de ms es normal
  saltos hacia atrás al sincronizar
  segundos intercalares y ajustes
  máquinas virtuales que se congelan y despiertan

CONSECUENCIAS
  «el último gana» elige al que tenía el reloj adelantado
  dos sucesos con la misma marca de tiempo
  un suceso posterior con marca anterior
  registros que parecen fuera de orden y no lo están
```

Y el fallo clásico, con la evidencia habitual:

```
resolución de conflictos por marca de tiempo
  A escribe a las 10:00:00,120 (reloj adelantado 80 ms)
  B escribe a las 10:00:00,090 (reloj correcto), DESPUÉS
  → gana A, y la escritura real más reciente se pierde
  → en silencio                                        ley 13
```

Lo que sí ordena:

```
RELOJ LÓGICO (Lamport)
  contador que se incrementa y se propaga
  → da orden causal: si A causó B, A < B
  → no distingue sucesos concurrentes

RELOJ VECTORIAL
  un contador por réplica
  → detecta la concurrencia: dice «estos dos son conflictivos»
  → coste: tamaño proporcional al número de réplicas

NÚMERO DE VERSIÓN POR ENTIDAD
  el más práctico; base de la escritura condicionada
  → «actualiza si la versión sigue siendo 7»       clase 149

RELOJ CON INCERTIDUMBRE ACOTADA
  el sistema conoce su error máximo y espera ese margen
  → lo usan bases distribuidas con hardware de tiempo
  → permite orden global pagando latencia
```

Y la regla práctica:

```
no uses la hora para decidir quién gana
úsala para diagnosticar, y aun así con desconfianza
→ para decidir, usa versiones o consenso
```

3. Consenso: qué resuelve y qué cuesta

El consenso resuelve un problema concreto: que varias réplicas acuerden un valor aunque algunas fallen. No resuelve la latencia ni el particionado de datos.

```
CÓMO FUNCIONA, en una frase
  una mayoría de réplicas debe confirmar antes de considerar
  un valor decidido

POR QUÉ MAYORÍA
  dos mayorías cualesquiera se solapan
  → imposible que dos grupos decidan cosas distintas

CONSECUENCIA DIRECTA
  con 3 réplicas se tolera 1 fallo
  con 5 réplicas se toleran 2
  con 2 réplicas se tolera 0    ← el error más común
```

Y el coste, que es lo que decide dónde se usa:

```
cada decisión cuesta al menos un viaje a la mayoría
  misma zona          ~1 ms
```

```
  entre zonas         ~2-4 ms
  entre regiones      ~30-120 ms

→ por eso el consenso entre regiones se usa para METADATOS
  (quién es el líder, qué configuración está activa) y no
  para cada escritura de negocio
```

Y las cuatro cosas para las que realmente se usa:

```
1. elegir líder (quién escribe)
2. acordar la configuración del grupo
3. registrar un orden total de operaciones (registro replicado)
4. bloqueos y arrendamientos distribuidos
```

Y una advertencia sobre los arrendamientos, que es donde más se falla:

```
un arrendamiento con expiración NO garantiza exclusión mutua
si el titular se congela y despierta
→ hace falta un testigo monótono que el almacén compruebe
→ sin eso, dos procesos se creen líderes a la vez
```

Cuándo NO hace falta consenso, que es la mayoría de las veces:

```
si la operación es idempotente y conmutativa
  → basta con entregar al menos una vez              clase 117
si el conflicto se puede detectar y resolver después
  → versiones y reconciliación
si hay un solo escritor por dato
  → no hay nada que acordar                           ley 21
```

Y esa última línea es la más importante de la clase:

```
la mayoría de los problemas de consistencia se evitan
decidiendo bien QUIÉN ESCRIBE, no añadiendo coordinación
```

4. Consistencia eventual sin que se note

La consistencia eventual es barata y correcta para casi todo, y produce quejas solo cuando se implementa sin cuidar la percepción.

```
LO QUE EL USUARIO NOTA
  hace un cambio y no lo ve            ← lo más grave
  ve un valor y luego uno más viejo    ← retroceso
  dos pantallas muestran cosas distintas a la vez
  un contador que baja

LO QUE EL USUARIO NO NOTA
  que su cambio tarde 200 ms en verse en OTRA sesión
  que un panel agregado vaya 5 s por detrás
  que un buscador tarde 30 s en indexar
```

Y las técnicas que resuelven cada caso:

```
VER LO PROPIO                consistencia de sesión
  leer del primario tras escribir, durante N segundos
  o llevar un testigo de versión en la sesión

NO RETROCEDER               lectura monótona
  fijar la réplica por sesión, o exigir versión ≥ la última
  vista

ESCRITURA PROPIA VISIBLE YA  actualización optimista en cliente
  mostrar el resultado esperado y corregir si falla
  ojo   si falla a menudo, es peor que esperar

CONTADORES                   tipos que convergen (CRDT) o
  agregación con marca de «aproximado»

OPERACIONES CRÍTICAS         fuerte, y solo esas
```

Y una regla de diseño que evita la mitad de los problemas:

```
no mezcles en la misma pantalla datos con garantías distintas
sin decirlo
→ «disponibilidad actualizada hace 12 s» es honesto y suficiente
→ mostrar un dato viejo como si fuera actual no lo es
```

Y la lista de comprobación de la clase:

- cada operación tiene su nivel de consistencia decidido
- el nivel está justificado por consecuencia de negocio
- no se cita CAP fuera de una partición real
- el compromiso normal (latencia vs consistencia) está medido
- ningún conflicto se resuelve por marca de tiempo
- hay versión por entidad y escritura condicionada
- el consenso se usa para metadatos, no para cada escritura
- ningún grupo de consenso tiene 2 miembros
- los arrendamientos usan testigo monótono
- el usuario ve siempre sus propias escrituras
- ninguna lectura retrocede dentro de una sesión
- los datos aproximados se muestran marcados como tales

Y el cierre que enlaza con la clase siguiente: decidir la consistencia de una operación obliga a escribir qué promete cada interfaz y qué pasa cuando cambia. Convertir eso en contratos que puedan evolucionar sin romper a nadie es la materia de la clase 188.

Ejemplo trabajado

El equipo de reservas decide la consistencia de sus once operaciones. Lo que sigue es la tabla de decisiones, los dos errores que había en el sistema anterior —uno de ellos vendía plazas dos veces— y lo que costó cada garantía.

El problema que disparó la revisión:

```
en campaña, 14 reservas duplicadas sobre la misma plaza en
3 días

causa investigada
  la comprobación de disponibilidad leía de una réplica de
  lectura con retraso de 200-900 ms
  dos peticiones simultáneas veían la misma plaza libre
  ambas escribían
  → la restricción de unicidad no existía porque el inventario
    se guardaba como un contador, no como plazas
```

Y el segundo error, más sutil:

```
las modificaciones de reserva se resolvían por marca de tiempo
  «gana la escritura más reciente»
  → dos servidores con 140 ms de deriva
  → 6 casos en el año en que una cancelación se perdía porque
    una modificación anterior tenía marca posterior
  → nadie lo detectó; se atribuyó a error del agente  ley 13
```

La tabla de decisiones, operación por operación.

operación	garantía	por qué
1 reservar plaza	LINEALIZABLE	vender dos veces cuesta dinero y reputación
2 cobrar	fuerte + idempotente	el cobro doble es inaceptable
3 cancelar	fuerte	libera inventario
4 modificar reserva	fuerte con versión	dos agentes editan a la vez
5 ver mi reserva	DE SESIÓN	el usuario debe ver lo que acaba de hacer
6 listar mis reservas	DE SESIÓN	lo mismo
7 buscar disponibilidad	EVENTUAL ≤ 2 s	200 ms de retraso es aceptable; se confirma al reservar
8 ver catálogo	EVENTUAL	cambia 1 vez/mes
9 ver precio	EVENTUAL ≤ 10 min	con validez de 10 min clase 185
10 panel de ocupación	EVENTUAL ≤ 30 s	marcado «hace N s»
11 informe de negocio	EVENTUAL ≤ 24 h	diario

Y la lectura de la tabla:

```
de 11 operaciones, 4 necesitan garantía fuerte
y esas 4 son el 3 % del tráfico
→ pagar consistencia fuerte para todo habría costado
  latencia en el 97 % restante sin ninguna ventaja
```

Cómo se implementó cada garantía, y qué costó.

```
1  RESERVAR PLAZA — linealizable
   antes    contador de plazas + lectura de réplica
   después  fila por plaza con clave única
            (hotel, tipo, fecha, nº de plaza)
            escritura condicionada: INSERT que falla si existe
            lectura desde el primario en el momento de reservar
   coste    +14 ms de p50 en el paso de confirmación
   efecto   duplicados imposibles por construcción, no por
            comprobación previa
   nota     esta es la corrección que importa: se pasó de
            «comprobar y luego escribir» a «escribir con
            restricción»                                clase 149

2  COBRAR — idempotente
   clave de idempotencia = id de reserva + intento
   la pasarela devuelve el mismo resultado si se repite
   coste    0; ya estaba, pero sin usarse en los reintentos

4  MODIFICAR — versión, no reloj
   antes    resolución por marca de tiempo
   después  cada reserva tiene versión; la escritura envía la
            versión leída y falla si cambió
            el cliente reintenta con el estado nuevo
   coste    0 € y 4 días de trabajo
   efecto   los 6 casos anuales de cancelación perdida
            desaparecen; ahora dan conflicto visible

5-6  VER LO MÍO — sesión
   tras escribir, la sesión lleva un testigo de versión
   las lecturas exigen réplica con versión ≥ testigo
   si ninguna réplica la tiene, lee del primario
   coste    ~4 % de las lecturas van al primario

7  BUSCAR — eventual con límite declarado
   retraso máximo aceptado                    2 s
   alerta si el retraso de réplica supera     5 s     ley 13
   y en la respuesta se indica la hora del dato

9  PRECIO — eventual con validez
   caché de 10 min; si precios no responde, último válido
   → esto es lo que lo convirtió en dependencia blanda  clase 185

10 PANEL — eventual, marcado
   la pantalla dice «ocupación a fecha de hace 18 s»
   → las quejas de «el panel está mal» cayeron a cero sin
     cambiar ni un dato
```

Dónde se usó consenso, y dónde no.

```
SÍ
  elección de primario de la base de reservas   gestionado
  arrendamiento del trabajo de reconciliación   3 réplicas
    con testigo monótono comprobado por el almacén

NO
  en ninguna escritura de negocio
  → porque hay un solo escritor por dato, y con eso no hay
    nada que acordar                                  ley 21
```

Y un error que se corrigió al revisar:

```
el arrendamiento del reconciliador estaba con 2 réplicas
  tolerancia a fallos = 0
  y sin testigo monótono
→ una pausa larga del recolector de basura hizo que dos
  procesos se creyeran titulares en enero
→ se corrigió a 3 réplicas y testigo comprobado en el almacén
```

El compromiso normal, medido, que es lo que CAP no describe:

```
confirmación de escritura de reserva
  local en la zona                        2,1 ms   pérdida posible
  mayoría multizona                       4,8 ms   sin pérdida zonal
  mayoría multirregión                  118 ms    sin pérdida regional

decisión   mayoría multizona
motivo     el objetivo de pérdida es de 1 minuto, no de cero;
           118 ms en cada reserva no compensa       clase 166
```

El resultado, tres meses después:

```
reservas duplicadas por campaña            14           0
cancelaciones perdidas al año               6           0
p50 del paso de confirmación             31 ms       45 ms
p50 de la búsqueda                       58 ms       58 ms
lecturas servidas por el primario           100 %       4 %
quejas por «el panel está mal»            9/mes         0
```

La lección que esta clase deja: de once operaciones solo cuatro necesitaban garantía fuerte, y la corrección que eliminó las reservas duplicadas no fue elegir un nivel de consistencia: fue dejar de guardar el inventario como un contador. Y de las dos quejas más ruidosas, una se resolvió escribiendo en la pantalla la antigüedad del dato.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-15-systems-architecture-engineering/187-consistencia-particiones-relojes-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 consistency-model 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 consistency-model 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 venden dos veces los mismos recursos pese a comprobar disponibilidad antes	Comprobar y luego escribir no es atómico, y la lectura venía de una réplica retrasada	Modela el recurso como fila única y usa escritura condicionada que falle; no compruebes antes, deja que la restricción decida.
Se pierden actualizaciones sin que nadie lo note	Los conflictos se resuelven por marca de tiempo y los relojes derivan	Usa versión por entidad y escritura condicionada; reserva los relojes para diagnóstico.
El usuario hace un cambio y no lo ve	Se lee de una réplica sin consistencia de sesión	Lleva un testigo de versión en la sesión y exige una réplica al menos tan reciente, o lee del primario durante unos segundos.
Cada discusión de arquitectura acaba citando CAP sin llegar a nada	Se aplica CAP fuera de una partición, donde el compromiso real es latencia contra consistencia	Mide el coste en milisegundos de cada nivel de confirmación y decide por operación.
Dos procesos se creen titulares del mismo trabajo	Arrendamiento con expiración sin testigo monótono, o grupo de dos réplicas	Usa un número de mandato monótono que el almacén compruebe, y grupos de consenso impares de al menos tres.
Los usuarios se quejan de que un panel «está mal»	Se muestra un dato eventual como si fuera actual	Indica la antigüedad del dato en la propia pantalla y declara el retraso máximo aceptado, con alerta si se supera.

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 dos errores habituales al citar CAP?
2. ¿Qué describe PACELC que CAP no describe?
3. ¿Por qué no se debe resolver un conflicto por marca de tiempo?
4. ¿Cuántos fallos tolera un grupo de consenso de dos réplicas?
5. ¿Qué garantía suele bastar cuando alguien pide «consistencia fuerte»?

Referencias

- Gilbert, S. y Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. <https://dl.acm.org/doi/10.1145/564585.564601>
- Abadi, D. (2012). Consistency tradeoffs in modern distributed database system design (PACELC). <https://ieeexplore.ieee.org/document/6127847>
- Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. <https://dl.acm.org/doi/10.1145/359545.359563>
- Ongaro, D. y Ousterhout, J. (2014). In search of an understandable consensus algorithm (Raft). <https://raft.github.io/raft.pdf>
- Kleppmann, M. (2017). Designing Data-Intensive Applications, caps. 5 y 9 — replicación, consistencia y consenso. <https://dataintensive.net/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Anterior	Índice	Siguiente
← 186 · Capacidad, latencia, throughput y teoría de colas	Parte 15 · Programa	188 · Contratos de API, eventos y compatibilidad evolutiva →

Evaluación — 187 Consistencia, particiones, relojes 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 consistency-model 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.

188 — Contratos de API, eventos y compatibilidad evolutiva

Parte: 15 — Arquitectura de sistemas e ingeniería de requisitos Nivel: intermedio-avanzado · Horas estimadas: 4
Laboratorio: api · Estado: EXECUTABLECORE

Propósito

Escribir contratos —de API y de eventos— que puedan cambiar durante años sin romper a nadie. La clase define qué forma parte del contrato y qué no (que es donde nacen casi todas las roturas), da las reglas de compatibilidad hacia atrás y hacia delante, explica por qué la versión en la ruta se usa mal y qué hacer en su lugar, y trata la retirada de una versión como un proceso con datos y no como un aviso por correo.

Resultados de aprendizaje

Al finalizar podrás:

1. Delimitar qué forma parte del contrato y qué es detalle interno.
2. Aplicar las reglas de compatibilidad hacia atrás y hacia delante.
3. Versionar sin multiplicar el mantenimiento.
4. Evolucionar esquemas de eventos con productores y consumidores independientes.
5. Retirar una versión con datos de uso y no con un aviso.

Conceptos centrales

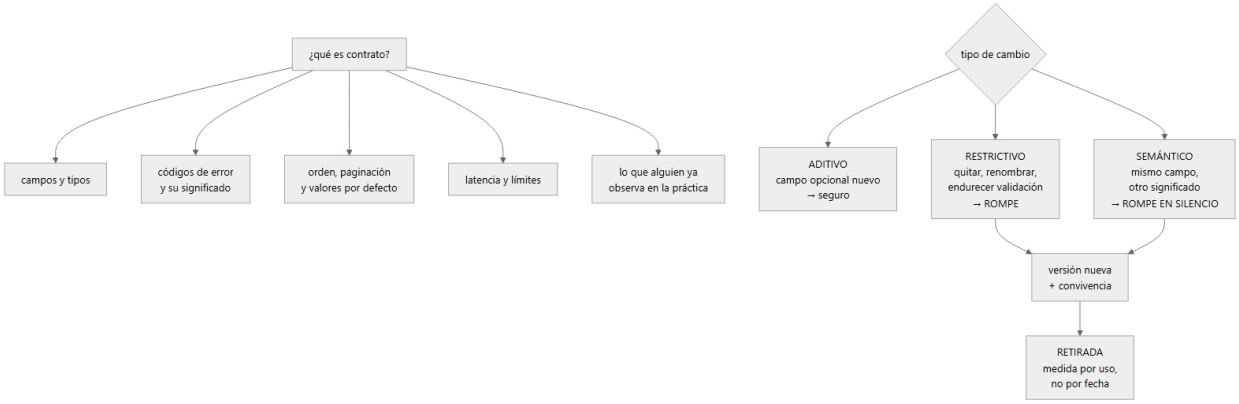
Concepto	Comprensión verificable
contrato	Todo aquello de lo que un consumidor puede depender legítimamente. Incluye más de lo que suele documentarse.
compatibilidad hacia atrás	Un consumidor antiguo sigue funcionando con un productor nuevo.
compatibilidad hacia delante	Un consumidor nuevo funciona con datos producidos por una versión antigua.
cambio aditivo	Añadir algo opcional. Es el único tipo de cambio seguro por defecto.
ley de Hyrum	Con suficientes usuarios, todo comportamiento observable acaba siendo del que alguien depende.
retirada	Proceso de sacar de servicio una versión, guiado por uso medido y no por fechas anunciadas.

Modelo mental

Una arquitectura es un conjunto de decisiones sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

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. Qué es contrato y qué no

Casi todas las roturas vienen de creer que el contrato es solo el esquema. El contrato es aquello de lo que alguien puede depender, y eso es más amplio:

```
EVIDENTEMENTE CONTRATO
  nombres de campos, tipos, obligatoriedad
  rutas, métodos, códigos de estado
  significado de cada código de error

CONTRATO Y CASI NUNCA DOCUMENTADO
  el ORDEN de los resultados sin ordenación explícita
  el tamaño de página por defecto
  qué campos vienen ausentes y cuáles nulos
  el formato exacto de un identificador
  la precisión de un decimal
  que dos llamadas seguidas devuelvan lo mismo
  la latencia habitual
  los límites de ritmo
  el mensaje de texto de un error, si alguien lo compara
```

Y la ley que gobierna esto, con el nombre que se le suele dar:

```
LEY DE HYRUM
  con suficientes usuarios de una interfaz, no importa lo que
  prometa el contrato: cualquier comportamiento observable
  acabará teniendo alguien que dependa de él

→ y por eso los cambios «internos» rompen clientes
```

Y la consecuencia práctica, que es una decisión de diseño:

```
reduce lo observable
  no devuelvas campos que no forman parte del contrato
  ordena explícitamente o aleatoriza a propósito
  no expongas identificadores con estructura adivinable
  declara qué NO es contrato, por escrito
```

Y una técnica que funciona muy bien y se usa poco:

```
rompe a propósito lo que no es contrato, pronto y a menudo
  → variar el orden cuando no se pide ordenación
  → variar el relleno de un identificador opaco
→ así nadie llega a depender de ello                clase 172
```

2. Reglas de compatibilidad

Hay tres tipos de cambio, y solo uno es seguro.

```
ADITIVO — seguro
  añadir un campo OPCIONAL a una respuesta
  añadir un parámetro opcional con valor por defecto
  añadir un valor nuevo a un enumerado ← ojo, ver abajo
  añadir un endpoint nuevo

RESTRICTIVO — rompe, y se nota
```

```
  quitar o renombrar un campo
  hacer obligatorio algo opcional
  endurecer una validación
  cambiar un tipo
  cambiar un código de estado

SEMÁNTICO — rompe EN SILENCIO, y es el peor
  el campo «precio» pasa de incluir impuestos a no incluirlos
  «estado: activo» cambia de significado
  la fecha pasa de local a UTC
  → el consumidor no falla: hace algo incorrecto
```

Y el tercero merece su propia regla:

```
si cambia el SIGNIFICADO, cambia el NOMBRE
  precio → precio_con_impuestos / precio_sin_impuestos
→ un campo con significado nuevo y nombre viejo es una trampa
```

Los enumerados, que rompen más de lo que parece:

```
añadir un valor nuevo es aditivo para el productor
y RESTRICTIVO para el consumidor que hace un switch exhaustivo

regla   los consumidores deben tolerar valores desconocidos
        desde el primer día, y el contrato debe decirlo
```

Las dos direcciones, que hacen falta cuando productor y consumidor se despliegan por separado:

```
HACIA ATRÁS   consumidor viejo + productor nuevo
  → necesaria siempre que el productor despliegue primero

HACIA DELANTE consumidor nuevo + datos viejos
  → necesaria siempre que el consumidor despliegue primero
  → y en eventos, SIEMPRE, porque hay datos históricos

y como no se controla el orden, hacen falta las dos
```

Y el patrón que resuelve los cambios restrictivos sin versión nueva:

```
EXPANDIR Y CONTRAER, en tres despliegues
  1  añadir lo nuevo, manteniendo lo viejo; escribir en ambos
  2  migrar consumidores a lo nuevo, y MEDIR que nadie usa
     lo viejo
  3  retirar lo viejo

→ funciona para campos, tablas, colas y endpoints
→ y el paso 2 es el que se salta y el que da los sustos
```

3. Versionar sin multiplicar el trabajo

La versión en la ruta —/v1/, /v2/— es lo más usado y suele aplicarse mal:

```
EL ERROR
  v2 se crea como copia de v1 y se mantienen las dos enteras
  → dos bases de código, dos conjuntos de pruebas, dos alertas
  → y v1 no se retira nunca                              ley 23

LO QUE SUELE FUNCIONAR MEJOR
  una sola implementación, con una capa de traducción fina
  para las versiones antiguas
  → la lógica vive una vez; la versión solo transforma
```

Y los mecanismos, con su uso adecuado:

```
VERSIÓN EN LA RUTA        cambios grandes y poco frecuentes
  fácil de enrutar y de medir; visible en los registros

VERSIÓN POR CABECERA      evolución continua
  menos visible; exige disciplina de medida

SIN VERSIÓN, SOLO ADITIVO cuando se puede sostener
  el mejor de todos si la disciplina aguanta

SELECCIÓN DE CAMPOS       el consumidor pide lo que quiere
  reduce el acoplamiento; complica el caché
```

Y la regla de cuántas versiones mantener:

```
dos, como máximo, salvo obligación contractual
→ y con una fecha de retirada de la antigua ya fijada al
  publicar la nueva
```

En eventos, el problema es distinto porque los datos persisten:

```
los mensajes antiguos siguen existiendo en el registro
y se reprocesan meses después                          clase 116

→ hace falta compatibilidad hacia delante SIEMPRE
→ y un registro de esquemas que valide en el PRODUCTOR,
  no en el consumidor                                  clase 148

reglas mínimas para un esquema de evento
  todo campo nuevo, opcional y con valor por defecto
  nunca reutilizar un nombre de campo retirado
  nunca cambiar el tipo de un campo
  el identificador del suceso y su clave, inmutables
  la versión del esquema, en el propio mensaje
```

Y una decisión de diseño de eventos que evita la mitad de los cambios:

```
publica HECHOS, no órdenes ni estado completo
  «reserva_confirmada» con su identificador y sus datos propios
  no «actualiza el panel de ocupación»
→ un hecho no cambia; la interpretación sí, y esa es del
  consumidor                                            clase 148
```

4. Retirar de verdad

Anunciar una retirada no retira nada. Este programa ha visto varias veces la misma escena: la versión antigua sigue viva años después del aviso.

Lo que no funciona:

```
el correo a «todos los equipos»
la nota en la documentación
la fecha en el registro de cambios
→ nadie lee nada de eso hasta que se rompe
```

Lo que funciona, en orden:

```
1. MEDIR QUIÉN LA USA
   por consumidor identificado, no solo volumen total
   → sin identidad por consumidor, no hay retirada posible

2. CONTACTAR A LOS QUE QUEDAN, con su nombre
   y saber por qué no han migrado
   → a veces la razón es que lo nuevo no cubre su caso

3. DEGRADAR PROGRESIVAMENTE
   añadir latencia artificial creciente
   devolver una cabecera de aviso
   apagar durante ventanas cortas y anunciadas
   → los «apagados de ensayo» encuentran a los que no responden
     al correo

4. APAGAR, con vuelta atrás preparada

5. BORRAR EL CÓDIGO
   → sin esto, sigue costando mantenimiento           ley 23
```

Y el dato que decide cuándo se puede apagar:

```
no «pasó la fecha»
sino «el uso es cero durante N días, y los consumidores
identificados han confirmado»
```

Y una advertencia sobre el paso 1:

```
si las llamadas llegan sin identidad de consumidor, no se
puede saber a quién romper
→ exigir identidad por consumidor desde el primer día es
  lo que hace posible retirar después
```

Las pruebas de contrato, que son lo que impide romper sin querer:

```
el consumidor declara qué espera
el productor ejecuta esas expectativas en su canalización
→ si un cambio del productor rompe a un consumidor, falla
  la canalización del productor, no la producción
→ y el catálogo de consumidores deja de ser una suposición
```

Y la lista de comprobación de la clase:

- ■ está escrito qué NO forma parte del contrato
- ■ no se devuelven campos que no son contrato
- ■ el orden es explícito o deliberadamente variable
- ■ todo cambio está clasificado: aditivo, restrictivo, semántico
- ■ ningún campo cambia de significado conservando el nombre
- ■ los consumidores toleran valores de enumerado desconocidos
- ■ los cambios restrictivos usan expandir y contraer
- ■ el paso intermedio se cierra midiendo uso, no suponiendo
- ■ los eventos son hechos y su esquema solo crece
- ■ hay registro de esquemas validando en el productor
- ■ cada llamada trae identidad de consumidor
- ■ hay pruebas de contrato en la canalización del productor
- ■ cada versión publicada nace con fecha de retirada
- ■ la retirada se decide por uso medido y termina borrando código

Y el cierre que enlaza con la clase siguiente: un contrato define qué se promete a quien llama, pero no dice quién puede llamar ni qué pasa si el que llama no es quien dice ser. Modelar eso —amenazas y confianza— es la materia de la clase 189.

Ejemplo trabajado

La plataforma de reservas tiene una API v1 usada por la app móvil, tres socios y un sistema interno. Lo que sigue es la evolución de un año: dos cambios semánticos que rompieron en silencio, la migración de un campo con expandir y contraer, y la retirada de v1 que llevaba tres años anunciada.

Punto de partida:

```
api-v1    publicada en 2021, «a retirar» desde 2023
api-v2    publicada en 2023
consumidores identificados                    ninguno
→ las llamadas llegaban sin cabecera de consumidor
→ por tanto, imposible saber a quién se rompía
```

Primer incidente: cambio semántico, marzo.

```
cambio    el campo «precio» pasó a incluir impuestos
motivo    unificar con la web, que ya los incluía
revisión  aprobada; se consideró «no rompe, mismo tipo»

consecuencia
  un socio facturó 3 semanas con precios un 21 % más altos
  no hubo ningún error: el sistema funcionaba
  se detectó por una reclamación del socio
  coste    4.100 € de abonos y una relación tocada

qué lo habría evitado
  regla: si cambia el significado, cambia el nombre
  → precio_sin_impuestos + precio_con_impuestos, y retirar
    «precio» con expandir y contraer
```

Segundo incidente: contrato no escrito, mayo.

```
cambio    se sustituyó la consulta de listado y el resultado
          dejó de venir ordenado por fecha
motivo    optimización; la consulta no pedía ordenación

consecuencia
  la app móvil mostraba las reservas desordenadas
  la app NUNCA había pedido ordenación: dependía del orden
  accidental de la base                          Hyrum
  1 versión de app rota; 9 días hasta la corrección en tiendas

decisión posterior
  todos los listados ordenan explícitamente y lo declaran
  y en los entornos de prueba, el orden se ALEATORIZA a
  propósito cuando no se pide
```

La migración del campo «precio», hecha bien:

```
despliegue 1   se añaden precio_sin_impuestos y
               precio_con_impuestos; «precio» se mantiene
               con su significado ORIGINAL
               y se marca como en retirada en la respuesta

medida         se instrumenta qué consumidores leen «precio»
               → requiere identidad por consumidor  ← ver abajo

despliegue 2   migran los consumidores
               app móvil       semana 3
               socio A         semana 5
               socio B         semana 6
               socio C         semana 14   ← el que faltaba
               interno         semana 4

cierre         14 días con 0 lecturas de «precio»

despliegue 3   «precio» retirado de la respuesta
tiempo total   18 semanas
```

Identidad por consumidor: el cambio que hizo posible todo lo demás.

```
antes   las llamadas llegaban con un testigo, sin identificar
        la aplicación cliente
después cada consumidor tiene su propia credencial y una
        cabecera obligatoria de identificación

qué reveló el primer mes
  consumidores esperados                              5
  consumidores reales                                11

  los 6 no esperados
    un panel interno de finanzas creado en 2022
    dos scripts de analítica de marketing
    un servicio de un socio que se creía retirado
    la herramienta de pruebas de un proveedor externo
    un trabajo programado del propio equipo

→ y dos de ellos usaban v1                             ley 20
```

La retirada de v1, tres años después del aviso.

```
situación al empezar
  peticiones/día a v1                            41.000
  anuncios de retirada enviados desde 2023            7
  efecto de esos anuncios                       ninguno

PASO 1 · medir por consumidor
  app móvil, versiones < 4.2                     38.100/día
  socio C                                         2.700/día
  panel de finanzas                                 190/día
  script de analítica                                10/día

PASO 2 · contactar con nombre
  app: el 6 % de instalaciones seguía en < 4.2, y no se
       actualiza sola en dispositivos antiguos
  socio C: no había migrado porque v2 no devolvía un campo
       que necesitaba          ← razón legítima, y desconocida
  finanzas: no sabían que existía ese panel
  analítica: script de una persona que ya no está     ley 20

PASO 3 · degradar progresivamente
  semana 1   cabecera de aviso y registro por consumidor
  semana 3   +200 ms de latencia artificial
  semana 5   +800 ms
  semana 7   apagado de ensayo de 15 min, anunciado
             → aparecieron 2 consumidores más que nadie
               había detectado
  semana 9   apagado de 2 h
             → el panel de finanzas se quejó; ahí se supo
               quién lo mantenía

PASO 4 · apagar
```

```
  semana 14, con vuelta atrás preparada y sin usarla
  uso residual en el momento del apagado          0,3 %
  → clientes de app muy antiguos, con mensaje de
    actualización obligatoria

PASO 5 · borrar el código
  semana 16
  líneas eliminadas                            11.400
  pruebas eliminadas                              340
  alertas retiradas                                 9
  ahorro de mantenimiento estimado     1,4 días/mes
```

Y los dos hallazgos del proceso:

```
el apagado de ensayo de 15 minutos encontró en una tarde
dos consumidores que siete correos en tres años no habían
encontrado                                          ley 22

y la razón real por la que el socio C no migraba no era
desidia: v2 no cubría su caso. Nadie lo había preguntado
```

Las pruebas de contrato que se montaron después:

