Multi-Cloud Engineering Program

Parte 10 — Observabilidad, SRE y confiabilidad

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	10 de 23
Clases	121–132
Evaluaciones	12
Horas estimadas	52

Índice

Parte 10 — Observabilidad, SRE y confiabilidad

← Parte 09 · Índice completo · Parte 11 →

Descargar: Esta parte en PDF · Manual integral

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

Operar sistemas mediante señales, objetivos y aprendizaje de incidentes.

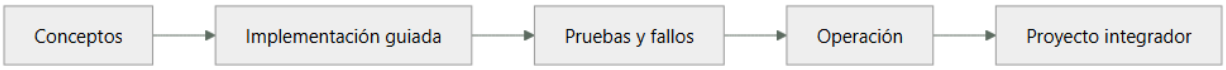
Resultados de la parte

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

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

Secuencia



Clases

ID	Clase	Laboratorio	Horas
121	Logs, métricas, trazas y eventos como señales	observability	4
122	Logging estructurado, correlación y retención	observability	4
123	Métricas, cardinalidad y modelos RED y USE	metrics	4
124	Tracing distribuido y OpenTelemetry	observability	4
125	Dashboards, alertas accionables y fatiga	observability	4
126	SLI, SLO, SLA y presupuesto de error	sre	4
127	Incidentes, severidad, comando y comunicación	incident	4
128	Runbooks, playbooks y automatización operativa	operations	4
129	Capacidad, rendimiento y pruebas de carga	performance	4
130	Timeouts, retries, backoff, circuit breaker y bulkhead	reliability	4
131	Chaos engineering y game days	chaos	4
132	Proyecto: operación SRE 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

- Site Reliability Engineering — Beyer et al..
- The Site Reliability Workbook — Beyer et al..
- Observability Engineering — Majors, Fong-Jones y Miranda.

121 — Logs, métricas, trazas y eventos como señales

Parte: 10 — Observabilidad, SRE y confiabilidad Nivel: avanzado · Horas estimadas: 4 Laboratorio: observability · Estado: EXECUTABLECORE

Propósito

Abrir la parte con la cifra que la cierra la 120: de veintiún problemas, tres se detectaron por una alerta y dieciocho cuando ya habían hecho daño. Esta clase establece el material con el que se corrige eso: cuatro señales, cada una con una pregunta que responde y otras que no puede responder. Y defiende una tesis concreta: la cuarta —qué cambió y cuándo— es la que cierra más incidentes en menos tiempo, y es la única que casi nadie instrumenta.

Resultados de aprendizaje

Al finalizar podrás:

1. Asignar a cada señal la pregunta que responde y las que no.
2. Correlacionar las cuatro con los mismos identificadores.
3. Instrumentar los cambios como señal de primera clase.
4. Prever el coste de una señal en el momento de emitirla, no de consultarla.
5. Decidir el muestreo sabiendo qué preguntas está descartando para siempre.

Conceptos centrales

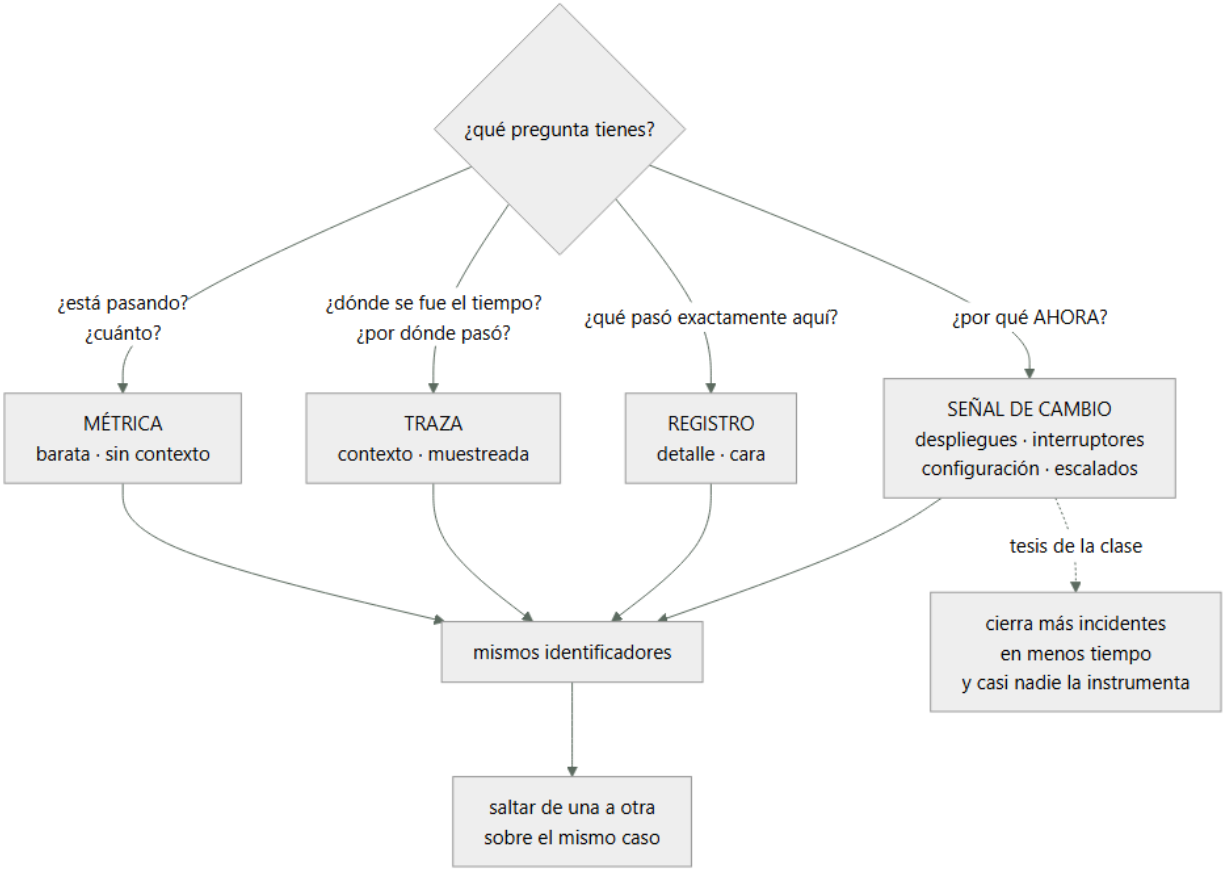
Concepto	Comprensión verificable
métrica	Valor numérico agregado en el tiempo. Barata y comparable; no lleva contexto de ningún caso concreto.
traza	Recorrido de una petición por los servicios que la atienden, con tiempos por tramo. Responde dónde se fue el tiempo.
registro	Línea con detalle de lo ocurrido en un punto. Es la señal más expresiva y la más cara por volumen.
señal de cambio	Registro de lo que se modificó: despliegues, interruptores, configuración, escalados, migraciones. Responde «por qué ahora».
correlación	Poder saltar de una señal a otra sobre el mismo caso, porque comparten identificadores.
coste en la emisión	El precio de una señal lo fija lo que se emite, no lo que se consulta. Guardarlo todo por si acaso es la causa principal de la factura.
muestreo	Quedarse con una parte de las trazas. Decide qué preguntas se podrán hacer después, porque lo no emitido no existe.