```
cada consumidor declara sus expectativas
  app móvil        14 expectativas
  socio A           6
  socio B           9
  socio C           7
  interno          11

se ejecutan en la canalización del PRODUCTOR
resultado en los 6 meses siguientes
  cambios bloqueados antes de salir                  5
  de ellos, cambios semánticos                       2
  incidentes de contrato en producción               0
```

La lección que esta clase deja: los dos incidentes del año no fueron por cambios que rompieran nada visible —el sistema respondió 200 en los dos casos— sino por un significado que cambió conservando el nombre y por un orden del que alguien dependía sin que estuviera en el contrato. Y la retirada de v1 no avanzó en tres años de anuncios y se resolvió en catorce semanas en cuanto hubo identidad por consumidor y un apagado de quince minutos.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-15-systems-architecture-engineering/188-contratos-de-api-eventos-y-compatibilidad-evolutiva/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 contract-evolution 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 contract-evolution 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 consumidor empieza a calcular mal sin que aparezca ningún error	Cambio semántico: el campo mantuvo el nombre y cambió el significado	Si cambia el significado, cambia el nombre; retira el antiguo con expandir y contraer.
Un cambio interno inofensivo rompe a un cliente	El cliente dependía de un comportamiento observable que no estaba en el contrato	Declara qué no es contrato, no devuelvas lo que no prometes y varía a propósito lo accidental para que nadie dependa de ello.
Un consumidor falla al recibir un valor de enumerado nuevo	El consumidor hacía un tratamiento exhaustivo de los valores conocidos	Exige en el contrato que los valores desconocidos se toleren, desde la primera versión.
Hay que mantener v1 y v2 completas y nadie retira nada	La versión nueva se creó copiando la implementación	Una sola implementación con capa de traducción para lo antiguo, máximo dos versiones y fecha de retirada al publicar.
El reproceso de mensajes antiguos falla con el consumidor nuevo	Falta compatibilidad hacia delante en el esquema de eventos	Campos nuevos siempre opcionales con valor por defecto, sin reutilizar nombres ni cambiar tipos, y validación en el productor.
Una versión lleva años anunciada como retirada y sigue viva	Se avisó por correo y no se midió el uso por consumidor	Exige identidad por consumidor, degrada progresivamente, haz apagados de ensayo y apaga cuando el uso sea cero; después borra el código.

■ 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 la ley de Hyrum y qué decisión de diseño se deriva de ella?
2. ¿Por qué el cambio semántico es peor que el restrictivo?
3. ¿En qué consisten los tres pasos de expandir y contraer, y cuál se salta siempre?
4. ¿Por qué los esquemas de eventos necesitan compatibilidad hacia delante siempre?
5. ¿Qué hace falta antes de poder retirar una versión de verdad?

Referencias

- Wright, H. (2017). Hyrum's Law. <https://www.hyrumslaw.com/>
- Confluent (2025). Schema evolution and compatibility — reglas hacia atrás y hacia delante en eventos. <https://docs.confluent.io/platform/current/schema-registry/fundamentals/schema-evolution.html>
- Pact (2025). Consumer-driven contract testing. <https://docs.pact.io/>

- Google (2025). API Improvement Proposals: versioning and compatibility. <https://google.aip.dev/180>
- Ambler, S. y Sadalage, P. (2006). Refactoring Databases — expandir y contraer aplicado a esquemas. <https://www.oreilly.com/library/view/refactoring-databases-evolutionary/0321293533/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 15 en PDF · Manual integral

Anterior	Índice	Siguiente
← 187 · Consistencia, particiones, relojes y consenso	Parte 15 · Programa	189 · Modelado de amenazas y arquitectura de confianza cero →

Evaluación — 188 Contratos de API, eventos y compatibilidad evolutiva

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 contract-evolution 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.

189 — Modelado de amenazas y arquitectura de confianza cero

Parte: 15 — Arquitectura de sistemas e ingeniería de requisitos Nivel: intermedio-avanzado · Horas estimadas: 4 Laboratorio: security · Estado: EXECUTABLECORE

Propósito

Modelar amenazas de forma que produzca decisiones y no un documento, y traducir «confianza cero» de eslogan a arquitectura concreta. La clase da el método de modelado en cuatro preguntas, la clasificación que evita olvidar categorías enteras, y la parte que casi siempre falta: decidir qué amenazas se aceptan sin mitigar, por escrito y con nombre, porque un modelo que mitiga todo no se ha usado nunca.

Resultados de aprendizaje

Al finalizar podrás:

1. Modelar amenazas con las cuatro preguntas y un diagrama de flujo de datos.
2. Clasificar amenazas por categoría para no olvidar familias enteras.
3. Traducir confianza cero en decisiones de identidad, red y datos.
4. Priorizar por alcance y no por probabilidad estimada a ojo.
5. Aceptar amenazas por escrito, con nombre, riesgo y fecha de revisión.

Conceptos centrales

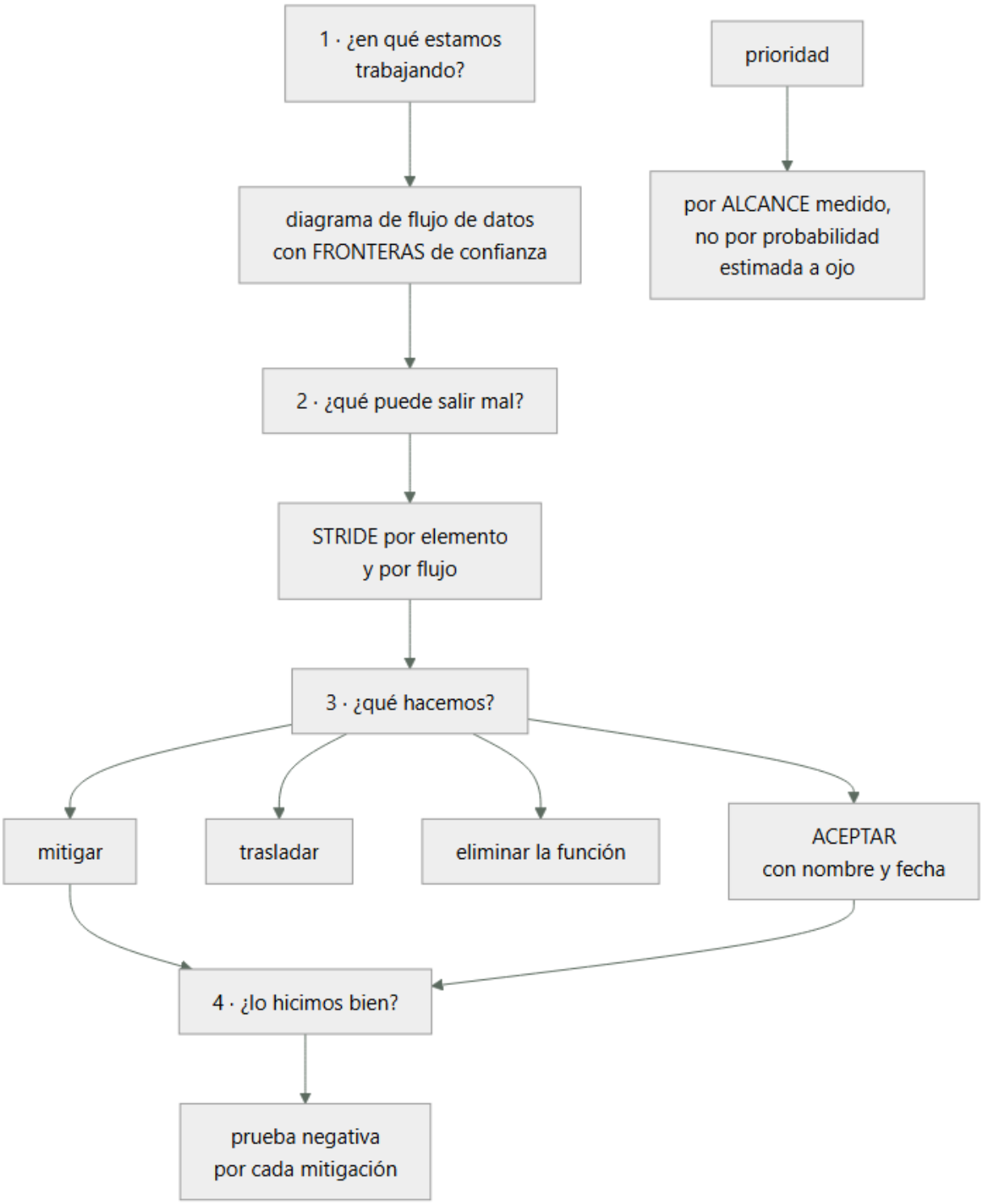
Concepto	Comprensión verificable
modelado de amenazas	Ejercicio estructurado para responder qué puede salir mal y qué se hace al respecto, antes de construir.
frontera de confianza	Línea que cruza un dato al pasar a un ámbito con distinto nivel de confianza. Es donde se validan cosas.
STRIDE	Clasificación de amenazas: suplantación, manipulación, repudio, revelación, denegación y elevación.
confianza cero	Ninguna petición se confía por su origen de red: se autentica, se autoriza y se registra siempre.
alcance	Lo que se puede tocar desde un punto comprometido. Es la medida que ordena la prioridad.
amenaza aceptada	La que se decide no mitigar, con motivo, quien la acepta y fecha de revisión.

Modelo mental

Una arquitectura es un conjunto de decisiones sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

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

■ Flujo de razonamiento



Desarrollo

1. Las cuatro preguntas

El modelado de amenazas se convierte en burocracia cuando se plantea como un formulario. Planteado como cuatro preguntas, cabe en una sesión de dos horas y produce decisiones.

```
1. ¿EN QUÉ ESTAMOS TRABAJANDO?
   un diagrama de flujo de datos, no de componentes
   → qué datos, por dónde viajan, dónde se guardan, quién
     los toca
   → y sobre todo: DÓNDE ESTÁN LAS FRONTERAS DE CONFIANZA
```

```
2. ¿QUÉ PUEDE SALIR MAL?
   por elemento y por flujo, con una clasificación que evite
   olvidar familias enteras

3. ¿QUÉ VAMOS A HACER?
   mitigar · trasladar · eliminar la función · ACEPTAR

4. ¿LO HICIMOS BIEN?
   una prueba negativa por cada mitigación         ley 22
```

Y la pregunta 1 merece detalle porque de ella depende todo:

```
LA FRONTERA DE CONFIANZA está donde cambia quién puede
hacer qué

  internet → borde                la más evidente
  borde → servicio interno
  servicio → base de datos
  cuenta de producción → cuenta de datos
  código propio → biblioteca de terceros
  persona → sistema (consola, acceso temporal)
  canalización → producción      ← la más olvidada  clase 108
  proveedor externo → nuestro sistema (webhook)  ← olvidada también
```

Y el hallazgo de la clase 182 vuelve aquí:

```
el webhook de la pasarela era un punto de ENTRADA externo
no dibujado
→ y por tanto no estaba en ningún modelo de amenazas
→ lo que no está en el diagrama no se analiza
```

Y una regla sobre cuándo hacerlo:

```
al diseñar, no al terminar
→ un modelo de amenazas después de construir produce una
  lista de cosas caras de arreglar

y se revisa cuando
  aparece un punto de entrada nuevo
  cambia quién puede escribir un dato
  se integra un tercero
  se cruza una frontera nueva de cuenta o de región
```

2. Qué puede salir mal, sin olvidar categorías

Preguntar «qué puede salir mal» sin estructura produce siempre las mismas tres respuestas. Una clasificación fuerza a recorrer todas las familias:

```
S  SUPLANTACIÓN        alguien se hace pasar por otro
   contramedida        autenticación fuerte, credenciales de
                       corta vida, identidad de carga    clase 134

T  MANIPULACIÓN        alguien altera datos o código
   contramedida        integridad, firma, permisos de escritura,
                       artefactos firmados               clase 106

R  REPUDIO             alguien niega haber hecho algo
   contramedida        registro de auditoría inalterable  clase 141

I  REVELACIÓN          alguien ve lo que no debe
   contramedida        cifrado, permisos, minimización de datos

D  DENEGACIÓN          alguien impide el servicio
   contramedida        límites, cuotas, aislamiento       clase 186

E  ELEVACIÓN           alguien obtiene más permisos
   contramedida        privilegio mínimo, separación de
                       ámbitos, sin permisos comodín      clase 134
```

Y el modo práctico de aplicarlo sin que dure una semana:

```
recorre los FLUJOS, no los componentes
por cada flujo que cruza una frontera, pregunta las seis
y apunta solo las que sean plausibles en ESTE sistema

→ un flujo interno entre dos módulos del mismo despliegue
```

```
  casi nunca merece las seis
→ un flujo que cruza de internet, siempre
```

Y las amenazas que este programa ha visto materializarse y que casi nunca aparecen en los modelos:

```
la canalización como vector                          clase 108
  un cambio en la definición del despliegue llega a producción
  sin revisión

la credencial de larga duración compartida           clase 179
  cuatro servicios usando la misma clave de tres años

el trabajo programado sin dueño                        ley 20
  con permisos amplios y sin nadie que lo mire

el acceso de emergencia                              clase 179
  que existe, nadie prueba, y a veces no funciona

el dato copiado a un entorno inferior
  la copia de producción en preproducción, sin ofuscar

el proveedor externo que llama de vuelta
  webhooks sin verificación de firma
```

Y una prioridad que ordena mejor que la probabilidad:

```
no preguntes «¿qué probabilidad tiene?» — nadie lo sabe
pregunta «¿HASTA DÓNDE SE LLEGA desde aquí?»

→ el alcance se mide, la probabilidad se inventa    clase 133
→ y reducir alcance protege contra amenazas que no se
  han imaginado
```

3. Confianza cero, en concreto

«Confianza cero» se vende como un producto y es una decisión de arquitectura. Su contenido real cabe en cinco reglas:

```
1. LA RED NO AUTORIZA
   estar dentro no da permiso
   → toda petición se autentica y se autoriza, también las
     internas
   → la red sigue importando, pero como contención, no como
     autorización                                    clase 135

2. IDENTIDAD DE CARGA, NO CREDENCIAL GUARDADA
   cada servicio tiene identidad propia, de corta vida
   → sin claves estáticas compartidas              clase 159

3. PERMISO MÍNIMO Y ESPECÍFICO
   por recurso y por acción; sin comodines
   → y revisado con permisos concedidos y no usados  clase 134

4. CUANTO OCURRE SE REGISTRA Y ES ATRIBUIBLE
   quién, qué, cuándo, desde dónde
   → y el registro va a una cuenta donde el comprometido no
     puede borrarlo                                  clase 141

5. VERIFICACIÓN CONTINUA
   el permiso no es permanente: caduca, se renueva y se revisa
   → acceso temporal con aprobación y caducidad
```

Y lo que confianza cero no es:

```
no es «quitar la red privada»
  → la segmentación sigue reduciendo alcance
no es un producto que se compra
no es aplicable de golpe a todo
  → se aplica primero donde el alcance es mayor
```

Y el orden práctico de adopción, por retorno:

```
1. eliminar credenciales de larga duración
2. identidad por carga y por consumidor            clase 188
3. autorización en cada salto, no solo en el borde
4. registro atribuible fuera del alcance del comprometido
```

```
5. segmentación por alcance, empezando por los datos
6. acceso humano temporal y aprobado
```

Y la comprobación que dice si está de verdad implantado:

```
toma la credencial de un servicio cualquiera y pregunta
  ¿a cuántos recursos llega?
  ¿cuánto dura?
  ¿queda registrado su uso en un sitio que ese servicio no
    puede tocar?
→ si la respuesta a la primera es «a casi todo», lo demás
  da igual
```

4. Lo que se acepta sin mitigar

Un modelo de amenazas donde todo está mitigado no se ha usado: se ha escrito para pasar una revisión.

```
LAS CUATRO RESPUESTAS POSIBLES
  MITIGAR      añadir un control
  TRASLADAR    contrato, seguro, proveedor
  ELIMINAR     quitar la función que crea la amenaza
               ← la más infravalorada
  ACEPTAR      no hacer nada, a propósito
```

Y «eliminar» merece atención porque resuelve más de lo que parece:

```
¿hace falta guardar este dato?          → si no, no se guarda
¿hace falta que este servicio escriba?  → si no, se le quita
¿hace falta exponer este endpoint?      → si no, se retira
→ el dato que no existe no se filtra                ley 20
```

La aceptación, escrita:

```
qué amenaza
qué pasaría si se materializa, en términos de negocio
por qué no se mitiga (coste, complejidad, no compensa)
quién la acepta, con nombre y cargo
en qué fecha se revisa
qué señal la convertiría en prioritaria
```

Y la última línea es la que hace útil la aceptación:

```
«se acepta mientras el negocio de empresa sea < 15 % de los
 ingresos»
→ convierte una decisión estática en una que se revisa sola
```

Comprobar que las mitigaciones funcionan, que es la pregunta 4 y la que más se salta:

```
por cada mitigación, una prueba negativa
  ¿se puede llegar a producción desde un entorno inferior?
  ¿se puede sacar un dato a un destino no declarado?
  ¿se puede desactivar el registro?
  ¿se puede usar una credencial fuera de su ámbito?
  ¿se detecta la técnica que decimos detectar?    clase 174

y la evidencia del programa: en la clase 179, de 10 pruebas
de seguridad, 3 fallaron; todas correspondían a controles
documentados como implantados                       ley 22
```

Y la lista de comprobación de la clase:

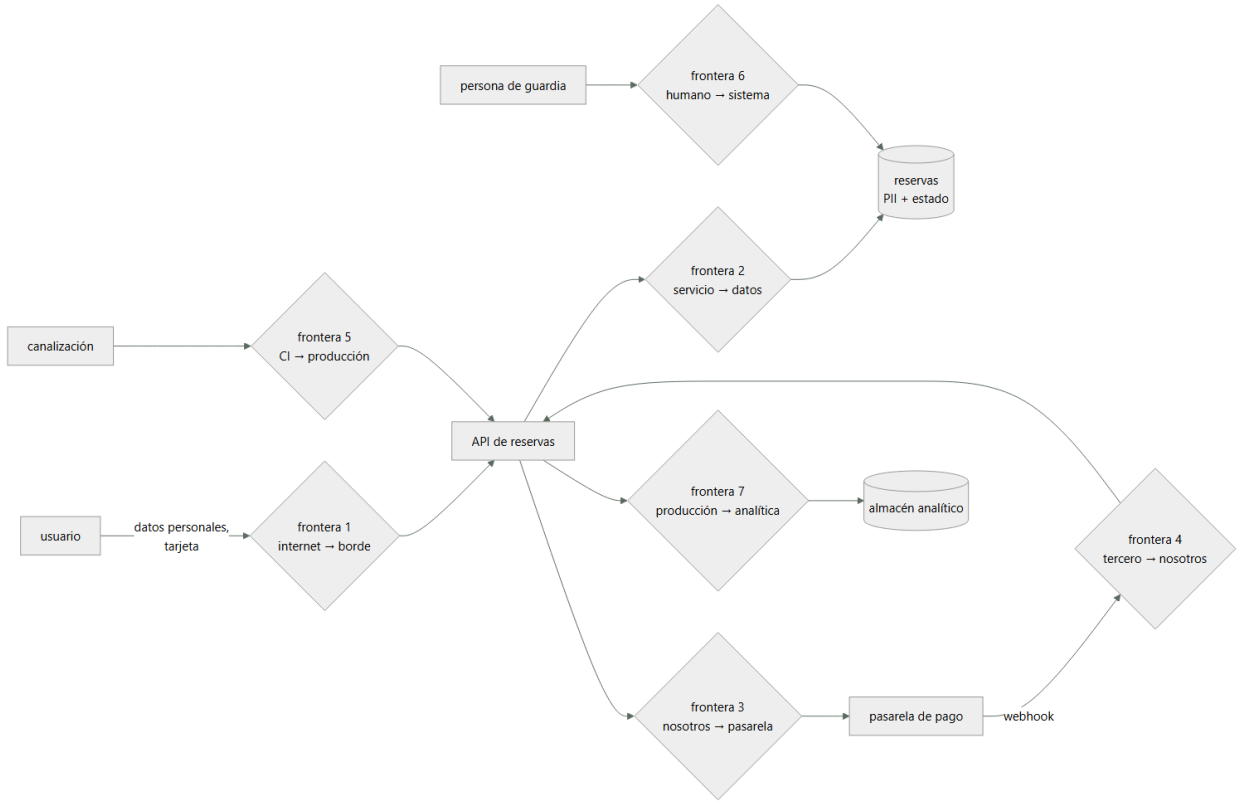
- ■ hay diagrama de flujo de datos, no de componentes
- ■ las fronteras de confianza están dibujadas
- ■ están incluidas la canalización y los webhooks externos
- ■ cada flujo que cruza frontera se ha recorrido con las seis categorías
- ■ la prioridad se fijó por alcance medido
- ■ cada amenaza tiene una de las cuatro respuestas
- ■ se consideró eliminar la función, no solo mitigar
- ■ las aceptadas tienen nombre, motivo, fecha y señal de revisión
- ■ cada mitigación tiene una prueba negativa
- ■ las pruebas se han ejecutado, no razonado
- ■ ninguna credencial de servicio llega «a casi todo»

Y el cierre que enlaza con la clase siguiente: todas las decisiones de esta parte —fronteras, consistencia, contratos, amenazas aceptadas— necesitan quedar escritas de forma que se puedan revisar y hacer cumplir. Registrarlas y comprobarlas automáticamente es la materia de la clase 190.

Ejemplo trabajado

El equipo de reservas hace el modelo de amenazas antes de construir. Lo que sigue es la sesión de dos horas: el diagrama con fronteras, las diecinueve amenazas encontradas, las cinco que se aceptaron por escrito y las tres pruebas negativas que fallaron.

El diagrama de flujo de datos, con fronteras:



Y la observación de la sesión:

```
las fronteras 4, 5 y 7 no estaban en el diagrama de
arquitectura original
→ y de las 19 amenazas encontradas, 8 estaban en esas tres
```

Las diecinueve amenazas, por frontera y categoría.

```
FRONTERA 1 · internet → borde
  S1  robo de sesión por testigo sin caducidad     MITIGAR
  D1  saturación de búsqueda desde una IP          MITIGAR
  I1  enumeración de reservas por identificador     MITIGAR
      secuencial
  T1  manipulación de precio en la petición         MITIGAR

FRONTERA 2 · servicio → datos
  E1  la credencial de la API permite leer TODAS    MITIGAR
      las tablas, incluidas las de consentimientos
  I2  copia de producción en preproducción sin      MITIGAR
      ofuscar
  R1  no se puede saber qué servicio hizo un        MITIGAR
      cambio concreto

FRONTERA 3 · nosotros → pasarela
  I3  datos de tarjeta en nuestros registros        ELIMINAR
  D2  la pasarela responde lento y agota hilos      MITIGAR

FRONTERA 4 · pasarela → nosotros (webhook)   ← no estaba
  S2  cualquiera puede llamar al webhook            MITIGAR
      haciéndose pasar por la pasarela
  T2  reproducción de un webhook antiguo            MITIGAR
  D3  ausencia de webhooks sin detección            MITIGAR
```

```
FRONTERA 5 · canalización → producción       ← no estaba
  E2  quien modifica la definición del despliegue   MITIGAR
      cambia producción sin revisión
  T3  artefacto sustituido entre construcción y     MITIGAR
      despliegue
  S3  credencial de despliegue de larga duración    MITIGAR

FRONTERA 6 · humano → sistema
  E3  acceso permanente de guardia a la base        MITIGAR
  R2  consulta directa sin registro atribuible      MITIGAR

FRONTERA 7 · producción → analítica          ← no estaba
  I4  el almacén analítico contiene PII completa    MITIGAR
      y lo consultan 40 personas
  I5  exportación a hoja de cálculo sin control      ACEPTAR
```

La prioridad, por alcance medido:

```
amenaza  alcance desde el punto comprometido
E1       11 tablas, incluidas consentimientos y pagos   ← máximo
E2       cualquier cambio en producción                 ← máximo
I4       PII de 2,3 M de clientes                        ← máximo
S3       despliegue en 4 servicios
E3       lectura y escritura en toda la base
S2       creación de reservas confirmadas falsas
...
D1       degradación de búsqueda                         ← bajo
```

Y la decisión de orden que salió de ahí:

```
se empieza por E1, E2 e I4, no por las más «probables»
motivo   el alcance se mide; la probabilidad se inventa
```

Las mitigaciones más significativas:

```
E1  la API pasa a tener 3 identidades, una por módulo
    reservas   → tablas de reservas e inventario
    contacto   → esquema de contacto
    consent.   → solo lectura, y escritura solo desde legal
    alcance    de 11 tablas a 4                      clase 183

E2  la definición del despliegue exige 2 revisiones y firma
    la identidad de la canalización no puede modificar
      políticas ni identidades
    despliegue solo desde artefacto firmado          clase 106

I4  el almacén analítico deja de recibir PII completa
    identificador seudonimizado, sin nombre ni correo
    la reidentificación exige una petición con aprobación
    consultantes con acceso a PII    de 40 a 3

I3  ELIMINAR: nunca se reciben datos de tarjeta
    el formulario de pago es de la pasarela; nosotros
    recibimos un testigo
    → la amenaza desaparece, no se mitiga

S2  verificación de firma del webhook + marca de tiempo
    con ventana de 5 min
T2  identificador de suceso único con registro de vistos
D3  alerta de ausencia de webhooks                    ley 13
```

Las cinco amenazas aceptadas, por escrito:

```
I5  exportación a hoja de cálculo desde el panel interno
    si ocurre   un empleado se lleva datos agregados sin PII
    por qué no  el control de exportación cuesta 3 semanas y
                bloquea el trabajo de negocio
    acepta      dirección de datos, María Alonso
    revisa      en 6 meses
    señal       si el panel llega a mostrar PII, deja de
                aceptarse

D1  saturación de búsqueda desde muchas IP distintas
    mitigado parcialmente con límite por IP; el ataque
    distribuido no
```

```
    por qué no  el servicio de protección cuesta 900 €/mes;
                la búsqueda degradada no impide reservar
    acepta      dirección de tecnología
    revisa      si hay un incidente real

R3  los registros de la aplicación se conservan 90 días,
    no 365
    por qué no  coste de almacenamiento
    acepta      dirección de tecnología
    señal       si aparece requisito de auditoría, cambia

S4  el socio C sigue usando una credencial de larga duración
    por qué no  su plataforma no soporta rotación automática
    acepta      dirección de alianzas, con nombre
    revisa      trimestral; alcance limitado a 2 endpoints
    señal       si su volumen supera el 10 % del tráfico

T4  no se firma el contenido de los eventos internos
    por qué no  viajan por un canal privado y el coste de
                verificación en cada consumidor no compensa
    acepta      arquitectura
    señal       si un consumidor externo llega a leerlos
```

Y la observación sobre esta lista:

```
cinco amenazas aceptadas de diecinueve
→ un modelo con cero aceptadas habría sido señal de que
  nadie lo miró con intención de decidir
```

Las pruebas negativas: 14 ejecutadas, 3 fallaron.

```
✓  llegar a producción desde preproducción            bloqueado
✓  desplegar artefacto sin firmar                     rechazado
✓  usar credencial de reservas sobre consentimientos  denegado
✓  desactivar el registro de auditoría                rechazado
✗  llamar al webhook sin firma válida
   → la verificación estaba implementada y DESACTIVADA por
     una variable de entorno puesta en pruebas y nunca
     revertida
✓  reproducir un webhook antiguo                      rechazado
✓  enumerar reservas por identificador                identificadores
                                                      opacos
✗  sacar datos a un destino externo no declarado
   → no había control de salida en la subred nueva  clase 135
✓  acceso temporal caduca en 60 min                   sí
✓  usar acceso de emergencia                          corregido
                                                      desde 179
✗  consultar la base como persona sin registro atribuible
   → el acceso por el túnel administrativo no registraba
     la identidad de la persona, solo la del túnel
✓  simular exfiltración de PII → detectada             9 min
✓  identidad de canalización modificando una política rechazado
✓  copia de producción en preproducción               ofuscada
```

Y lo que enseñaron los tres fallos:

```
los tres controles estaban DOCUMENTADOS como implantados
dos de los tres estaban implementados y desactivados
y el tercero se implantó en una subred y no en la nueva
→ ninguno se habría detectado revisando la documentación
                                                       ley 22
```

La lección que esta clase deja: de las diecinueve amenazas, ocho estaban en las tres fronteras que no aparecían en el diagrama de arquitectura —el webhook del proveedor, la canalización y el flujo hacia analítica—. La amenaza más grave se resolvió eliminando la función: al no recibir nunca datos de tarjeta, la categoría entera desapareció. Y de catorce pruebas, tres fallaron sobre controles que constaban como implantados.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-15-systems-architecture-engineering/189-modelado-de-amenazas-y-arquitectura-de-confianza-cero/lab.py
```

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

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

Evidencia esperada

El artefacto principal es un control con amenaza, mitigación y evidencia. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye system-threat-model 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 modelo de amenazas no encuentra nada relevante	Se dibujaron componentes en vez de flujos de datos y no se marcaron las fronteras de confianza	Haz un diagrama de flujo de datos, marca cada frontera e incluye canalización, webhooks de terceros y salidas hacia analítica.
Se priorizan amenazas poco importantes	Se ordenó por probabilidad estimada a ojo	Ordena por alcance medido desde cada punto comprometido; reducir alcance protege también de lo que no se imaginó.
El documento de amenazas no tiene ninguna aceptada	Se escribió para pasar una revisión, no para decidir	Acepta explícitamente lo que no compensa mitigar, con motivo, nombre, fecha y señal que lo reabra.
Un control documentado como implantado no funciona	Nunca se probó; a veces está implementado y desactivado por configuración	Una prueba negativa por mitigación, ejecutada en el entorno real y repetida periódicamente.
Confianza cero se quedó en comprar un producto	Se confundió el eslogan con las decisiones de identidad, permiso y registro	Aplica las cinco reglas por orden de retorno y comprueba a cuántos recursos llega la credencial de un servicio cualquiera.
Una categoría entera de riesgo no aparece en el análisis	Se preguntó «qué puede salir mal» sin clasificación	Recorre las seis categorías por cada flujo que cruce una frontera de confianza.

■ 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 cuatro preguntas del modelado de amenazas y cuál se salta más?
2. ¿Qué fronteras de confianza suelen faltar en los diagramas?
3. ¿Por qué se prioriza por alcance y no por probabilidad?
4. ¿Qué cinco decisiones concretas contiene «confianza cero»?
5. ¿Qué debe incluir una amenaza aceptada para ser útil?

Referencias

• Shostack, A. (2014). Threat Modeling: Designing for Security. <https://shostack.org/books/threat-modeling-book>

• Threat Modeling Manifesto (2020) — las cuatro preguntas. <https://www.threatmodelingmanifesto.org/>

• NIST SP 800-207 (2020). Zero Trust Architecture. <https://csrc.nist.gov/pubs/sp/800/207/final>

• Microsoft (2025). STRIDE threat model. <https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats>

• Google (2024). BeyondProd — confianza cero aplicada a cargas y canalizaciones. <https://cloud.google.com/docs/security/beyondprod>

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

Descargar: Parte 15 en PDF · Manual integral

Anterior	Índice	Siguiente
← 188 · Contratos de API, eventos y compatibilidad evolutiva	Parte 15 · Programa	190 · ADRs, fitness functions y gobierno de decisiones →

Evaluación — 189 Modelado de amenazas y arquitectura de confianza cero

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 system-threat-model 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.

190 — ADRs, fitness functions y gobierno de decisiones

Parte: 15 — Arquitectura de sistemas e ingeniería de requisitos Nivel: intermedio-avanzado · Horas estimadas: 4
Laboratorio: architecture · Estado: EXECUTABLECORE

Propósito

Dejar por escrito las decisiones de arquitectura de forma que sirvan años después, y comprobarlas automáticamente para que no se erosionen. La clase da el formato de registro de decisión que funciona —premisas y alternativas descartadas, no justificaciones—, define las funciones de aptitud como pruebas ejecutables de propiedades arquitectónicas, y aborda de frente el motivo por el que ambas cosas suelen fracasar: si estorban más de lo que ayudan, se rodean.

Resultados de aprendizaje

Al finalizar podrás:

1. Escribir un registro de decisión con premisas, alternativas y coste de cambio.
2. Distinguir qué decisiones merecen registro y cuáles no.
3. Implementar funciones de aptitud que fallen la canalización.
4. Elegir qué propiedades se comprueban automáticamente y cuáles no.
5. Evitar que el registro y las comprobaciones se conviertan en trámite.

Conceptos centrales

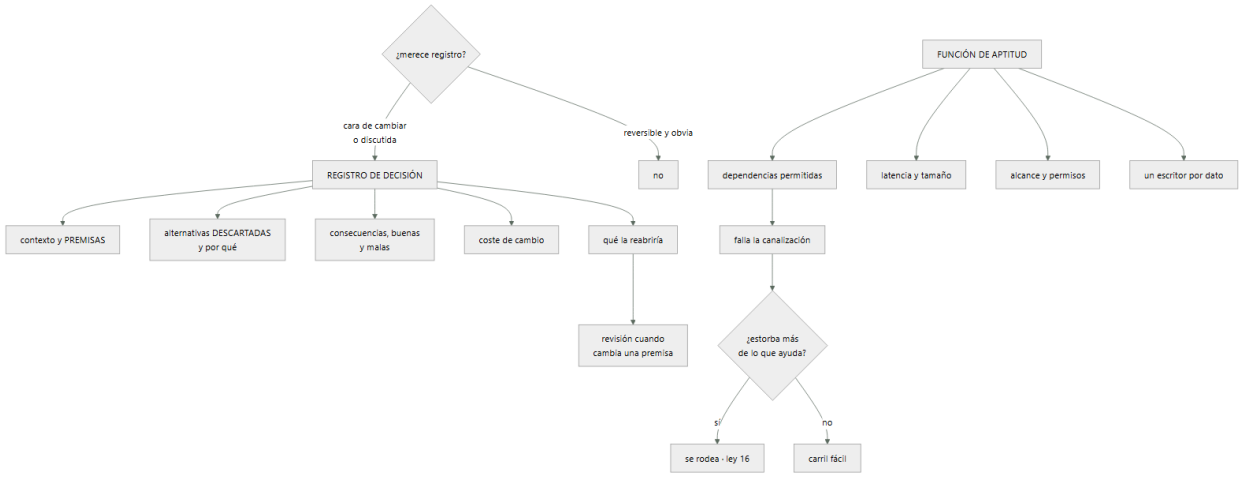
Concepto	Comprensión verificable
registro de decisión (ADR)	Documento corto que fija una decisión con su contexto, alternativas descartadas y consecuencias.
premisa	Lo que se creía cierto al decidir. Cuando cambia, la decisión se revisa; sin premisas escritas, no hay forma de saberlo.
función de aptitud	Comprobación ejecutable de una propiedad arquitectónica: dependencias, latencia, alcance, cobertura.
erosión arquitectónica	Deriva entre la arquitectura decidida y la construida. Ocurre por acumulación de excepciones pequeñas.
decisión superada	La que fue correcta y dejó de serlo. Se marca como tal, no se borra.
gobierno útil	El que ayuda a decidir más rápido. El que solo pide papeles se rodea.
carril fácil	Camino por defecto que cumple lo decidido sin esfuerzo. Es lo que hace innecesario vigilar.

Modelo mental

Una arquitectura es un conjunto de decisiones sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

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 registro de decisión que sirve

Un registro de decisión útil se lee dentro de tres años, por alguien que no estaba, y le permite entender si la decisión sigue siendo correcta. Eso exige una forma concreta.

```
TÍTULO      una frase que diga la decisión, no el tema
            «Un escritor por dato en el módulo de precios»
            no «Arquitectura de precios»

ESTADO      propuesta · aceptada · superada por [otra]

CONTEXTO Y PREMISAS
            qué se sabía, con cifras
            «el catálogo cambia 1 vez/mes y los precios 9»
            ← esto es lo que hará revisable la decisión

DECISIÓN    en imperativo, sin adornos

ALTERNATIVAS DESCARTADAS
            cada una, con el motivo del descarte
            ← la parte más valiosa y la que más se omite

CONSECUENCIAS
            lo bueno Y lo malo que se acepta

COSTE DE CAMBIO
            qué costaría revertirla dentro de un año  clase 181

QUÉ LA REABRIRÍA
            la señal concreta que obliga a revisarla
```

Y los dos campos que separan un registro útil de uno decorativo:

```
PREMISAS
  sin ellas no se puede saber si la decisión caducó
  «decidimos X porque el tráfico era 900/s» permite ver que
  a 5.000/s hay que rediscutirlo

ALTERNATIVAS DESCARTADAS
  evitan que alguien vuelva a proponer lo mismo dentro de
  dos años y se pierda un mes
  y si la alternativa se descartó por una premisa que ya no
  se cumple, se recupera sola
```

Qué merece registro y qué no:

```
SÍ
  lo caro de cambiar                          clase 181
  lo que se discutió y hubo desacuerdo
  lo que sorprendería a alguien de fuera
  lo que se decidió NO hacer
  las amenazas aceptadas                      clase 189
```

```
  las cesiones entre atributos de calidad

NO
  lo reversible y obvio
  lo que el código ya dice
  lo que solo tiene una opción razonable
```

Y el error de proceso que arruina el registro:

```
escribirlo DESPUÉS de decidir, para cumplir
→ sale una justificación, no un registro
→ y las alternativas descartadas se inventan

escribirlo ANTES, como borrador, y usarlo para decidir
→ el propio ejercicio de escribir las alternativas cambia
  decisiones
```

Y una decisión que casi nunca se registra y siempre hace falta:

```
la que se toma y resulta EQUIVOCADA
  se marca como superada, se enlaza la nueva y se escribe
  qué premisa falló
→ borrarla pierde la información más útil del archivo
```

2. Funciones de aptitud

Una decisión escrita se erosiona: nadie la incumple de golpe, se incumple una excepción cada vez. Las funciones de aptitud la convierten en algo que falla solo.

```
QUÉ ES
  una comprobación ejecutable de una propiedad arquitectónica
  que corre en la canalización y falla el cambio

QUÉ NO ES
  una prueba unitaria (que comprueba comportamiento)
  un panel (que informa y nadie mira)              ley 15
```

Las que más valor dan, agrupadas por lo que protegen:

```
FRONTERAS Y DEPENDENCIAS                          clase 183
  ningún módulo importa el interior de otro
  ninguna consulta cruza esquemas
  el grafo de dependencias no tiene ciclos
  ningún módulo depende de más de N módulos

DATOS                                               ley 21
  cada tabla tiene exactamente un escritor
  ningún esquema nuevo sin dueño declarado
  ningún campo de PII sin clasificación

RENDIMIENTO                                       clase 186
  el p99 del flujo crítico no supera X en la prueba
  ninguna operación hace más de N llamadas de red
  ninguna consulta sin índice sobre tablas grandes

SEGURIDAD                                         clase 189
  ninguna identidad con permisos comodín
  ningún almacén público
  ninguna credencial de larga duración
  ningún recurso sin etiqueta de dueño

OPERACIÓN
  todo servicio expone las cuatro señales         clase 121
  toda alerta tiene procedimiento enlazado        clase 125
  todo servicio nuevo tiene objetivo declarado

CONTRATOS                                         clase 188
  ningún cambio rompe las expectativas declaradas
  todo esquema de evento es compatible hacia delante
```

Y las tres propiedades de una función de aptitud que funciona:

```
1. FALLA, no informa
   un panel en verde no impide nada

2. ES RÁPIDA
```

```
   si tarda 20 minutos, alguien la desactivará

3. TIENE SALIDA DECLARADA
   una excepción con motivo, dueño y fecha
   → sin salida, la gente rodea la comprobación   ley 16
   → con salida registrada, las excepciones se pueden contar
```

Y la métrica que dice si el sistema de comprobaciones está sano:

```
número de excepciones vivas, y su antigüedad
  → si crece siempre, la regla está mal calibrada
  → si hay excepciones de más de un año, o se arreglan o
    la regla se retira                          clase 170
```

3. Por qué esto fracasa, y cómo evitarlo

Registros de decisión y funciones de aptitud fracasan por el mismo mecanismo que los controles de la parte 14, y conviene decirlo antes de implantarlos:

```
si cuesta más rellenar el registro que tomar la decisión,
se rellena después y mal

si la comprobación bloquea sin explicar cómo cumplirla,
se busca la forma de saltársela

si el órgano que revisa tarda dos semanas, la gente decide
y pide perdón                                    ley 16
```

Lo que hace que funcione, en cinco puntos:

```
1. EL CARRIL FÁCIL CUMPLE SOLO                  clase 171
   la plantilla de servicio nuevo ya trae señales, objetivo,
   etiquetas, identidad mínima
   → así la comprobación casi nunca falla, y cuando falla
     señala algo real

2. LA COMPROBACIÓN DICE CÓMO ARREGLARLO
   mensaje con el motivo, el enlace a la decisión y el
   comando o el cambio concreto

3. REGISTRO CORTO Y EN EL REPOSITORIO
   una página, junto al código, en el mismo cambio
   → no en una herramienta aparte que nadie abre

4. LA REVISIÓN ES SÍNCRONA Y CORTA
   30 minutos con las personas que deciden, no un comité
   que se reúne cada quince días                clase 191

5. SE RETIRAN REGLAS
   una regla que genera excepciones constantes está mal
   → retirarla es una decisión legítima, y se registra
```

Y una comprobación honesta que conviene hacer cada seis meses:

```
¿cuántos registros se han leído en el último año?
  → si la respuesta es «ninguno», el formato o el sitio
    están mal
  → los registros útiles se leen cuando alguien propone algo
    parecido, y eso ocurre

¿cuántas veces una función de aptitud paró algo que era
un problema real?
  → si nunca, o la regla sobra o el carril ya lo impide
```

Y la relación con lo aprendido en la parte 14:

```
un gobierno de decisiones es un control
y un control que estorba se rodea
→ la diferencia entre gobierno útil e inútil no está en las
  reglas, está en si el camino que las cumple es el más fácil
```

4. Mantener vivo el archivo

Un archivo de decisiones crece y, sin cuidado, se vuelve inutilizable por acumulación.

```
ORGANIZACIÓN QUE FUNCIONA
```

```
  numeración correlativa, sin carpetas por tema
  índice con título y estado, generado automáticamente
  enlaces entre decisiones relacionadas y superadas
  búsqueda por texto: es como se usa de verdad

LO QUE NO FUNCIONA
  reorganizar por categorías
  borrar las superadas
  editar una decisión pasada en vez de superarla
```

Y el ciclo de revisión, que es lo que evita el archivo muerto:

```
se revisa una decisión cuando
  cambia una de sus premisas              ← el disparador real
  alguien propone lo que ya se descartó
  una función de aptitud falla repetidamente
  llega la fecha escrita en «qué la reabriría»

no se revisa «periódicamente todas»
→ eso produce reuniones sin objeto
```

Y una práctica que este programa ha usado en todas sus partes:

```
escribir lo que se espera que ocurra, y volver a corregirlo
con evidencia
→ aplicado a decisiones: anotar qué se espera que pase tras
  la decisión, y a los seis meses comparar
→ es la única forma de mejorar el criterio, y casi nadie
  lo hace
```

Y la lista de comprobación de la clase:

- cada decisión cara de cambiar tiene registro
- el registro incluye premisas con cifras
- incluye alternativas descartadas con su motivo
- incluye consecuencias malas, no solo buenas
- incluye coste de cambio y qué la reabriría
- se escribió antes de decidir, no después
- las decisiones erróneas están marcadas como superadas
- las propiedades importantes tienen función de aptitud
- las comprobaciones fallan, no informan
- cada comprobación dice cómo cumplirla
- hay salida declarada con dueño y fecha
- se cuentan las excepciones vivas y su antigüedad
- el carril por defecto cumple sin esfuerzo
- se han retirado reglas que no aportaban

Y el cierre que enlaza con la clase siguiente: las decisiones registradas hay que defenderlas ante quien las paga, quien las opera y quien las sufre, y cada uno pregunta cosas distintas. Comunicar arquitectura a esas audiencias es la materia de la clase 191.

Ejemplo trabajado

El equipo de reservas implanta registro de decisiones y funciones de aptitud. Lo que sigue son dos registros reales —uno de ellos superado a los ocho meses—, las once funciones que montaron, y el motivo por el que tres se retiraron.

Registro 007 · Un escritor por dato en precios

```
ESTADO      aceptada · 2024-03-11

CONTEXTO Y PREMISAS
  los precios se guardan en la tabla de catálogo
  precios cambia 9 veces al mes; catálogo, 1
  el 64 % de los cambios de catálogo son en realidad
    cambios de precio
  3 incidentes de «precio incorrecto» sin causa conocida
  volumen: 4,1 M de registros de catálogo

DECISIÓN
  el precio sale de la tabla de catálogo a un almacén propio
  del servicio de precios, que es su único escritor
  catálogo recibe los cambios por evento

ALTERNATIVAS DESCARTADAS
```

```
  1 dejarlo donde está y añadir validación
    → no resuelve el acoplamiento; lo hace más lento de
      descubrir
  2 vista materializada en catálogo
    → sigue habiendo dos escritores durante la transición,
      y la transición no termina nunca
  3 sacar precios a servicio pero compartiendo la tabla
    → monolito distribuido                          clase 184

CONSECUENCIAS
  buenas   precios se despliega sin bloquear a catálogo;
           desaparece el acoplamiento por datos
  malas    una llamada más en el flujo de reserva (+1,4 ms);
           consistencia eventual del precio en catálogo
           (≤ 10 min, aceptado por revenue)

COSTE DE CAMBIO
  alto. Volver atrás exige migrar el dato de nuevo: ~4 semanas

QUÉ LA REABRIRÍA
  si los cambios de precio bajan por debajo de 2/mes
  si el retraso de propagación resulta inaceptable para
    revenue
```

Y lo que este registro evitó nueve meses después:

```
un ingeniero nuevo propuso «simplificar» volviendo a guardar
el precio en catálogo
→ el registro 007 estaba enlazado desde el módulo
→ la discusión duró 10 minutos en vez de dos semanas
```

Registro 011 · Réplica de lectura para el panel de ocupación

```
ESTADO      SUPERADA por 019 · 2024-11-04

CONTEXTO Y PREMISAS (marzo 2024)
  el panel de ocupación hace 40 consultas pesadas/min
  la base primaria está al 71 % de utilización
  el panel lo usan 12 personas de operaciones
  presupuesto disponible: sí

DECISIÓN
  añadir una réplica de lectura dedicada al panel

ALTERNATIVAS DESCARTADAS
  1 caché de 60 s → «los datos deben ser exactos»  ← premisa
  2 vista materializada → complejidad de refresco

CONSECUENCIAS
  +410 €/mes; retraso de réplica de 200-900 ms
```

Y por qué se superó:

```
QUÉ FALLÓ (registrado en el 019)
  la premisa «los datos deben ser exactos» nunca se comprobó
  con nadie
  al preguntar a las 12 personas en octubre: ninguna
  necesitaba exactitud al segundo; la decisión de negocio
  que tomaban con ese panel era de turno, no de minuto

DECISIÓN NUEVA (019)
  caché de 30 s con la antigüedad mostrada en pantalla

EFECTO
  réplica retirada          -410 €/mes
  carga en el primario      -18 %
  quejas por el panel       de 9/mes a 0            clase 187

LECCIÓN REGISTRADA
  la premisa venía de una suposición del equipo técnico,
  no de una pregunta a los usuarios
  → desde entonces, toda premisa sobre lo que «necesita»
    alguien lleva el nombre de quien lo dijo
```

Las once funciones de aptitud, y qué encontró cada una.

```
regla                                    fallos    resultado
═══════════════════════════════════════════════════════════════════════
1  ninguna consulta cruza esquemas         17     todas reales
   → 17 consultas cruzadas acumuladas en 2 años   clase 183

2  cada tabla tiene un solo escritor        4     todas reales
   → incluida una que nadie conocía

3  sin permisos comodín en identidades      9     8 reales

4  sin credenciales de larga duración       6     todas reales

5  todo recurso con etiqueta de dueño      23     todas reales

6  todo servicio expone las 4 señales       3     reales

7  toda alerta con procedimiento enlazado  11     reales

8  ninguna operación > 6 llamadas de red    2     1 real,
                                                  1 excepción

9  el p99 del flujo crítico < 500 ms        0     nunca falló
   → RETIRADA: la prueba de carga ya lo cubre y esta
     duplicaba 9 min de canalización

10 ningún fichero de más de 800 líneas     41     0 reales
   → RETIRADA: medía tamaño, no acoplamiento; 41 excepciones
     en 3 meses                                     ley 17

11 el grafo de módulos sin ciclos           1     real
```

Y las tres reglas retiradas o corregidas:

```
RETIRADA 9    nunca falló y costaba 9 min por cambio
RETIRADA 10   41 excepciones y ningún problema real detectado
              → la regla medía lo fácil de medir      ley 17
CORREGIDA 3   generaba 1 falso positivo de cada 9 por un
              patrón legítimo; se ajustó el patrón
```

El dato que decidió si el sistema estaba sano, a los seis meses:

```
excepciones vivas                                  14
  con dueño y fecha                                14
  vencidas y sin renovar                            2  → arregladas
  de más de 6 meses                                 0

registros de decisión escritos                     23
registros consultados al menos una vez             17
discusiones cerradas citando un registro            6
decisiones superadas                                3

funciones que pararon un problema real              9 de 11
tiempo añadido a la canalización                 2,4 min
```

Y la comprobación que evitó que esto se volviera trámite:

```
el carril fácil se ajustó tres veces
  la plantilla de servicio nuevo pasó a traer etiquetas,
  señales, objetivo y alerta con procedimiento
  → los fallos de las reglas 5, 6 y 7 en servicios NUEVOS
    bajaron a cero
  → los que siguen fallando son servicios antiguos, que es
    exactamente lo que se quiere ver
```

La lección que esta clase deja: de las once funciones de aptitud, dos se retiraron y una de ellas —el límite de tamaño de fichero— había generado cuarenta y una excepciones sin detectar un solo problema real, que es el ejemplo perfecto de optimizar la medida en lugar del objetivo. Y del archivo de decisiones, la entrada más valiosa fue la que se superó: no porque la decisión fuera mala, sino porque dejó escrito que su premisa central nunca se había comprobado con nadie.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-15-systems-architecture-engineering/190-adrs-fitness-functions-y-gobierno-de-decisiones/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 adr-library 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 adr-library para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

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

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
Los registros de decisión no ayudan a nadie tres años después	No incluyen premisas, así que no se puede saber si la decisión caducó	Escribe el contexto con cifras concretas y añade qué señal reabriría la decisión.
Se vuelve a proponer una opción que ya se había descartado	No se registraron las alternativas ni el motivo del descarte	Incluye siempre las alternativas descartadas con su razón; si el motivo era una premisa que ya no se cumple, la opción vuelve legítimamente.
El registro suena a justificación	Se escribió después de decidir, para cumplir	Redáctalo como borrador antes de decidir y úsalo para decidir; el propio ejercicio cambia decisiones.
La arquitectura decidida se erosiona sin que nadie la incumpla de golpe	No hay comprobación automática de las propiedades importantes	Convierte las decisiones estructurales en funciones de aptitud que fallen la canalización.
Una comprobación acumula decenas de excepciones y no encuentra problemas	Mide algo fácil de medir en lugar de la propiedad que importa	Cuenta excepciones vivas y su antigüedad; retira la regla si no detecta problemas reales.
La gente rodea el gobierno de decisiones	Cuesta más que decidir, bloquea sin explicar y la revisión tarda semanas	Haz que el carril por defecto cumpla solo, que el mensaje diga cómo arreglarlo, y revisa en sesiones cortas y síncronas.

■ 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 dos campos separan un registro de decisión útil de uno decorativo?
2. ¿Por qué conviene escribir el registro antes de decidir?
3. ¿Qué tres propiedades tiene una función de aptitud que funciona?
4. ¿Qué indica que una regla está mal calibrada?
5. ¿Qué hace innecesario vigilar el cumplimiento de una decisión?