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

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 señales, cuatro preguntas

La clasificación útil no es por formato sino por la pregunta que cada una contesta:

```
MÉTRICA          ¿está pasando? ¿cuánto? ¿va peor que ayer?
  responde       agregados, tendencias, comparaciones
  no responde    qué le pasó a esta petición concreta
  coste          bajo por dato, y crece con la CARDINALIDAD (clase 123)

TRAZA            ¿por dónde pasó y dónde se fue el tiempo?
  responde       la cadena de llamadas de un caso, con tiempos
  no responde    con qué frecuencia ocurre, si está muestreada
  coste          medio, y se controla con el muestreo

REGISTRO         ¿qué pasó exactamente en este punto?
  responde       el detalle: valores, decisiones, errores
  no responde    nada agregado sin procesarlo antes
  coste          alto, proporcional al VOLUMEN

SEÑAL DE CAMBIO  ¿por qué ahora?
  responde       qué se modificó y cuándo
  no responde    el efecto; hay que cruzarla con las otras
  coste          casi nulo: son unos cientos de eventos al día
```

Y la asimetría que justifica la tesis de la clase: la cuarta es la más barata de todas y la que menos se instrumenta.

La razón por la que funciona tan bien es estadística: la mayoría de los problemas empiezan porque algo cambió, y las partes 08 y 09 han multiplicado las cosas que pueden cambiar sin desplegar código:

```
despliegue de una versión                    clase 102
cambio de un interruptor                     clase 105
confirmación en el repositorio de entorno    clase 103
escalado de instancias o de consumidores     clases 113, 117
cambio de una regla en la puerta de entrada  clase 118
migración de datos o cambio de esquema       clases 109, 115
```

```
caducidad de una credencial o de un certificado
rotación de una clave
ventana de mantenimiento del proveedor
```

Y lo que hace falta es poco: una serie temporal de eventos con quién, qué y cuándo, dibujada encima de los gráficos.

```
{"momento":"2026-08-03T09:12:04Z","tipo":"interruptor",
 "que":"recomendaciones-v3","de":"5%","a":"100%",
 "quien":"…","referencia":"…"}
```

La pregunta que responde —«¿qué cambió en los diez minutos anteriores al problema?»— es la primera de cualquier incidente, y sin esta señal se contesta preguntando por los canales del equipo.

2. Vigilar y poder preguntar

Hay dos actividades distintas que se llaman parecido y conviene separar:

```
VIGILAR       preguntas conocidas de antemano
              paneles y alertas escritos antes de que pase nada
              → responde «¿esto que sé que puede fallar, está fallando?»

PODER PREGUNTAR   preguntas nuevas sobre datos ya recogidos
              → responde «¿qué demonios está pasando ahora?»
```

Y los dos hacen falta. Lo que no funciona es esperar que uno haga el trabajo del otro:

```
solo vigilancia    solo se detectan los fallos que alguien anticipó
                   y los de la parte 09 no los anticipó nadie
solo exploración   nadie está mirando a las 3 de la madrugada
```

Y la propiedad que separa un sistema del que se pueden hacer preguntas nuevas de uno del que no:

```
¿puedo filtrar por una dimensión que no había previsto?
  versión, cliente, región, tipo de dispositivo, plan, ruta
  → si la dimensión no se emitió, la respuesta es no, para siempre
```

Y de ahí la regla que gobierna esta parte entera:

```
lo que no se emite no existe
y lo que se emite se paga
```

Las dos mitades tiran en direcciones opuestas, y resolver esa tensión es el trabajo. Lo que la resuelve razonablemente bien:

```
métricas          pocas dimensiones, muy vigiladas            clase 123
trazas            muchas dimensiones, muestreadas             clase 124
registros         detalle completo, retención corta           clase 122
cambios           todo, siempre: es barato
lago              lo que haya que conservar mucho tiempo,
                  en formato columnar                         clase 112
```

Y una cuarta capa que a menudo falta: los datos derivados que no se pueden recalcular. Un agregado diario de peticiones por cliente ocupa poco y permite responder preguntas de hace dos años que ninguna retención razonable cubriría.

3. Correlacionar, o no sirve de nada

Cuatro señales sin nada en común son cuatro sistemas separados y un incidente se investiga saltando entre pestañas y comparando marcas de tiempo a ojo.

Lo que hace falta es poco y hay que imponerlo desde el principio:

```
IDENTIFICADOR DE TRAZA en todo
  en cada registro, en cada mensaje de cola (clase 113), en cada evento
  (clase 115), en la respuesta al cliente y en el motor durable (clase 119)

NOMBRES IGUALES en las cuatro
  el servicio se llama igual en la métrica, en el registro y en la traza
  → parece obvio y casi nunca se cumple

EJEMPLARES
  el punto alto de una métrica lleva adjunta una traza de ese momento
  → es el salto de «hay latencia» a «mira esta petición»

ATRIBUTOS COMUNES
  servicio, versión, entorno, región, y el identificador del sujeto
```

```
   cuando aplique
```

Y el recorrido que esto habilita, que es el que hay que poder hacer en un incidente:

```
1. una alerta dice que el percentil 99 subió              métrica
2. un ejemplar lleva a una petición lenta concreta        traza
3. la traza señala qué tramo consume el tiempo            traza
4. los registros de ese tramo, filtrados por la traza     registro
5. y la línea de cambios dice qué se tocó hace 12 min     cambio
```

Cinco pasos, tres minutos. Sin correlación, los mismos cinco pasos son media hora y varias conversaciones.

La propagación del contexto es lo que sostiene el paso 1 a 4, y hay que cuidarla en tres sitios donde se rompe siempre:

```
al encolar          el identificador viaja en el mensaje, no en memoria
al responder y
  procesar después  el contexto asíncrono se pierde si no se traslada a mano
en los trabajos
  programados       no tienen petición de origen: hay que crear la traza
```

Y una decisión que ahorra mucho tiempo después: devolver el identificador de correlación al cliente en una cabecera de respuesta. Cuando alguien reclama, trae consigo la llave de todo lo demás.

4. Lo que cuesta y lo que se descarta

La factura de telemetría suele sorprender, y su causa es casi siempre la misma: se decide guardar todo por si acaso.

Qué dispara el coste de cada señal:

```
métricas     el número de series = combinaciones de etiquetas
             una etiqueta con identificadores de usuario multiplica por
             millones                                      clase 123
trazas       el volumen emitido; se controla con muestreo  clase 124
registros    el volumen en bytes y la retención            clase 122
cambios      irrelevante
```

Y las cuatro palancas, en orden de eficacia:

```
1. no emitir lo que nadie consulta
   → medirlo: qué paneles, alertas y consultas usan cada serie
2. retención por capas: caliente días, frío meses, agregado años
3. muestreo de trazas, con las excepciones del apartado siguiente
4. resumir en la emisión: agregar en el proceso antes de enviar
```

La primera es la que más da y la que menos se hace. Y su versión medible es una pregunta incómoda:

```
¿qué proporción de las métricas emitidas no aparece en ningún panel
ni en ninguna alerta ni en ninguna consulta de los últimos 90 días?
```

El muestreo, que es la decisión con más consecuencias de esta clase, porque lo descartado no vuelve:

```
EN LA CABEZA    se decide al empezar la petición
  + barato, y el sistema no tiene que retener nada
  − se descartan errores y casos lentos por puro azar

EN LA COLA      se decide al terminar, viendo el resultado
  + conserva el 100 % de los errores y de los lentos
  − hay que retener las trazas hasta decidir, y cuesta más
```

Y la política que funciona en la práctica:

```
siempre         errores, peticiones lentas, clientes marcados,
                y todo lo que ocurre durante un incidente
muestreado      el resto, a una tasa que se pueda pagar
nunca perdido   el conteo: la métrica cuenta el 100 %, aunque
                la traza sea de una de cada cien
```

La última línea es la que evita el error más común: muestrear no debe falsear los números. Si se muestrea 1 de 100, o se cuenta aparte con métricas, o se pondera al agregar.

Y una advertencia sobre la telemetría como dependencia: si el sistema de observabilidad se cae, la aplicación no debe caerse con él. Emisión asíncrona, con memoria acotada y descarte al llenarse.

Y la lista de comprobación de la clase:

- ■ cada señal tiene escrita la pregunta que responde
- ■ hay señal de cambio: despliegues, interruptores, configuración, escalados

- los cambios se dibujan encima de los gráficos
- el identificador de traza viaja en registros, mensajes y eventos
- los nombres de servicio coinciden en las cuatro señales
- el identificador de correlación se devuelve al cliente
- el contexto se propaga al encolar y en trabajos programados
- el muestreo conserva errores y lentos, y no falsea los conteos
- está medido qué proporción de lo emitido no consulta nadie
- la retención está por capas y lo que se guarde años está agregado
- la aplicación no se cae si la telemetría no responde

Y el cierre que enlaza con la clase siguiente: de las cuatro señales, la más cara y la que más se usa mal es el registro. Cómo se estructura, cómo se correlaciona y cuánto se conserva es la materia de la clase 122.

Ejemplo trabajado

Antes de instrumentar nada, CloudShop hace un ejercicio que cuesta una tarde: coger los veintiún problemas de las clases 109 a 119 y preguntarse, uno a uno, qué señal los habría detectado y cuánto antes. El resultado orienta todo el trabajo de la parte.

Cómo se detectaron de verdad.

por una caída total	6
por una reclamación de un cliente	4
por una auditoría o un descuadre contable	4
por la factura	2
porque a alguien le pareció raro	2
por una alerta	3
	■■
	21

Qué señal los habría detectado.

conexiones agotadas al escalar (109)	métrica de conexiones	4 min
duplicados por invisibilidad (113)	métrica de duplicados	3 sem
retardo de réplica en el proceso nocturno (109)	métrica de retardo	6 sem
conmutación lenta (109)	señal de cambio + traza	—
partición caliente (110)	métrica POR PARTICIÓN	2 días
consulta no prevista (110)	ninguna: no es un fallo	—
avalancha de caché (111)	métrica de origen	11 d
penetración de caché (111)	métrica de aciertos	2 min
caché vaciado (111)	señal de cambio	9 min
fuga entre usuarios (111)	ninguna de las cuatro	6 días
coste del lago (112)	factura, y alerta de datos leídos	1 mes
archivo de objetos pequeños (112)	alerta de coste	2 meses
41.216 en cola de fallidos (113)	métrica de la cola	67 d
tormenta de reintentos (113)	métrica de reintentos	36 min
mensajes perdidos por posición (114)	ninguna directa; conteo emitido/procesado	3 sem
rebalanceos en bucle (114)	métrica de rebalanceos	40 min
cambio de significado de campo (115)	señal de cambio	17 min
consumidor parado 19 h (115)	métrica de antigüedad	19 h
eventos no publicados (116)	métrica de la tabla de salida	6 meses
trabajo de fondo congelado (117)	conteo emitido/recibido	3 sem
trabajadores del motor caídos (119)	métrica de cola sin trabajadores	4 h

El recuento por señal.

métrica sencilla, de algo que ya existía	13 de 21
señal de cambio	3 de 21
conteo de emitidos frente a procesados	2 de 21
ninguna de las cuatro	2 de 21
factura	1 de 21

Y las tres conclusiones que salieron de la tabla:

1. Trece de veintiuno se detectaban con una métrica que no hacía falta inventar. Antigüedad de una cola, retardo de una réplica, número de conexiones, aciertos de caché, rebalanceos. Ninguna es sofisticada; simplemente nadie las miraba.

2. Dos problemas necesitaban una métrica que no existe en ningún sitio por defecto:

```
mensajes emitidos por el productor    frente a    procesados por el consumidor
trabajos lanzados                     frente a    trabajos completados
```

Es una comprobación de conservación —lo que entra tiene que salir— y detectó los dos casos que ninguna métrica de servicio veía: la posición confirmada con huecos y la instancia congelada al responder. Se instrumentó en los once consumidores.

3. Dos no los detecta ninguna señal de esta clase.

```
la fuga entre usuarios del caché (111)
  → la detectó una prueba automática, no la observabilidad
la consulta no prevista (110)
  → no era un fallo: era una petición de producto
```

Y eso es un límite honesto: hay problemas que la observabilidad no ve porque el sistema funciona correctamente según todas sus señales.

La señal de cambio, instrumentada en una semana.

Era la más barata y no existía. Se recogieron cinco orígenes:

```
despliegues (clase 102)                    ~40 al día
cambios de interruptor (clase 105)         ~6 al día
confirmaciones del repositorio de entorno  ~40 al día
escalados por encima de un umbral          ~15 al día
cambios en la puerta de entrada            ~2 al día
                                           ████████
total                                      ~103 eventos al día
```

Ciento tres eventos al día. Coste de almacenamiento: despreciable. Y el efecto medido en los tres meses siguientes:

```
incidentes en los que se preguntó
«¿alguien ha tocado algo?» por chat            todos          2 de 19
tiempo medio hasta identificar el cambio
causante                                       31 min         90 s
incidentes resueltos revirtiendo el cambio
identificado en menos de 5 min                   1 de 12       8 de 19
```

Y el caso que lo justifica solo:

```
11:42  sube la latencia del percentil 99 del catálogo
11:43  la línea de cambios muestra: interruptor «cache-v2» al 100 % a las 11:38
11:44  revertido
11:45  latencia normal
```

Tres minutos. El mismo tipo de incidente había costado treinta y cinco minutos en la clase 105, y la diferencia entera fue tener dibujados los cambios encima del gráfico.

El coste, medido antes de crecer.

```
coste mensual de telemetría al empezar              410 €
coste de cómputo                                  9.200 €
proporción                                          4,5 %

series de métricas emitidas                      41.000
series usadas en algún panel, alerta o consulta
de los últimos 90 días                            8.900
proporción sin usar                                78 %
```

Setenta y ocho por ciento emitido y no consultado nunca. Se dejaron de emitir 18.000 series de las menos usadas —conservando las de conservación del punto 2— y el coste bajó a 260 €. Ninguna consulta ni alerta se rompió en seis meses.

El muestreo, decidido con una prueba.

```
errores con traza disponible                  1 %            100 %
peticiones lentas con traza                   1 %            100 %
volumen de trazas                             1×              2,3×
coste                                         38 €            87 €
incidentes en los que faltó la traza
del caso concreto                          7 de 12         0 de 19
```

Siete de doce incidentes en los que la traza del caso investigado no se había conservado. El muestreo en la cola cuesta cincuenta euros más al mes y elimina esa categoría entera.

Estado al cabo de tres meses.

```
señales instrumentadas                        3 de 4          4 de 4
eventos de cambio al día                         0             103
series de métricas emitidas                 41.000         23.000
series sin consultar                           78 %           12 %
coste mensual de telemetría                   410 €          310 €
muestreo de trazas                  1 de 100 en cabeza    en cola
incidentes sin la traza del caso              7 de 12         0 de 19
identificador de correlación al cliente          no             sí
métricas de conservación (emitido/procesado)      0             11
tiempo hasta identificar el cambio causante   31 min          90 s
```