Referencias

- Nygard, M. (2011). Documenting architecture decisions. <https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions>
- Ford, N., Parsons, R. y Kua, P. (2017). Building Evolutionary Architectures — funciones de aptitud. <https://www.oreilly.com/library/view/building-evolutionary-architectures/9781491986356/>
- ThoughtWorks (2025). Technology Radar: lightweight ADRs. <https://www.thoughtworks.com/radar/techniques/lightweight-architecture-decision-records>
- ArchUnit (2025). Testing architecture rules in the build. <https://www.archunit.org/>
- Open Policy Agent (2025). Policy as code — comprobaciones ejecutables en la canalización. <https://www.openpolicyagent.org/docs/latest/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 15 en PDF · Manual integral

Anterior	Índice	Siguiente
← 189 · Modelado de amenazas y arquitectura de confianza cero	Parte 15 · Programa	191 · Architecture review y comunicación con stakeholders →

Evaluación — 190 ADRs, fitness functions y gobierno de decisiones

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-library 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.

191 — Architecture review y comunicación con stakeholders

Parte: 15 — Arquitectura de sistemas e ingeniería de requisitos Nivel: intermedio-avanzado · Horas estimadas: 4
Laboratorio: governance · Estado: EXECUTABLECORE

Propósito

Someter una arquitectura a revisión de forma que encuentre problemas, y comunicarla a audiencias que preguntan cosas distintas. La clase da el método de revisión basado en escenarios —que encuentra riesgos donde la revisión por opinión no encuentra nada—, la forma de traducir decisiones técnicas a las tres preguntas que hace cada audiencia, y la disciplina que hace útil una revisión: buscar riesgos y puntos sensibles, no aprobar o rechazar.

Resultados de aprendizaje

Al finalizar podrás:

1. Ejecutar una revisión de arquitectura basada en escenarios de calidad.
2. Distinguir riesgo, punto sensible y cesión, y registrarlos.
3. Traducir una decisión técnica a lo que cada audiencia necesita saber.
4. Presentar una arquitectura con la estructura que resiste preguntas.
5. Evitar que la revisión se convierta en un trámite de aprobación.

Conceptos centrales

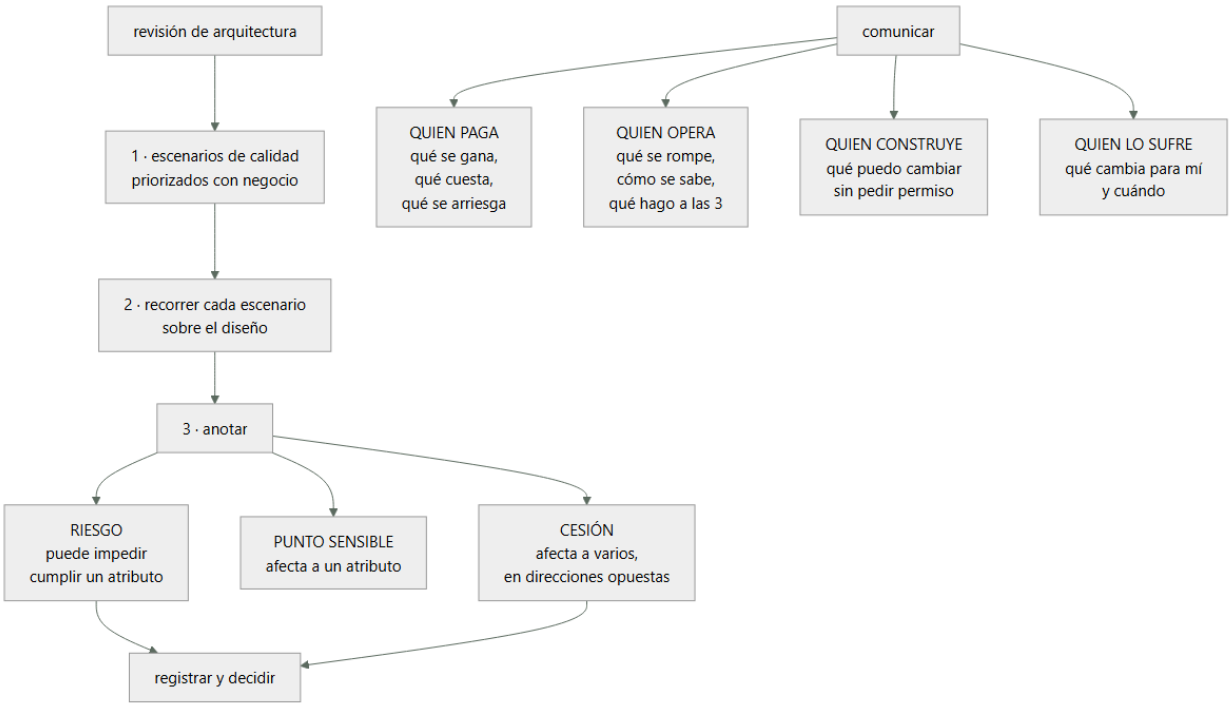
Concepto	Comprensión verificable
revisión por escenarios	Someter la arquitectura a escenarios de calidad concretos y ver cómo responde, en vez de opinar sobre el diagrama.
riesgo	Decisión que puede impedir cumplir un atributo de calidad. Es el producto principal de una revisión.
punto sensible	Decisión de la que depende un solo atributo. Cambiarla lo afecta directamente.
punto de cesión	Decisión de la que dependen varios atributos en direcciones opuestas. Es donde se decide de verdad.
audiencia	Quien paga, quien opera, quien construye y quien lo sufre. Cada una pregunta cosas distintas.
revisión de aprobación	La que solo dice sí o no. No encuentra nada y se convierte en trámite.

Modelo mental

Una arquitectura es un conjunto de decisiones sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

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. Revisión por escenarios

La revisión habitual —presentar el diagrama y pedir comentarios— produce observaciones sobre nombres, tecnologías preferidas y detalles visibles. No encuentra riesgos porque no pregunta nada concreto.

La revisión por escenarios sí:

```
1. SE PARTE DE LOS ESCENARIOS DE CALIDAD          clase 181
   los medibles, no los adjetivos

2. SE PRIORIZAN CON NEGOCIO
   importancia para el negocio × dificultad técnica percibida
   → se revisan los 5 u 8 primeros, no los veinte

3. SE RECORRE CADA ESCENARIO SOBRE EL DISEÑO
   «entra este estímulo, en estas condiciones: ¿qué pasa,
    componente por componente?»
   → y se anota dónde el diseño no tiene respuesta

4. SE CLASIFICA LO ENCONTRADO
   riesgo · punto sensible · cesión · no-riesgo

5. SE DECIDE QUÉ HACER CON CADA RIESGO
   y quién, y para cuándo
```

Y la distinción que ordena los hallazgos:

```
RIESGO
  «si el pico llega a 3.000/s, el grupo de conexiones se
   agota y la latencia se dispara»
  → tiene consecuencia y probabilidad; se decide

PUNTO SENSIBLE
  «el tamaño del grupo de conexiones determina el caudal
   máximo»
  → una sola decisión afecta a un atributo; se documenta

CESIÓN
  «leer del primario garantiza ver lo propio y aumenta la
   carga del primario»
  → afecta a dos atributos en direcciones opuestas; es donde
    hay que decidir explícitamente              clase 181
```

```
NO-RIESGO
  también se anota: «esto está resuelto, y así»
  → evita revisarlo otra vez dentro de seis meses
```

Y la regla que hace útil la revisión:

```
el producto de una revisión es una LISTA DE RIESGOS con dueño
y fecha, no un veredicto
→ «aprobado» no dice nada y no mejora el diseño
→ y una revisión que no encuentra riesgos no se hizo bien
```

Y una advertencia de método:

```
quien presenta no puede ser quien modera
y quien revisa debe poder decir «no lo sé» sin coste
→ si la revisión evalúa a la persona, deja de encontrar cosas
```

2. Cómo se conduce, en la práctica

Una revisión útil dura entre dos y cuatro horas y tiene un orden concreto.

```
ANTES
  el material se reparte con 3 días de antelación
  → diagrama C1 y C2                              clase 182
  → escenarios de calidad medibles
  → registros de decisión relevantes              clase 190
  → lo que ya se sabe que es riesgo

  quien revisa llega con preguntas escritas
  → sin esto, la sesión se llena de preguntas de contexto

DURANTE
  20 min   contexto y objetivo de negocio, con cifras
  20 min   arquitectura: C1, C2, y las decisiones caras
  30 min   priorización de escenarios con negocio presente
  90 min   recorrido de los 5-8 escenarios elegidos
  20 min   riesgos, dueños y fechas

DESPUÉS
  lista de riesgos publicada en 48 h
  cada riesgo con dueño, acción y fecha
  los no-riesgos también anotados
```

Y los cuatro papeles que hacen falta:

```
QUIEN PRESENTA        conoce el diseño
QUIEN MODERA          controla el tiempo y evita el desvío
QUIEN REVISA (2-4)    de fuera del equipo, con experiencia
                      distinta: operación, datos, seguridad
NEGOCIO               prioriza los escenarios ← imprescindible
```

Y el papel que más falta y más cambia el resultado:

```
sin negocio en la sala, los escenarios se priorizan por
criterio técnico
→ y se revisa lo interesante en vez de lo que importa
```

Las preguntas que más encuentran, por si hay poco tiempo:

```
¿qué pasa si esta dependencia responde en 5 s en vez de
  fallar?                                       clase 185
¿quién escribe este dato?                         ley 21
¿qué se rompe si desplegamos esto y aquello en orden
  distinto?
¿cómo nos enteramos de que esto ha fallado?       ley 13
¿qué pasa el día que haya que volver atrás?
¿esto se ha probado o se ha razonado?             ley 22
¿cuánto cuesta al mes y por qué esa cifra?
¿qué decisión de aquí es la más cara de cambiar?
```

Y un patrón que aparece en casi todas las revisiones:

```
el equipo conoce sus riesgos técnicos
y NO conoce los de operación ni los de retirada
→ por eso quien revisa debe venir de operación, no solo
  de arquitectura
```

3. Cuatro audiencias, cuatro preguntas

La misma arquitectura se cuenta de cuatro formas, porque cada audiencia decide algo distinto.

```
QUIEN PAGA (dirección, negocio, finanzas)
  qué se gana, en términos de negocio
  qué cuesta, al mes y de una vez
  qué se arriesga, y qué pasa si sale mal
  cuándo se nota

  NO quiere    nombres de tecnologías, diagramas de cajas
  SÍ quiere    una cifra de antes y una de después,
               con su origen                      clase 179

QUIEN OPERA (guardia, plataforma, soporte)
  qué se rompe y con qué frecuencia
  cómo me entero
  qué hago a las 3 de la mañana
  qué he de aprender y cuánto trabajo nuevo me traes

  NO quiere    la justificación de la decisión
  SÍ quiere    procedimientos, alertas y quién es el dueño

QUIEN CONSTRUYE (equipos que usarán esto)
  qué puedo cambiar sin pedir permiso
  dónde están las fronteras
  qué contrato tengo que respetar
  cómo pruebo mi parte

  SÍ quiere    fronteras claras y un camino fácil  clase 171

QUIEN LO SUFRE (usuarios, socios, otros equipos)
  qué cambia para mí
  cuándo, y qué tengo que hacer
  qué pasa si no hago nada

  SÍ quiere    fechas y una lista de acciones
```

Y el error más frecuente de comunicación:

```
contar a dirección el porqué técnico
y a operación el porqué de negocio
→ ambos escuchan educadamente y no deciden nada
```

La estructura que resiste preguntas, para cualquiera de las cuatro:

```
1. el problema, con la cifra que lo demuestra
2. lo que se descubrió que no se sabía          clase 178
3. la decisión, en una frase
4. las alternativas descartadas y por qué
5. lo que se acepta a cambio         ← lo que da credibilidad
6. cómo sabremos si funcionó, y cuándo
```

Y el punto 5 es el que más se omite y el que más confianza genera:

```
una propuesta sin desventajas no se cree
→ decir «esto añade 1,4 ms y consistencia eventual de
  10 minutos, aceptado por revenue» hace creíble el resto
```

Y una técnica que funciona con dirección:

```
presentar DOS opciones viables, no una
  con su coste, su riesgo y su plazo
  y una recomendación clara
→ una sola opción parece una decisión ya tomada que se viene
  a ratificar, y genera resistencia
```

4. Que no se convierta en trámite

La revisión de arquitectura degenera en comité de aprobación con una facilidad notable, y entonces cumple la ley 16 como cualquier otro control.

```
SEÑALES DE QUE YA ES UN TRÁMITE
  la revisión no ha rechazado ni cambiado nada en un año
  los equipos la convocan cuando ya está construido
  la salida es un «aprobado» sin lista de riesgos
```

```
  se revisa todo, sin criterio de umbral
  tarda semanas en conseguirse una fecha
  quien revisa no ha operado nada nunca
```

Y las correcciones, una por señal:

```
UMBRAL CLARO DE QUÉ SE REVISA
  lo caro de cambiar y lo que cruza fronteras de equipo
  → el resto, no                                  clase 181

PRONTO Y EN BORRADOR
  se revisa el diseño, no lo construido
  → una revisión sobre algo ya hecho solo puede aprobar o
    generar rencor

SALIDA = RIESGOS, NO VEREDICTO
  quien decide sigue siendo el equipo
  → la revisión aporta riesgos que el equipo no vio; lo que
    haga con ellos es suyo, y queda registrado

DISPONIBILIDAD RÁPIDA
  si conseguir fecha tarda tres semanas, se decide sin
  revisión y se pide perdón                        ley 16

REVISORES QUE HAN OPERADO
  quien nunca ha estado de guardia no pregunta lo que
  importa a las 3 de la mañana
```

Y una medida honesta del valor de la revisión:

```
¿cuántos riesgos encontrados por revisión se materializaron
después?
  → alto: la revisión ve bien
¿cuántos incidentes tuvieron una causa que la revisión no vio?
  → esos son los que enseñan a revisar mejor
```

Y la lista de comprobación de la clase:

- ■ la revisión parte de escenarios medibles, no del diagrama
- ■ negocio estuvo presente para priorizar
- ■ se recorrieron los escenarios sobre el diseño, uno a uno
- ■ lo encontrado está clasificado en riesgo, sensible o cesión
- ■ los no-riesgos también están anotados
- ■ cada riesgo tiene dueño, acción y fecha
- ■ la salida no es «aprobado» sino una lista
- ■ quien presenta no modera
- ■ al menos un revisor viene de operación
- ■ la revisión ocurrió sobre el diseño, no sobre lo construido
- ■ cada audiencia recibió lo que necesita decidir
- ■ se dijo explícitamente qué se acepta a cambio

Y el cierre que enlaza con la clase siguiente: con requisitos, fronteras, disponibilidad, capacidad, consistencia, contratos, amenazas, decisiones registradas y revisión, queda aplicarlo todo a un sistema completo y comprobar si se sostiene. Es la materia de la clase 192, que además cierra la parte 15.

Ejemplo trabajado

La arquitectura de reservas se somete a revisión antes de construir. Lo que sigue son los escenarios priorizados con negocio, el recorrido de los cinco primeros, los nueve riesgos encontrados —tres de los cuales el equipo no había visto— y cómo se contó la misma decisión a cuatro audiencias.

La priorización, hecha con negocio en la sala:

```
escenario                      negocio   dificultad   revisar
QA-1 latencia de búsqueda         alta      media        sí
QA-2 disponibilidad de reserva    alta      alta         sí
QA-3 añadir método de pago        alta      alta         sí
QA-5 recuperar tras borrado       alta      alta         sí
QA-6 diagnóstico sin guardia      media     alta         sí
QA-4 alcance tras compromiso      alta      media        no*
QA-7 coste en campaña             media     baja         no

* QA-4 se revisó por separado en el modelo de amenazas
```

Y una sorpresa de la priorización:

```
el equipo había puesto QA-1 (latencia) como el más importante
negocio puso QA-3 (añadir método de pago) igual de alto
motivo: había dos integraciones comprometidas por contrato
para el año siguiente, y el equipo no lo sabía
→ eso cambió el orden del trabajo                clase 181
```

Recorrido del escenario QA-2 · disponibilidad de reserva.

```
estímulo   pico de campaña; el servicio de recomendaciones cae
recorrido
  el borde recibe la petición                    ok
  la API llama a precios                         ← ¿y si tarda?
  la API llama a catálogo                        ← ¿y si tarda?
  la API escribe la reserva                      ok
  la API llama a la pasarela                     dura, aceptada
  la API publica el evento                       asíncrona, ok

HALLAZGOS
  RIESGO 1   catálogo estaba declarado blando y la llamada no
             tenía plazo ni alternativa
             → el equipo lo sabía desde la clase 185
  RIESGO 2   el plazo hacia la pasarela era de 30 s
             → en saturación llena los hilos      clase 186
             → EL EQUIPO NO LO HABÍA VISTO
  CESIÓN     precio cacheado 10 min: gana disponibilidad,
             pierde exactitud → decidida y registrada
```

Recorrido del escenario QA-3 · añadir un método de pago en 5 días.

```
recorrido, paso a paso, sobre el diseño
  añadir el proveedor nuevo en el coordinador de pago
  → el coordinador vive DENTRO de reservas          ok
  añadir la opción en la app                         ok
  añadir la conciliación contable
  → la conciliación está en facturación heredada    ← aquí

HALLAZGOS
  RIESGO 3   el escenario NO se cumple: la parte contable
             depende de un equipo externo con su propio
             calendario, y su plazo típico es de 6 semanas
             → EL EQUIPO NO LO HABÍA VISTO
             → 5 días es imposible mientras la conciliación
               viva ahí
  acción     definir un contrato de evento hacia facturación
             para que los métodos nuevos no requieran cambios
             en el heredado                        clase 188
```

Recorrido del escenario QA-6 · diagnóstico sin guardia nocturna.

```
estímulo   suben los errores de pago un martes a las 3:00
recorrido
  ¿salta una alerta?                    sí, ritmo de consumo
  ¿llega al móvil?                      sí
  ¿identifica el servicio?              sí
  ¿identifica el cambio responsable?    ← ¿hay línea de cambios?
  ¿se puede diagnosticar sin acceso de escritura?

HALLAZGOS
  RIESGO 4   la línea de cambios solo incluye despliegues
             propios; los cambios de configuración de la
             pasarela y las banderas de función no aparecen
             → EL EQUIPO NO LO HABÍA VISTO
  RIESGO 5   el panel de diagnóstico no es usable en móvil
  NO-RIESGO  la correlación por identificador de traza está
             bien resuelta; se anota para no revisarlo otra vez
```

Los nueve riesgos, con dueño y fecha:

```
#  riesgo                              dueño       fecha    est.
1  catálogo dependencia dura sin plazo  reservas    sem 2   alto
2  plazo de 30 s hacia la pasarela      reservas    sem 1   alto
3  QA-3 imposible por conciliación      arquitect.  sem 6   alto
4  línea de cambios incompleta          plataforma  sem 4   medio
5  panel no usable en móvil             plataforma  sem 8   medio
```

```
6  restauración de precios sin ensayar  datos       sem 5   alto
7  sin control de salida en la subred   seguridad   sem 3   alto
   nueva
8  el socio C con credencial estática   alianzas    trim    bajo
   (amenaza aceptada, se anota)                     revisión
9  precios: un solo escritor, sin plan  reservas    sem 10  medio
   si esa persona no está

de los nueve, los que el equipo NO había visto     3, 4, 5
y los tres venían de revisores de operación
```

Y la observación del moderador, anotada en el acta:

```
los riesgos que el equipo ya conocía eran todos técnicos
los tres nuevos eran de operación y de dependencia
organizativa
→ confirma que un revisor que ha estado de guardia encuentra
  lo que el equipo no ve
```

La misma decisión, contada a cuatro audiencias. Decisión: sacar precios a su propio servicio.

```
A DIRECCIÓN (5 minutos)
  «Los cambios de precio tardan hoy entre 2 y 6 semanas
   porque hay que coordinar con catálogo, y 4 veces este
   año una promoción salió tarde. Separando precios, pasan
   a 5 días. Cuesta 10 semanas de trabajo y unos 210 €/mes
   más. A cambio, el precio que se ve en el catálogo puede
   ir hasta 10 minutos por detrás; revenue lo ha aceptado
   por escrito. Sabremos si funcionó en el primer trimestre:
   la medida es días desde que se aprueba un precio hasta
   que está vivo.»

A OPERACIÓN (5 minutos)
  «Un servicio más, con su guardia. Se rompe si su almacén
   no responde: entonces reservas usa el último precio válido
   y sigue funcionando, así que no es una llamada de
   madrugada. La alerta que sí lo es: retraso de propagación
   por encima de 15 minutos. Procedimiento escrito y probado
   por alguien de fuera del equipo. Dueño: equipo de
   reservas. Y os quitamos tres alertas de ‹precio
   incorrecto› que llevaban dos años sin causa conocida.»

A LOS EQUIPOS QUE CONSTRUYEN
  «El precio ya no está en la tabla de catálogo. Se pide al
   servicio de precios o se escucha su evento. Contrato
   versionado, compatible hacia delante, con pruebas de
   contrato en nuestra canalización: si rompemos algo, falla
   nuestro despliegue, no el vuestro. Podéis cambiar vuestra
   parte sin pedirnos nada mientras respetéis el contrato.»

AL SOCIO Y AL EQUIPO DE INFORMES (quien lo sufre)
  «A partir del 14 de octubre, el campo precio del catálogo
   deja de actualizarse en tiempo real y puede ir hasta 10
   minutos por detrás. Si necesitáis el precio exacto en el
   momento, usad el endpoint de precios, que os damos ahora.
   Si no hacéis nada, veréis precios con hasta 10 minutos de
   retraso. Contacto para dudas: X.»
```

Y el detalle que hizo que dirección aprobara sin discusión:

```
se presentaron DOS opciones
  A  separar precios              10 semanas, 210 €/mes
  B  mantener y añadir validación  3 semanas, 0 €
     → no resuelve el bloqueo; los 5 días siguen siendo
       6 semanas
  recomendación: A, con el motivo

→ y la pregunta de dirección fue sobre el plazo, no sobre
  si hacerlo
```

La lección que esta clase deja: la revisión encontró nueve riesgos y los tres que el equipo no había visto no eran técnicos: un plazo demasiado largo hacia un tercero, una línea de cambios incompleta y un escenario de negocio que resultaba imposible por depender de un equipo externo. Los tres los encontraron revisores que habían estado de guardia. Y el escenario que negocio priorizó más alto —añadir un método de pago— el equipo ni siquiera sabía que

estaba comprometido por contrato.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-15-systems-architecture-engineering/191-architecture-review-y-comunicacion-con-stakeholders/lab.py
```

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

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

Reto verificable

Construye 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 solo produce comentarios sobre nombres y tecnologías	Se presentó el diagrama y se pidieron opiniones, sin escenarios concretos	Parte de escenarios de calidad medibles y recórrelos uno a uno sobre el diseño.
Se revisa lo interesante en vez de lo que importa	Los escenarios se priorizaron con criterio técnico	Prioriza con negocio presente, cruzando importancia de negocio y dificultad técnica.
La revisión no ha cambiado nada en un año	Es un comité de aprobación que llega cuando ya está construido	Revisa el diseño en borrador, con umbral claro de qué se revisa, y que la salida sea una lista de riesgos con dueño y fecha.
El equipo sale de la revisión sin haber aprendido nada nuevo	Los revisores tienen el mismo perfil que quien presenta	Incluye al menos un revisor que haya estado de guardia; los riesgos de operación y de retirada son los que el equipo no ve.
Dirección escucha la propuesta y no decide	Se le contó el porqué técnico en vez de coste, riesgo y plazo	Presenta dos opciones viables con su coste y riesgo, y una recomendación clara.
Una propuesta genera desconfianza pese a ser buena	Se presentó sin desventajas	Di explícitamente qué se acepta a cambio y quién lo ha aceptado.

■ Seguridad, ética y costo

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

■ Preguntas de comprobación

1. ¿Qué diferencia hay entre riesgo, punto sensible y punto de cesión?
2. ¿Por qué el producto de una revisión no debe ser un veredicto?
3. ¿Qué aporta tener a negocio en la sala durante la priorización?
4. ¿Qué necesita saber quien opera y qué no le interesa?
5. ¿Qué señales indican que la revisión se ha convertido en trámite?

Referencias

- Clements, P., Kazman, R. y Klein, M. (2001). Evaluating Software Architectures: Methods and Case Studies (ATAM). <https://www.oreilly.com/library/view/evaluating-software-architectures/9780201704822/>
- SEI (2000). ATAM: Method for Architecture Evaluation. <https://insights.sei.cmu.edu/library/atam-method-for-architecture-evaluation/>
- Bass, L., Clements, P. y Kazman, R. (2021). Software Architecture in Practice, cap. de evaluación. <https://www.oreilly.com/library/view/software-architecture-in/9780136886051/>
- Brown, S. (2019). Software Architecture for Developers, vol. 2 — comunicación por audiencias. <https://leanpub.com/visualising-software-architecture>
- Google (2025). Architecture review practices en Cloud Architecture Framework. <https://cloud.google.com/architecture/framework>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 15 en PDF · Manual integral

Anterior	Índice	Siguiente
← 190 · ADRs, fitness functions y gobierno de decisiones	Parte 15 · Programa	192 · Proyecto: arquitectura completa de CloudShop →

Evaluación — 191 Architecture review y comunicación con stakeholders

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.

192 — Proyecto: arquitectura completa de CloudShop

Parte: 15 — Arquitectura de sistemas e ingeniería de requisitos Nivel: intermedio-avanzado · Horas estimadas: 8
Laboratorio: capstone · Estado: EXECUTABLECORE

Propósito

Aplicar las once clases anteriores a una arquitectura completa de CloudShop y comprobar si se sostiene. La clase da el guion del proyecto, el orden en que se toman las decisiones, el documento que hay que entregar y los criterios con que se evalúa. Y cierra la parte 15: corrige las cinco predicciones de la clase 180 —dos acertadas, tres a medias—, actualiza el recuento de leyes, añade la ley 24 y escribe la hipótesis de la parte 16.

Resultados de aprendizaje

Al finalizar podrás:

1. Producir una arquitectura completa con el método de toda la parte.
2. Ordenar las decisiones de la más cara de cambiar a la más barata.
3. Entregar un documento que resista una revisión por escenarios.
4. Corregir las cinco predicciones de la clase 180 con evidencia.
5. Escribir la hipótesis de la parte 16 en forma refutable.

Conceptos centrales

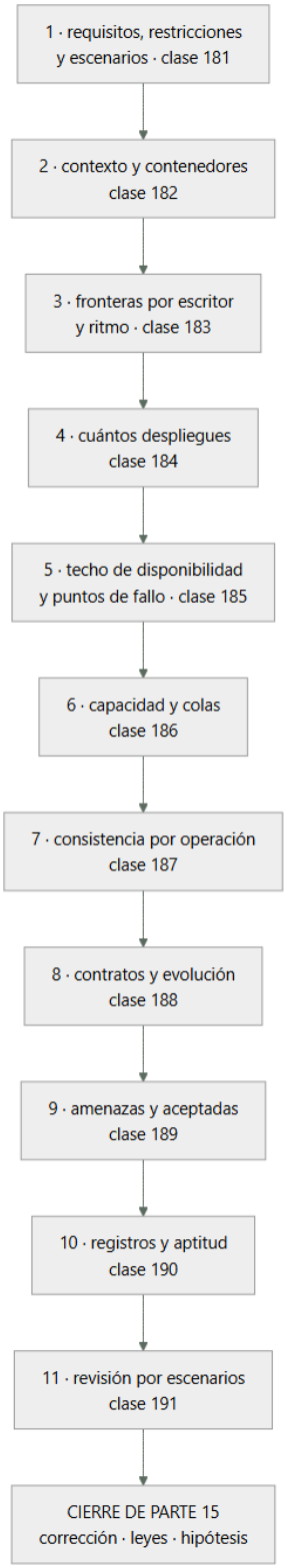
Concepto	Comprensión verificable
arquitectura completa	Conjunto coherente de decisiones que cubre requisitos, fronteras, fiabilidad, capacidad, datos, contratos, amenazas y gobierno.
orden de decisión	Secuencia por coste de cambio: primero lo caro, después lo barato y con datos.
coherencia	Que las decisiones no se contradigan entre sí. Se comprueba cruzando escenarios con decisiones.
entregable	El documento que otra persona puede revisar y usar para construir.
ley 24	Lo que no está en el diagrama no se analiza, y lo que se omite es siempre del mismo tipo.
hipótesis de parte	Afirmación refutable escrita antes de estudiar, que la parte siguiente corrige con evidencia.

Modelo mental

Una arquitectura es un conjunto de decisiones sobre estructuras, interfaces y atributos de calidad, respaldadas por escenarios y evidencia.

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 encargo y su orden

El encargo. CloudShop quiere rehacer su plataforma de pedidos, que hoy es un sistema heredado de nueve años con sesenta equipos alrededor. El proyecto consiste en producir la arquitectura, no en construirla.

```
lo que se entrega
  el documento de arquitectura completo
```

```
   las decisiones registradas
   la lista de riesgos de su propia revisión
   y lo que se decidió NO hacer
```

El orden de las decisiones, que es el mismo criterio de coste de cambio de la clase 181:

```
1  REQUISITOS Y ESCENARIOS                          clase 181
   funcionales, restricciones comprobadas y 6-8 escenarios
   medibles, con los conflictos resueltos

2  CONTEXTO Y CONTENEDORES                          clase 182
   C1 y C2, con flechas anotadas y quién escribe cada almacén

3  FRONTERAS                                        clase 183
   por escritor de dato y por ritmo de cambio, con el
   porcentaje de cambios acoplados del histórico

4  DESPLIEGUES                                      clase 184
   cuántas unidades, con uno de los cinco motivos y su cifra

5  FIABILIDAD                                       clase 185
   techo calculado, duras convertidas en blandas, análisis
   de puntos de fallo con las cuatro preguntas

6  CAPACIDAD                                        clase 186
   concurrencia con Little, codo con modelo abierto, margen
   que cubra la pérdida de una zona

7  CONSISTENCIA                                     clase 187
   por operación, con la consecuencia de negocio

8  CONTRATOS                                        clase 188
   qué es contrato, cómo evoluciona, cómo se retira

9  AMENAZAS                                         clase 189
   flujos, fronteras de confianza, y las aceptadas por escrito

10 GOBIERNO                                         clase 190
   registros con premisas y funciones de aptitud

11 REVISIÓN                                         clase 191
   por escenarios, con riesgos, dueños y fechas
```

Y dos reglas sobre el orden:

```
los pasos 1 a 4 condicionan todo lo demás y se hacen primero
los pasos 5 a 7 pueden obligar a volver al 3 o al 4
→ volver es normal; lo anormal es no volver nunca
```

Y el error de método que más se comete en este proyecto:

```
empezar por el paso 2, dibujando
→ el diagrama sale bonito y no responde a ningún escenario
→ y las fronteras se copian del organigrama            ley 21
```

2. El entregable y su evaluación

El documento, que cabe en unas quince páginas y no debe tener más:

```
1  PROBLEMA Y CONTEXTO
   la cifra que demuestra que hay un problema
   lo que se descubrió que no se sabía              clase 178

2  REQUISITOS
   funcionales en una lista corta
   restricciones, con la evidencia de que son reales
   escenarios de calidad medibles, con punto de medida
   conflictos y quién cede
   lo que el sistema NO hará

3  ARQUITECTURA
   C1 y C2 con flechas anotadas
   fronteras, con el criterio que las sitúa
   despliegues, con el motivo de cada separación
   vista de datos: quién escribe qué
```

```
4  PROPIEDADES
   techo de disponibilidad, calculado
   puntos únicos de fallo, con los no técnicos
   capacidad: codo, margen y concurrencia
   consistencia por operación

5  EVOLUCIÓN
   contratos y su versionado
   plan de retirada de lo viejo

6  SEGURIDAD
   flujos y fronteras de confianza
   amenazas mitigadas y ACEPTADAS con nombre y fecha

7  GOBIERNO
   registros de decisión (los caros de cambiar)
   funciones de aptitud

8  RIESGOS
   de la propia revisión, con dueño y fecha

9  LO QUE NO SE HACE, y por qué
```

Los criterios de evaluación, publicados antes de empezar:

```
1  los escenarios son medibles y tienen punto de medida    2
2  las restricciones están comprobadas, no supuestas       1
3  las fronteras se justifican con datos del histórico     3
4  cada separación en servicio tiene motivo y cifra        2
5  el techo de disponibilidad está calculado               2
6  el análisis de fallos incluye los no técnicos           3
7  la capacidad usa Little y modelo abierto                2
8  la consistencia se decide por operación                 3
9  los contratos tienen plan de retirada                   1
10 hay amenazas ACEPTADAS por escrito                      2
11 los registros tienen premisas y alternativas            2
12 la revisión produjo riesgos que el diseño no veía       3
13 está escrito lo que NO se hace                          1
14 hay al menos una decisión marcada como equivocada       3
```

Y el criterio 14 pesa como los que más, por el mismo motivo que en la clase 178:

```
un proyecto sin ninguna decisión corregida no se ha revisado
→ y el archivo de decisiones sin ninguna superada es un
  archivo de justificaciones                          clase 190
```

Y tres errores que hunden este proyecto:

```
ESCENARIOS QUE SON ADJETIVOS
  «rápido», «escalable», «seguro»
  → no se puede diseñar contra eso                    clase 181

SEPARAR EN SERVICIOS SIN DIVIDIR DATOS
  → monolito distribuido                              clase 184

PROMETER UNA DISPONIBILIDAD SIN CALCULARLA
  → se incumple por aritmética                        clase 185
```

3. Cierre de la parte 15: corrección de las cinco predicciones

Las cinco predicciones de la clase 180, corregidas con la evidencia de las clases 181 a 191.