La lección que esta clase abre para la parte 10: el ejercicio de la tabla costó una tarde y reorientó el trabajo entero. Trece de veintiún problemas se detectaban con métricas triviales que ya se podían emitir y que nadie miraba, lo que significa que el problema no era de herramientas ni de datos: era de no haber decidido qué mirar. Y la señal que más redujo el tiempo de diagnóstico fue la más barata de las cuatro y la única que no estaba: saber qué cambió y cuándo.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-10-observability-sre-reliability/121-logs-metricas-trazas-y-eventos-como-senales/lab.py
```

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

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

Evidencia esperada

El artefacto principal es telemetría correlacionada con una pregunta operativa. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye mapa-telemetria 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
En cada incidente se pregunta por chat si alguien ha tocado algo	No existe señal de cambio: despliegues, interruptores, configuración y escalados no se registran juntos	Recoge los cambios de todos los orígenes en una serie temporal y dibújalos encima de los gráficos.
Se investiga un caso concreto y su traza no existe	Muestreo en la cabeza: se descartó por azar antes de saber que era un error	Muestreo en la cola conservando siempre errores, lentos y todo lo que ocurre durante un incidente.
La factura de telemetría crece sin que nadie sepa por qué	Se emite todo por si acaso, y el coste lo fija la emisión	Mide qué proporción de lo emitido no se consulta en 90 días y deja de emitirlo; aplica retención por capas.
Investigar exige saltar entre herramientas comparando horas a ojo	Las señales no comparten identificadores ni nombres	Propaga el identificador de traza a registros, mensajes y eventos, unifica los nombres de servicio y añade ejemplares.
Los números agregados cambian al ajustar el muestreo	Se están contando trazas muestreadas como si fueran el total	Cuenta con métricas al 100 % o pondera al agregar; muestrear no debe falsear conteos.
Se pierden datos entre dos sistemas y ninguna señal lo indica	Faltan métricas de conservación: cuánto se emitió frente a cuánto se procesó	Instrumenta el par emitido/procesado en cada frontera asíncrona.

Seguridad, ética y costo

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

Preguntas de comprobación

1. ¿Qué pregunta responde cada una de las cuatro señales y cuál no puede responder?
2. ¿Por qué la señal de cambio es la más barata y la que más reduce el tiempo de diagnóstico?
3. ¿Qué diferencia hay entre vigilar y poder preguntar, y por qué hacen falta las dos?
4. ¿Qué se pierde para siempre al muestrear en la cabeza y qué cuesta muestrear en la cola?
5. ¿Qué mide una métrica de conservación y qué tipo de fallo detecta?

Referencias

- OpenTelemetry (2025). Signals overview — métricas, trazas, registros y su correlación. <https://opentelemetry.io/docs/concepts/signals/>
- Google SRE (2025). Monitoring distributed systems — señales, alertas y el papel de los cambios. <https://sre.google/sre-book/monitoring-distributed-systems/>
- Majors, C., Fong-Jones, L. y Miranda, G. (2022). Observability Engineering, caps. 1-3 — preguntas nuevas sobre datos existentes. <https://www.oreilly.com/library/view/observability-engineering/9781492076438/>
- OpenTelemetry (2025). Sampling: head and tail — qué se descarta en cada estrategia. <https://opentelemetry.io/docs/concepts/sampling/>
- Prometheus (2025). Exemplars — salto de una métrica a una traza concreta. <https://prometheus.io/docs/prometheus/latest/featureflags/#exemplars-storage>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 10 en PDF · Manual integral

Evaluación — 121 Logs, métricas, trazas y eventos como señales

Preguntas

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

Reto verificable

Entrega mapa-telemetria 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.

122 — Logging estructurado, correlación y retención

Parte: 10 — Observabilidad, SRE y confiabilidad Nivel: avanzado · Horas estimadas: 4 Laboratorio: observability · Estado: EXECUTABLECORE

Propósito

Convertir la señal más cara y peor usada en algo consultable. La clase defiende tres cambios concretos: que un registro es un dato, no una frase, y por tanto el mensaje es constante y los valores son campos; que una petición debería producir una línea ancha con todo en vez de doce líneas dispersas; y que el registro es el sitio donde más datos personales se filtran, porque se copia a más lugares de los que nadie recuerda. Y termina con el caso que sorprende: cuando registrar es lo que tumba el sistema.

Resultados de aprendizaje

Al finalizar podrás:

1. Emitir registros estructurados con mensaje constante y valores en campos.
2. Usar los niveles para decidir quién actúa, no para expresar énfasis.
3. Sustituir líneas dispersas por una línea ancha por unidad de trabajo.
4. Impedir que credenciales y datos personales lleguen al registro.
5. Evitar que el propio registro amplifique un incidente.

Conceptos centrales

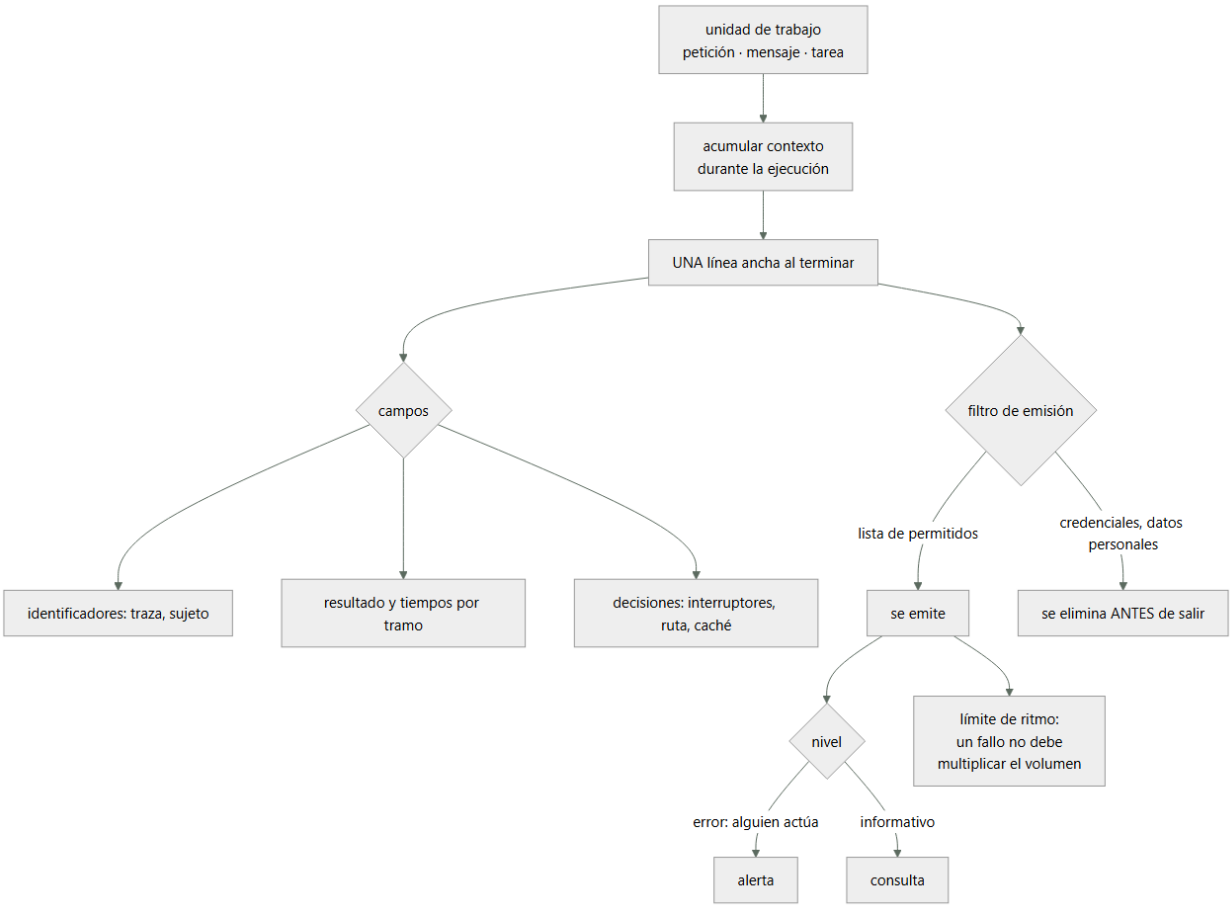
Concepto	Comprensión verificable
registro estructurado	Línea con campos nombrados y tipos, no texto con valores incrustados. Es lo que permite filtrar, agrupar y agregar.
mensaje constante	El texto no cambia entre ejecuciones; lo que varía va en campos. Permite agrupar todas las ocurrencias del mismo suceso.
línea ancha	Un solo registro por unidad de trabajo con todo lo que se sabe al terminar: identificadores, tiempos, resultados y decisiones.
nivel	Indicación de quién debe actuar. Error significa que alguien tiene que hacer algo; si nadie actúa, no era error.
depuración dirigida	Elevar el detalle solo para las peticiones marcadas, en producción, sin cambiar el nivel global.
amplificación por registro	Un fallo genera un pico de registros que consume procesador, disco o red y agrava el propio fallo.

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

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

■ Flujo de razonamiento



Desarrollo

1. Un registro es un dato

La diferencia entre estas dos líneas decide si el registro sirve para algo:

```
mal   "Error procesando el pedido 1421 del cliente 88 tras 3 intentos"
bien  {"msg":"pedido.proceso.fallido","pedido":"1421",
       "cliente":"88","intentos":3,"causa":"tiempo_agotado"}
```

La primera solo se puede buscar con expresiones frágiles. La segunda se puede filtrar, agrupar y contar:

```
¿cuántos fallos de este tipo hubo hoy?          agrupar por msg
¿afectan a un cliente concreto?                 filtrar por cliente
¿la mediana de intentos ha subido?              agregar intentos
```

Y la regla que lo hace posible: el mensaje es una constante. Si el texto lleva dentro el identificador, cada línea es un mensaje distinto y no se puede agrupar nada.

```
mal   msg: "pedido 1421 fallido"      → un millón de mensajes distintos
bien  msg: "pedido.proceso.fallido"   → un mensaje con un millón de casos
```

Y conviene tratar los nombres de mensaje como un catálogo pequeño y estable, igual que los tipos de evento de la clase 115: si cada persona inventa el suyo, no hay agrupación posible.

Los niveles, que se usan casi siempre como énfasis y deberían indicar quién actúa:

```
ERROR    alguien tiene que hacer algo
         si nadie hace nada nunca, no era un error
AVISO    el nivel más inútil que existe en la práctica:
         ni exige acción ni aporta contexto, y llena el volumen
         → o sube a error, o baja a informativo
INFO     hechos de negocio y de ciclo de vida; sirven para consultar
DEPURACIÓN  detalle para diagnosticar; apagado por defecto
```

Y la comprobación que ordena esto de golpe:

```
coge los diez mensajes de nivel error más frecuentes
```

```
¿alguien hizo algo con ellos el último mes?
  no → no son errores; bájalos
```

Es la ley 15 aplicada aquí: un nivel de error que suena mil veces al día deja de significar algo.

Y un caso concreto muy frecuente: los errores de cliente. Una petición mal formada de un cliente externo no es un error tuyo; es un hecho informativo con un código. Si se registra como error, el error deja de ser útil.

2. Una línea ancha, no doce estrechas

El patrón habitual reparte el conocimiento de una petición por todo el código:

```
"petición recibida"
"consultando caché"
"caché fallido"
"consultando base"
"base respondió en 41 ms"
"aplicando descuento"
…
"respuesta 200 en 212 ms"
```

Doce líneas, ningún sitio con la historia completa, y para reconstruirla hay que buscar por identificador y ordenar por tiempo. Y en un sistema con concurrencia, el orden puede engañar.

La alternativa es acumular contexto y emitir una sola línea al terminar:

```
{"msg":"http.peticion","traza":"a91c…","ruta":"/checkout",
 "metodo":"POST","estado":200,"duracion_ms":212,
 "cliente":"88","plan":"pro","region":"eu-west",
 "cache":"fallo","bd_ms":41,"pago_ms":118,"reintentos":0,
 "interruptores":{"pago-v2":true},"version":"2.14.1"}
```

Y lo que se gana es grande:

```
volumen              baja mucho: una línea en vez de doce
consultas nuevas     se pueden hacer sin tocar el código:
                     «¿la latencia es peor para el plan pro en eu-west
                      con el interruptor activo?»
reconstrucción       no hace falta: todo está en la misma línea
correlación          los identificadores están ahí por construcción
```

La segunda es la que convierte el registro en algo de lo que se pueden hacer preguntas nuevas, que es lo que la clase 121 llamó poder preguntar.

Y el patrón se aplica a cualquier unidad de trabajo, no solo a peticiones:

```
un mensaje consumido de una cola          clase 113
una actividad del motor durable           clase 119
un trabajo programado
una llamada saliente a un tercero
```

Y lo que sí merece líneas aparte:

```
arranque y parada del proceso
cambios de configuración detectados
errores con su traza de pila
decisiones excepcionales que quieras poder auditar
```

Y la depuración dirigida, que resuelve la tensión entre detalle y volumen:

```
nivel global: informativo
y si la petición viene marcada —cabecera, cliente concreto,
  muestreo del 0,1 %— se registra con todo el detalle
→ detalle de depuración en producción, sin cambiar el nivel global
→ y sin desplegar
```

Se controla con un interruptor de operación de la clase 105, con su fecha de revisión.

3. Lo que nunca debe salir

El registro es el sitio por donde más se filtran datos que no deberían salir, y por un motivo estructural: se copia a muchos más lugares de los que nadie recuerda.

```
del proceso al agente de recogida
del agente al sistema central
del sistema central a copias de seguridad
```

```
a un lago para análisis                              clase 112
a paneles compartidos con equipos que no deberían verlo
y a la pantalla de quien depura, y a un chat, y a una captura
```

Y la ley 11 remata: lo que entró se queda, en más sitios de los que se pueden purgar.

Lo que no debe llegar nunca:

```
contraseñas, testigos, claves, cookies de sesión
números de tarjeta completos, códigos de verificación
datos personales: nombre, correo, teléfono, dirección, documento
cuerpos completos de peticiones y respuestas
cabeceras de autorización
cadenas de conexión
```

Y el mecanismo, que tiene que ser lista de permitidos y no de prohibidos:

```
lista de prohibidos   se enumeran los campos a ocultar
                      → el campo nuevo que nadie añadió a la lista sale

lista de permitidos   solo se emiten los campos declarados
                      → lo que no está declarado no sale, y hay que
                        pedirlo a propósito
```

El segundo es más incómodo y es el único que aguanta el paso del tiempo.

Y dos precauciones más:

```
cuerpos y cabeceras     nunca completos; campos concretos, declarados
identificadores         pseudonimizar el del usuario, y guardar la
                        correspondencia en un sitio con más control
excepciones             una traza de pila puede llevar valores dentro:
                        revisar qué se serializa
librerías de terceros   registran por su cuenta; hay que comprobarlo,
                        y suele ser una sorpresa
```

La última es la que aparece en más auditorías: un cliente HTTP que registra la petición completa en nivel de depuración, y alguien sube el nivel una tarde.

Y la comprobación automática que funciona, barata de montar:

```
un escáner sobre una muestra de registros que busca patrones:
  correos, tarjetas, testigos con formato conocido, documentos
ejecutado a diario, alertando por hallazgo
→ es la misma medicina que la clase 101 aplicó al repositorio
```

Y qué hacer cuando ya ha salido, que es el procedimiento de siempre:

```
1. rotar lo que fuera una credencial
2. corregir la emisión
3. purgar donde se pueda purgar, sabiendo que no será en todos los sitios
```

4. Coste, retención y el registro como causa del incidente

El registro es la señal más cara, y su coste se decide al emitir.

```
volumen ≈ peticiones × líneas por petición × bytes por línea

1.200 pet/s × 12 líneas × 400 B ≈ 5,8 MB/s ≈ 500 GB/día
con línea ancha: 1.200 × 1 × 900 B ≈ 1,1 MB/s ≈ 93 GB/día
```

Y las palancas, en orden:

```
1. línea ancha en vez de líneas dispersas          factor 4-6
2. quitar el nivel de aviso, o subirlo o bajarlo   factor variable, grande
3. muestrear los registros de éxito                conservando todos los
                                                   errores y los lentos
4. retención por capas                             días caliente, meses frío
5. lo que haya que guardar años, agregado o en
   formato columnar                                clase 112
```

La tercera sorprende y es legítima: no hace falta guardar el registro de las mil peticiones correctas por segundo; hacen falta los conteos, que ya están en las métricas, y una muestra.

Y la retención se decide por uso, no por costumbre:

```
diagnóstico de un incidente              7-14 días caliente
investigación de un problema recurrente  30-90 días
```

```
auditoría y cumplimiento                lo que exija la norma,
                                        y en un sitio aparte con
                                        controles distintos
```

La última línea importa: mezclar registros de auditoría con los de diagnóstico obliga a aplicar a todo la retención más larga y los controles más estrictos.

Y el caso que cierra la clase: cuando registrar es lo que tumba el sistema.

```
empieza un fallo en una dependencia
cada petición fallida emite una excepción con traza de pila
el volumen de registro se multiplica por cien
→ la escritura consume procesador
→ y si es síncrona, BLOQUEA el hilo de la petición
→ el disco o la cuota se llenan
→ y el sistema de registro empieza a rechazar, con más errores
```

Lo que lo evita:

```
emisión asíncrona con memoria acotada y descarte al llenarse
  → nunca bloquear la petición por registrar
límite de ritmo por mensaje: «este mensaje, 10 por segundo como mucho,
  y luego un resumen con el conteo»
no registrar la traza de pila completa en errores esperables
y alerta sobre el propio volumen de registro, que es un síntoma
```

La penúltima línea es la más rentable: un tiempo de espera agotado no necesita cincuenta líneas de traza de pila; necesita un campo con la causa.

Y la lista de comprobación de la clase:

- ■ todos los registros son estructurados, con mensaje constante
- ■ los nombres de mensaje son un catálogo estable
- ■ el nivel de error implica que alguien actúa; se revisa cada trimestre
- ■ los errores de cliente no se registran como errores propios
- ■ hay una línea ancha por unidad de trabajo
- ■ existe depuración dirigida sin cambiar el nivel global
- ■ la emisión usa lista de permitidos, no de prohibidos
- ■ se comprueba qué registran las librerías de terceros
- ■ hay un escáner diario de datos sensibles sobre una muestra
- ■ los registros de auditoría están separados de los de diagnóstico
- ■ la emisión es asíncrona y nunca bloquea la petición
- ■ hay límite de ritmo por mensaje y alerta sobre el volumen

Y el cierre que enlaza con la clase siguiente: la señal barata de la clase 121 tiene una trampa propia que arruina presupuestos y sistemas enteros —añadir una dimensión con demasiados valores distintos—, y es la materia de la clase 123.

Ejemplo trabajado

CloudShop tiene una factura de registro que ha crecido un 40 % en seis meses y un incidente que empeoró por culpa del propio registro. El ejercicio son cuatro cambios, medidos uno a uno.

Punto de partida.

```
volumen diario                                  510 GB
coste mensual                                 3.100 €
líneas por petición, mediana                       14
proporción estructurada                          31 %
nivel de aviso                               41 % del volumen
consultas que alguien hace al día                ~25
retención                                    90 días, todo caliente
```

Cambio 1: estructurar y fijar el catálogo de mensajes.

El 69 % eran frases con valores dentro. Al estructurar aparecieron los mensajes reales:

```
mensajes distintos antes de normalizar          1,4 millones
después                                                310
```

Un millón cuatrocientos mil «mensajes distintos» eran el mismo suceso con identificadores dentro. Y con trescientos diez mensajes ya se podía agrupar:

```
los 10 mensajes de nivel error más frecuentes          74 % de los errores
de ellos, con alguna acción en el último mes              2 de 10
```

Ocho de los diez errores más frecuentes no los había mirado nadie. Cinco eran errores de cliente —peticiones mal formadas de socios— y bajaron a informativo; tres se corrigieron.

```
líneas de nivel error al día                210.000         9.400
de ellas, accionables                         ~400          ~380
proporción accionable                         0,2 %          4 %
```

Cambio 2: la línea ancha.

```
bytes por petición                          5.400            920
volumen diario                              510 GB          98 GB
coste mensual                              3.100 €          690 €
campos disponibles por petición           dispersos          31
```

Y el efecto que no se esperaba, que es el importante:

```
consultas nuevas hechas en los 2 meses siguientes        41
de ellas, imposibles antes sin cambiar el código         33
```

Ejemplos de las preguntas que pasaron a ser posibles:

```
«¿la latencia del plan pro es peor en una región concreta?»
«¿los fallos de pago se concentran en una versión del cliente móvil?»
«¿el interruptor nuevo cambia la proporción de fallos de caché?»
```

Ninguna requería instrumentación nueva: los campos ya estaban, solo que repartidos en catorce líneas.

Cambio 3: los datos personales que llevaban dos años saliendo.

El escáner diario se montó en una tarde. El primer día:

```
correos electrónicos en registros                  41.200 / día
teléfonos                                           8.900 / día
direcciones completas                               2.100 / día
números de tarjeta completos                            0
testigos de sesión                                    340 / día
```

Y el origen de los tres primeros era uno solo:

```
un cliente HTTP de una librería registraba la petición completa
en nivel de depuración
y el nivel de ese componente estaba en depuración desde una
investigación de hacía 14 meses
```

Catorce meses. Es el caso del apartado tercero, literal.

Y los 340 testigos venían de un sitio distinto: una traza de pila que serializaba el objeto de la petición.

```
mecanismo                              lista de prohibidos   lista de permitidos
campos declarados                                —                  24
datos personales detectados por el escáner   52.200 / día       0-3 / día
los 0-3 residuales                               —          campos nuevos sin declarar,
                                                            detectados el mismo día
registros purgados                               —        hasta donde se pudo
testigos rotados                                 —              todos
```

La fila de los residuales enseña la ventaja de la lista de permitidos: el campo nuevo no sale, y el escáner solo lo ve cuando alguien intenta añadirlo.

Cambio 4: el incidente que el registro amplificó.

```
16:04  el proveedor de pago empieza a agotar tiempos de espera
16:04  cada fallo emite una excepción con traza de pila: 4,2 KB por línea
16:05  volumen de registro: de 1,1 MB/s a 140 MB/s
16:06  la escritura era SÍNCRONA: los hilos de petición se bloquean
16:06  la latencia sube de 210 ms a 4,8 s en peticiones que no tocan el pago
16:11  el sistema de registro empieza a rechazar; más errores
16:31  se recupera el proveedor; el sistema tarda 9 min más en normalizarse
```

Un fallo de una dependencia que afectaba al 12 % de las peticiones degradó el 100 %, y la causa fue registrar.

```
emisión                                    síncrona     asíncrona, cola acotada
qué pasa al llenarse la cola                bloquea      descarta y cuenta
traza de pila en errores esperables            sí             no, campo causa
límite de ritmo por mensaje                    no        10/s + resumen
alerta sobre el volumen de registro            no        > 3× la mediana

ensayo del mismo fallo, después:
```

volumen de registro en el pico	140 MB/s	4,1 MB/s
latencia de peticiones no afectadas	4,8 s	230 ms
peticiones degradadas	100 %	12 %

La retención, separada por uso.

todo caliente	90 días	14 días
capa fría	no había	90 días
agregados en el lago	no había	2 años
registros de auditoría	mezclados	sistema aparte, 7 años, con sus propios controles
coste mensual total	3.100 €	520 €

Al cabo de cuatro meses.

volumen diario	510 GB	98 GB
coste mensual	3.100 €	520 €
líneas por petición	14	1
mensajes distintos	1,4 millones	310
líneas de nivel error al día	210.000	9.400
proporción de errores accionables	0,2 %	4 %
datos personales por día en registros	52.200	0-3
consultas nuevas posibles sin tocar código	—	33
degradación por amplificación de registro	100 %	12 %
retención caliente	90 días	14 días

La lección que esta clase traslada a la parte 10: el volumen bajó a la quinta parte y la utilidad subió, porque son la misma decisión. Doce líneas dispersas cuestan seis veces más que una línea ancha y responden menos preguntas. Y el hallazgo más incómodo no fue el coste: fue que el 99,8 % de lo que el sistema llamaba error no lo miraba nadie, y que un componente llevaba catorce meses escribiendo correos y teléfonos de clientes en el registro porque alguien subió un nivel para una investigación y no lo bajó.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-10-observability-sre-reliability/122-logging-estructurado-correlacion-y-retencion/lab.py
```

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

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

Evidencia esperada

El artefacto principal es telemetría correlacionada con una pregunta operativa. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

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

■ Criterio de aceptación

- [ ] lab.py termina con código 0 y genera JSON válido.
- [ ] La entrega conecta al menos tres requisitos con mecanismos verificables.
- [ ] Existe una prueba positiva y una prueba negativa con evidencia.