```
1. «la parte 15 formalizará con vocabulario lo que el programa
    ya obtuvo por evidencia; el valor no estará en los nombres
    sino en poder discutir una decisión antes de tomarla»

   CORRECTA, y en las dos mitades. Casi todo lo de esta parte
   ya había aparecido: el acoplamiento por datos era la ley 21
   desde la clase 147; el techo de disponibilidad se había
   calculado en la 164; el codo con modelo abierto salió en la
   129. Lo que añadió la parte fue poder discutirlo antes:
   el registro 007 cerró en diez minutos una propuesta que
   habría costado dos semanas de discusión.

2. «de los atributos de calidad, el que más decisiones cambiará
```

```
     será la modificabilidad, porque es el único que se paga
     todos los meses»

   A MEDIAS, y la distinción importa. La modificabilidad decidió
   la ESTRUCTURA: las fronteras de la clase 183, la separación
   de precios de la 184 y el escenario QA-3 que nadie había
   pedido con esas palabras. Pero contando decisiones una a una,
   la disponibilidad cambió más cosas: once puntos únicos de
   fallo, cinco conversiones de dependencia, y el techo que
   obligó a renegociar una promesa contractual. Acertamos qué
   atributo decide la arquitectura y fallamos al confundirlo
   con el que decide más veces.

3. «la frontera correcta coincidirá con quién escribe cada dato
     en más del 70 % de los casos»

   CASI. De las seis decisiones de frontera de la clase 183,
   cuatro las decidió el escritor del dato (67 %) y dos el
   ritmo de cambio. La cifra se quedó tres puntos corta, y el
   mecanismo era el correcto: el criterio del escritor colocó
   la mayoría de las fronteras, y ninguna de las que colocó
   hubo que mover después.

4. «el análisis de puntos de fallo encontrará que la mayoría de
     los puntos únicos no son de infraestructura sino de
     conocimiento y de procedimiento»

   PRIMERA MITAD CORRECTA Y SUBESTIMADA: 8 de 11 no eran de
   infraestructura (73 %). SEGUNDA MITAD MAL REPARTIDA: los de
   conocimiento y procedimiento sumaban 5 de 11 (45 %), que no
   es mayoría. Los otros tres eran de datos y de respuesta
   degradada, una categoría que la predicción no contemplaba
   y que resultó ser de las que más daño hacen: una réplica
   lenta es peor que una caída.

5. «los registros de decisión y las funciones de aptitud
     fracasarán como los controles: si estorban, se rodean; y
     el registro se rellenará después de decidir, para cumplir»

   PRIMERA MITAD CORRECTA: dos de once funciones de aptitud se
   retiraron, y una de ellas —el límite de tamaño de fichero—
   había generado cuarenta y una excepciones sin detectar un
   solo problema real. SEGUNDA MITAD NO SE CUMPLIÓ, y no por
   suerte: se cumplió lo contrario porque se adoptó la práctica
   de escribir el borrador ANTES de decidir. Predijimos un
   fracaso que era evitable con una regla de método, y no
   habíamos previsto la regla.
```

Marcador: dos correctas, tres a medias. Y el patrón de los fallos es el mismo de la parte 14: acertamos los mecanismos y fallamos los repartos. Predecir qué fuerza actúa resulta más fácil que predecir cuánto pesa cada una.

El recuento de leyes, cerrada la parte 15.

```
ley 13  lo que no se mira deja de funcionar en silencio        33
ley 15  la señal existe y nadie la mira                        25
ley 14  el coste se decide al crear, no al pagar               22
ley 22  un procedimiento nunca ejecutado no funciona           20
ley 16  un control que estorba se rodea                        20
ley 21  el acoplamiento vive en quién escribe                  17
ley 20  lo que no tiene dueño se filtra y se desperdicia       16
ley 19  la compensación hace invisible el fallo                10
ley 17  se optimiza la medida, no el objetivo                  10
ley 18  lo asíncrono traslada la garantía, no la elimina        8
ley 23  la capacidad la limita lo que ya se mantiene            7
```

Y la parte 15 obliga a escribir una ley nueva:

```
LEY 24
  lo que no está en el diagrama no se analiza,
  y lo que se omite del diagrama es siempre del mismo tipo

apariciones en esta parte                                      4
  clase 182   5 elementos reales no dibujados, entre ellos un
              punto de entrada externo con 3 años de vida
```

```
  clase 185   8 de 11 puntos únicos no aparecían en ningún
              diagrama
  clase 189   8 de 19 amenazas estaban en las 3 fronteras que
              faltaban: webhook, canalización y salida a
              analítica
  clase 191   los 3 riesgos que el equipo no vio eran de
              operación y de dependencia organizativa

y lo que se omite es siempre lo mismo
  lo que entra de fuera sin que lo llamemos nosotros
  lo que despliega en vez de servir
  lo que sale hacia analítica o hacia terceros
  lo que hacen las personas
```

4. La hipótesis de la parte 16

Escrita antes de estudiar la parte 16 (clases 193 a 204, redes en profundidad), para que la clase 204 la corrija con evidencia:

```
1. la red va a resultar ser la capa donde más decisiones
   irreversibles se toman con menos deliberación: los rangos
   de direcciones y el diseño de conectividad se eligen en
   una tarde y condicionan una década                ley 14

2. de todos los problemas de red que aparezcan, la mayoría no
   serán de encaminamiento ni de rendimiento sino de NOMBRES
   Y CERTIFICADOS: resolución, propagación y caducidad

3. el coste de salida y el tráfico entre zonas volverán a ser
   la línea más grande y la peor atribuida, como en la
   clase 168; y seguirá sin tener dueño               ley 20

4. la malla de servicios aparecerá otra vez como respuesta a
   problemas que ya están resueltos en otra capa, y el
   análisis honesto volverá a dejarla en una o dos capacidades
   que sí aporta                                   clase 152

5. el diagnóstico de red será donde más se note la ley 24: los
   diagramas de red omitirán sistemáticamente lo mismo —lo que
   entra de fuera, lo que sale a terceros y las rutas que
   alguien añadió a mano— y ahí estarán los incidentes
```

Y el cierre de la parte 15: de once clases, lo que más problemas destapó no fue ninguna técnica nueva, sino mirar el sistema con una pregunta que nadie hacía: quién escribe este dato, qué pasa si esto responde lento en vez de caerse, y qué hay funcionando que no está dibujado. La parte 16 baja una capa —a las direcciones, las rutas, los nombres y los certificados— y empieza por la decisión que más veces se toma sin pensarla: el plan de direccionamiento. Es la clase 193.

Ejemplo trabajado

La arquitectura de pedidos de CloudShop, resuelta con el método de la parte. Lo que sigue es el resumen del entregable, con las cifras que decidieron cada paso y las dos decisiones que hubo que corregir durante el propio proyecto.

Paso 1 · Requisitos y escenarios.

```
el problema, con cifra
  el plazo medio para poner en producción un cambio de
  catálogo es de 5 semanas; 3 competidores lo hacen en días
  y el margen por pedido cayó de 4,10 € a 3,25 € en 2 años

restricciones comprobadas
  AWS por compromiso de consumo hasta 2029          real
  datos de clientes de la UE en la UE               legal
  el sistema de facturación no se toca              real
  27 personas en 4 equipos; no hay más este año     real
  «para el Black Friday»    → ¿y si no?              NO era
                              restricción: la campaña se
                              lanza igual

escenarios (7, medibles)
```

```
  QA-1  p99 del listado ≤ 300 ms en el borde, a 8.000/s
  QA-2  el flujo de compra 99,9 % mensual
  QA-3  añadir una categoría de producto nueva en ≤ 3 días,
        sin coordinar despliegues
  QA-4  alcance desde un servicio comprometido: ≤ 2 almacenes
  QA-5  restaurar tras borrado: pérdida ≤ 5 min, servicio
        ≤ 1 h, MEDIDO
  QA-6  diagnóstico del flujo de compra en ≤ 10 min desde el
        móvil
  QA-7  coste por pedido ≤ 0,21 €

conflictos resueltos
  QA-1 × QA-3   cede QA-1 de 300 a 380 ms; acepta producto
  QA-2 × QA-7   cede QA-2 de 99,9 % a 99,8 %; sin segunda
                región activa; acepta dirección de tecnología

lo que NO hará
  no sustituye facturación
  no gestiona el almacén físico
  no soporta suscripciones en esta fase
  no se internacionaliza fuera de la UE
```

Paso 3 · Fronteras, con datos del histórico.

```
cambios acoplados, 18 meses
  catálogo × precios                  68 %   → juntos o
                                             precio fuera
  pedidos × pagos                     74 %   → juntos
  catálogo × inventario               12 %   → separados
  pedidos × envíos                    31 %   → dudoso
  clientes × marketing                 4 %   → separados

ritmo de cambio
  precios                 11/mes   revenue
  contenido de catálogo    2/mes   producto
  reglas de envío          7/mes   logística
  flujo de pedido          2/mes   producto
  inventario               1/mes   operaciones

escritores actuales, del C2
  tabla de catálogo        4 escritores  ← el problema
  tabla de pedidos         3 escritores
  tabla de inventario      3 escritores

fronteras resultantes (6)
  pedidos+pagos · precios · catálogo · inventario ·
  envíos · clientes

  decididas por escritor de dato                  4
  decididas por ritmo de cambio                   2
```

Paso 4 · Despliegues: 4, no 6.

```
monolito modular   pedidos+pagos, catálogo, clientes
precios            servicio   motivo 1 (11/mes vs 2) y 3
envíos             servicio   motivo 5 (equipo de logística real)
inventario         servicio   motivo 4 (dato compartido con el
                              almacén físico, otro régimen)

coste marginal   3 × 2,7 = 8,1 días-persona/mes sobre 540
                 → 1,5 % de la capacidad
```

Paso 5 · Fiabilidad: el techo obligó a renegociar.

```
techo inicial del flujo de compra
  borde 99,99 × API 99,95 × base 99,99 × precios 99,90
  × inventario 99,90 × pasarela 99,95 × identidad 99,99
  = 99,67 %

prometido en QA-2 (ya cedido)                  99,80 %
→ incumplido por aritmética

conversiones
  precios      → último valor válido, 15 min        blanda
  identidad    → validación local                   fuera
```

```
  inventario   → NO convertible: vender sin stock cuesta más
                 que no vender

techo final   99,84 %      cumple

puntos únicos encontrados                              13
  de infraestructura                                    4
  de datos                                              2
  de respuesta degradada                                2
  de conocimiento                                       2
  de procedimiento                                      3
→ 9 de 13 no aparecían en el C2
```

Paso 6 · Capacidad.

```
Little sobre el listado
  λ objetivo 8.000/s, W 45 ms → L = 360 simultáneas
  grupo dimensionado para ρ = 0,65 → 554 conexiones
  el almacén admite 400
  → decisión: caché de listado con 92 % de aciertos,
    λ efectiva al almacén 640/s → L = 29

codo, modelo abierto, catálogo completo
  medido            7.400/s
  prueba cerrada previa decía        19.000/s   ×2,6

margen
  pico previsto 8.000/s; 3 zonas; ρ objetivo 0,65
  capacidad = 8.000 / (0,65 × 0,66) = 18.650/s
```

Paso 7 · Consistencia, por operación.

```
comprar (reserva de stock)      linealizable   fila por unidad
cobrar                          fuerte + idem.
ver mi pedido                   de sesión
listar catálogo                 eventual ≤ 5 s
ver precio                      eventual ≤ 15 min
panel de ventas                 eventual ≤ 60 s, marcado

de 6, dos necesitan garantía fuerte, y son el 4 % del tráfico
```

Las dos decisiones que hubo que corregir durante el proyecto:

```
CORRECCIÓN 1   envíos se había separado por motivo 1
  la cifra decía 7 cambios/mes frente a 2
  al revisar el histórico con más detalle, 5 de esos 7 eran
  cambios de TARIFA, un dato, no de lógica
  → el motivo real era el 5 (equipo de logística propio),
    no el 1
  → la separación se mantiene, pero por otra razón, y eso
    cambia qué sale con ella: las tarifas se quedan como
    dato con un escritor, no como servicio

CORRECCIÓN 2   inventario iba a ser eventual
  premisa   «el stock se puede ajustar después»
  la revisión por escenarios preguntó qué pasa en Black
  Friday con 400 pedidos/s sobre 12 unidades
  → sobreventa de 300 unidades en 40 segundos
  → la premisa venía de la operación en días normales
  → decisión corregida: linealizable con fila por unidad
  registro 014, superando el 009
```

El resultado de la revisión por escenarios:

```
riesgos encontrados                                    11
  que el equipo ya conocía                              7
  que NO conocía                                        4
    · el plazo de 20 s hacia la pasarela
    · la línea de cambios no incluye banderas de función
    · QA-3 imposible mientras la conciliación viva en
      facturación
    · las tarifas de envío las escribe también un fichero
      que sube logística a mano cada lunes

los 4 los encontraron revisores de operación y de datos
```

Y el último es el que mejor resume la parte:

```
un fichero subido a mano cada lunes escribía un dato que
el diseño daba por tener un solo escritor
→ no estaba en ningún diagrama                          ley 24
→ no lo sabía nadie del equipo de arquitectura
→ y llevaba cuatro años funcionando
```

Lo que se decidió no hacer, escrito:

```
no se separa catálogo de contenido: 2 cambios/mes no lo
  justifican
no se monta segunda región activa: 7.900 €/mes frente a
  510 €/mes de pérdida esperada
no se adopta malla de servicios: de sus 5 capacidades, 4 ya
  están resueltas                                clase 152
no se migra facturación: fuera de alcance, y declarado como
  deuda con fecha de revisión a 12 meses
```

La lección que este proyecto deja: de las once clases de la parte, la que más cambió el resultado no fue ninguna de las técnicas —fue preguntar quién escribe cada dato, que colocó cuatro de las seis fronteras y descubrió un fichero que alguien subía a mano los lunes. Y las dos decisiones que hubo que corregir se corrigieron por la misma causa: una premisa que venía de la operación en días normales y que nadie había comprobado contra el escenario que importaba.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-15-systems-architecture-engineering/192-proyecto-arquitectura-completa-de-
cloudshop/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 cloudshop-system-design 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 cloudshop-system-design 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 diagrama sale bonito y no responde a ninguna pregunta	Se empezó dibujando en vez de escribiendo escenarios	Empieza por requisitos, restricciones comprobadas y escenarios medibles; dibuja después.
Las fronteras acaban copiando el organigrama	No se usaron datos del histórico para situarlas	Calcula cambios acoplados y ritmo de cambio, e identifica quién escribe cada almacén.
La disponibilidad prometida se incumple desde el primer día	Se prometió sin calcular el techo de dependencias duras	Multiplica las duras, convierte las que puedas en blandas y renegocia la promesa antes de firmarla.
Una decisión de consistencia funciona en días normales y falla en campaña	La premisa venía de la operación habitual y no se contrastó con el escenario extremo	Recorre cada decisión contra el escenario de calidad más exigente antes de cerrarla.
El proyecto no tiene ninguna decisión corregida	No se revisó de verdad, o se corrigió sin dejar constancia	Marca las decisiones superadas y escribe qué premisa falló; es la parte más valiosa del archivo.
Aparece un escritor de datos que nadie conocía	Solo se analizó lo que estaba dibujado	Contrasta el diagrama con las dependencias observadas y busca en concreto lo que entra de fuera, lo que sale a terceros y lo que hacen las personas a mano.

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 los pasos 1 a 4 se hacen antes que los demás?
2. ¿Qué criterio de evaluación pesa como los que más y por qué?
3. ¿Cuál de las cinco predicciones de la clase 180 falló y en qué mitad?
4. ¿Qué dice la ley 24 y qué cuatro cosas se omiten siempre de los diagramas?
5. ¿Qué pregunta colocó cuatro de las seis fronteras del proyecto?

Referencias

- Bass, L., Clements, P. y Kazman, R. (2021). Software Architecture in Practice, 4.ª ed. <https://www.oreilly.com/library/view/software-architecture-in/9780136886051/>
- Ford, N., Richards, M. y otros (2021). Software Architecture: The Hard Parts — decisiones con compromisos explícitos. <https://www.oreilly.com/library/view/software-architecture-the/9781492086888/>
- Kleppmann, M. (2017). Designing Data-Intensive Applications. <https://dataintensive.net/>
- AWS (2025). Well-Architected Framework. <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html>
- Google (2025). Cloud Architecture Framework. <https://cloud.google.com/architecture/framework>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 15 en PDF · Manual integral

Evaluación — 192 Proyecto: arquitectura completa de CloudShop

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 cloudshop-system-design 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.