- [ ] Seguridad, costo y operación aparecen como decisiones, no como anexos.
- [ ] Se declara una limitación y una condición que obligaría a revisar el diseño.
- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
No se puede contar cuántas veces ha ocurrido un suceso	El mensaje lleva los valores dentro, así que cada línea es un mensaje distinto	Mensaje constante y valores en campos; trata los nombres de mensaje como un catálogo estable.
Hay cientos de miles de líneas de error al día y nadie hace nada con ellas	El nivel se usa como énfasis y los errores de cliente se registran como propios	Error significa que alguien actúa; revisa los diez más frecuentes cada trimestre y baja los que nadie atiende.
Reconstruir lo que pasó en una petición exige buscar y ordenar doce líneas	El contexto está repartido por el código	Acumula contexto y emite una línea ancha por unidad de trabajo con todo lo que se sabe al terminar.
Aparecen datos personales en los registros	Se usa lista de prohibidos y una librería registra la petición completa	Lista de permitidos, revisión de lo que registran las librerías y escáner diario sobre una muestra.
Un fallo parcial degrada el sistema entero	El registro se emite de forma síncrona y su volumen se multiplica durante el fallo	Emisión asíncrona con cola acotada, límite de ritmo por mensaje y sin trazas de pila en errores esperables.
La retención de todo es la que exige el requisito más estricto	Los registros de auditoría están mezclados con los de diagnóstico	Sepáralos en destinos distintos con sus propios controles y plazos.

■ Seguridad, ética y costo

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

■ Preguntas de comprobación

1. ¿Por qué el mensaje debe ser constante y los valores ir en campos?
2. ¿Qué debería significar el nivel de error y cómo se comprueba que se cumple?
3. ¿Qué gana una línea ancha frente a doce líneas dispersas, además de volumen?
4. ¿Por qué la lista de permitidos es la única que aguanta el paso del tiempo?
5. ¿Cómo puede el propio registro convertir un fallo parcial en una caída general?

Referencias

- Honeycomb (2025). Structured events and wide events — una línea ancha por unidad de trabajo. <https://docs.honeycomb.io/get-started/basics/observability/>
- OpenTelemetry (2025). Logs data model and correlation — campos, atributos y enlace con trazas. <https://opentelemetry.io/docs/specs/otel/logs/data-model/>
- OWASP (2025). Logging cheat sheet — qué no registrar nunca y cómo depurar en la emisión. <https://cheatsheetseries.owasp.org/cheatsheets/LoggingCheatSheet.html>
- Google SRE (2025). Practical alerting and log volume — coste del registro y su papel durante incidentes. <https://sre.google/sre-book/practical-alerting/>
- Charity Majors (2019). Logs versus structured events — por qué agrupar exige mensaje constante. <https://charity.wtf/2019/02/05/logs-vs-structured-events/>

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

Descargar: Parte 10 en PDF · Manual integral

Anterior	Índice	Siguiente
← 121 · Logs, métricas, trazas y eventos como señales	Parte 10 · Programa	123 · Métricas, cardinalidad y modelos RED y USE →

Evaluación — 122 Logging estructurado, correlación y retención

Preguntas

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

Reto verificable

Entrega contrato-logs 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.

123 — Métricas, cardinalidad y modelos RED y USE

Parte: 10 — Observabilidad, SRE y confiabilidad Nivel: avanzado · Horas estimadas: 4 Laboratorio: metrics · Estado: EXECUTABLECORE

Propósito

Emitir la señal barata sin arruinarse y sin mentir. La clase se apoya en tres hechos que se incumplen constantemente: el coste de las métricas es el producto de los valores de sus etiquetas, y basta una mala para multiplicarlo por millones; los percentiles no se promedian, así que casi todos los paneles que agregan percentiles entre instancias muestran un número que no existe; y la ausencia de un dato no es un cero, que es la forma que toma aquí la ley 13. Y da los dos modelos que dicen qué medir para no acabar con miles de series que nadie consulta.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir el tipo de métrica adecuado y saber qué agregaciones son válidas.
2. Calcular la cardinalidad antes de añadir una etiqueta.
3. Aplicar los dos modelos: servicios por peticiones y recursos por saturación.
4. Configurar intervalos de histograma que hagan útil el percentil 99.
5. Detectar la ausencia de datos, que no dispara ninguna alerta por sí sola.

Conceptos centrales

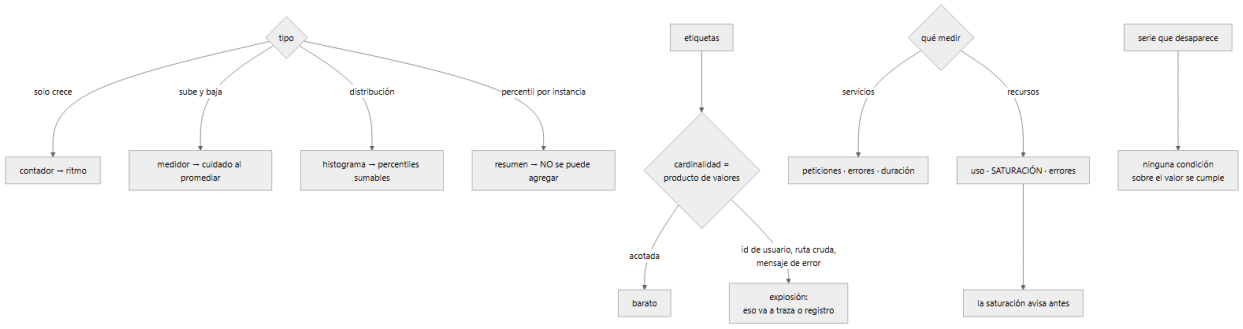
Concepto	Comprensión verificable
contador	Valor que solo crece. No se consulta su valor absoluto, sino su ritmo de crecimiento; hay que tratar los reinicios.
medidor	Valor instantáneo que sube y baja. Promediarlo entre instancias suele ocultar justo la que está mal.
histograma	Cuentas por intervalos. Es la única forma de calcular percentiles que se puedan sumar entre instancias.
cardinalidad	Número de series distintas: el producto de los valores posibles de cada etiqueta. Es lo que determina el coste.
saturación	Cuánto trabajo espera en cola. Es el indicador que avisa antes, porque sube cuando la latencia todavía es normal.
ausencia	Que una serie deje de emitirse. No es un valor bajo: es que no hay valor, y ninguna condición sobre el valor se cumple.

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

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. Tipos, y qué agregación es válida

```
CONTADOR      solo crece; se reinicia al reiniciar el proceso
  se usa      su ritmo: peticiones por segundo, errores por segundo
  no se usa   su valor absoluto
  ojo         hay que tratar el reinicio, o se ve una caída enorme

MEDIDOR       valor instantáneo: memoria, conexiones abiertas, cola
  se usa      su valor, y su máximo entre instancias
  ojo         PROMEDIAR entre instancias oculta la que está mal:
              39 instancias al 10 % y una al 100 % dan media 12 %

HISTOGRAMA    cuentas por intervalos de valor
  se usa      percentiles, y se pueden SUMAR entre instancias
  ojo         la elección de intervalos decide si el p99 sirve

RESUMEN       percentiles calculados en cada instancia
  se usa      poco
  ojo         NO se pueden agregar: el p99 de las medias de p99
              no es el p99 de nada
```

Y la regla que más paneles rompe, escrita sin rodeos:

```
LOS PERCENTILES NO SE PROMEDIAN

tres instancias con p99 = 100 ms, 100 ms y 900 ms
promedio de p99 = 367 ms
p99 real del conjunto = puede ser 850 ms
→ el número que se muestra no corresponde a ninguna petición
```

La forma correcta es sumar los intervalos del histograma de todas las instancias y calcular el percentil sobre la suma.

```
histogram_quantile(0.99, sum by (le) (rate(http_duracion_bucket[5m])))
```

Y una consecuencia práctica: si el sistema solo ofrece percentiles precalculados por instancia, no se puede saber el percentil global. Hay que cambiar a histograma.

Y dos avisos más sobre agregación:

```
sumar medidores entre instancias    a veces tiene sentido (conexiones
                                    totales) y a veces no (uso de memoria %)
dividir dos ritmos                  válido: errores/s entre peticiones/s
                                    da la proporción de error
```

2. Cardinalidad: el producto que arruina

Cada combinación distinta de valores de etiquetas es una serie, y el coste es proporcional al número de series.

```
http_peticiones{servicio, ruta, metodo, estado}

  servicios      15
  rutas          40
  métodos         5
  estados        12
                 ■■■■■
  series      36.000     manejable
```

Y ahora una etiqueta mal elegida:

```
+ cliente      190          →   6.840.000
+ usuario      2 millones   →  72.000 millones
```

Y lo que hace daño no es solo el coste: el sistema de métricas se vuelve lento o se cae, y arrastra a la vigilancia entera justo cuando hace falta.

Las etiquetas prohibidas, que son siempre las mismas:

```
identificador de usuario, de sesión, de petición, de pedido
la ruta cruda con parámetros:  /pedidos/1421  →  /pedidos/{id}
el mensaje de error completo
direcciones IP
marcas de tiempo
nombres de recurso efímeros: pods, contenedores, instancias
```

La última es traicionera: con despliegues frecuentes, una etiqueta con el nombre del pod crea series nuevas cada día, y aunque dejen de emitirse siguen ocupando durante la retención.

Y la regla que ordena la decisión, y que enlaza con la clase 121:

```
si una dimensión tiene muchos valores posibles y hace falta
para investigar, va en la TRAZA o en la LÍNEA ANCHA, no en la métrica
```

Y dos técnicas para conservar utilidad sin cardinalidad:

```
AGRUPAR VALORES
  estado → 2xx, 4xx, 5xx  en vez de los 60 códigos
  cliente → plan del cliente (4 valores) en vez del cliente (190)

MÉTRICAS APARTE PARA LOS POCOS QUE IMPORTAN
  los 10 clientes mayores con etiqueta propia, el resto agrupado en «otros»
```

Y el control que evita las sorpresas, porque una etiqueta nueva puede multiplicar la factura sin que nadie lo note:

```
límite de series por métrica, con rechazo y alerta al superarlo
revisión de las métricas de mayor cardinalidad, cada mes
y comprobación en la canalización: una etiqueta nueva se justifica
```

3. Qué medir: dos modelos

Sin un modelo se acaba con miles de series que nadie consulta —el 78 % del ejemplo de la clase 121—. Los dos que funcionan responden a preguntas distintas.

```
PARA SERVICIOS QUE ATIENDEN PETICIONES
  ritmo       peticiones por segundo
  errores     proporción de fallos
  duración    distribución, con histograma
  → describe lo que EXPERIMENTA quien llama

PARA RECURSOS
  uso         qué proporción del tiempo está ocupado
  saturación  cuánto trabajo espera en cola
  errores     fallos del propio recurso
  → describe lo que SUFRE el recurso
```

Y hacen falta los dos porque un servicio puede ir bien mientras un recurso se acerca al límite:

```
latencia normal, errores cero
y la cola de conexiones al 92 %
→ el primer modelo dice que todo va bien
→ el segundo dice que quedan minutos
```

La saturación es el indicador que avisa antes, y es el que menos se instrumenta. Los sitios concretos de este programa:

```
conexiones del agrupador en uso frente al máximo      clase 109
profundidad y antigüedad de una cola                  clase 113
retraso del consumidor de un registro                 clase 114
concurrencia usada frente al límite                   clase 117
hilos ocupados, memoria antes de recolectar
cola de escritura a disco
```

Y una advertencia sobre el uso de procesador como indicador principal: es de los peores. Un sistema puede estar bloqueado esperando entrada/salida con el procesador al 8 % y otro al 95 % sirviendo perfectamente. La saturación dice más.

Los intervalos del histograma, que deciden si el percentil sirve:

```
intervalos por defecto: 5, 10, 25, 50, 100, 250, 500, 1000, 2500 ms
si tu servicio responde en 3 ms, todo cae en el primero
  → el p99 no distingue nada
si responde en 8 s, todo cae en el último
  → el p99 es «más de 2,5 s», y no dice cuánto
```

La regla: intervalos alrededor de lo que de verdad ocurre, y uno colocado exactamente en el valor que importa:

```
si el objetivo es «el 99 % por debajo de 300 ms»
→ que haya un intervalo justo en 300 ms
→ así la proporción por debajo del objetivo es una división exacta
   y no una interpolación
```

Eso conecta directamente con la clase 126, y es la razón por la que conviene decidir el objetivo antes de fijar los intervalos.

Y un matiz sobre lo que se mide: la duración desde dónde. El tiempo del servidor no incluye la cola de entrada ni la red, así que puede ser excelente mientras el usuario espera segundos. Medir también en el borde, donde el número incluye lo que el usuario sufre.

4. Lo que no está, y lo que cuesta

La ausencia. Una alerta que dice «avisa si los errores superan el 5 %» no se dispara si la métrica deja de emitirse. No hay errores, no hay peticiones, no hay nada: la condición sobre el valor simplemente no se evalúa.

```
el servicio se cae del todo         → no emite
el recolector pierde el objetivo    → no emite
cambia una etiqueta                 → la serie vieja muere y nace otra
se renombra la métrica              → igual
```

Y el resultado es el peor posible: el sistema está caído y el panel está verde. Es la ley 13, otra vez, ahora en las métricas.

Lo que lo detecta:

```
alerta explícita de ausencia
  absent(up{trabajo="pedidos"}) == 1
  o «no ha llegado ningún dato de esta serie en 5 minutos»

vigilancia del propio sistema de recogida
  objetivos esperados frente a objetivos recogidos

latido: una métrica que siempre vale 1 y cuya ausencia es la señal
```

Y el caso emparentado: una serie que desaparece al cambiar una etiqueta. Las consultas y alertas dejan de encontrarla y nadie da error. Por eso los nombres de métrica y etiqueta merecen el mismo trato que un contrato: catálogo, revisión y cambio compatible.

El coste, con las mismas palancas que la clase 122 y una propia:

```
1. reducir cardinalidad         es la que más da, con diferencia
2. dejar de emitir lo que nadie consulta
3. reglas de agregación previa  precalcular las consultas caras y
                                guardar el resultado como serie nueva
4. reducir resolución con la edad
                                cada 15 s los últimos días,
                                cada 5 min los meses siguientes
5. intervalo de recogida        pasar de 10 s a 30 s divide por tres,
                                y hay que comprobar qué se pierde
```

La tercera es además de rendimiento: una consulta que tarda cuarenta segundos en un panel también tarda cuarenta segundos al evaluar una alerta, y eso retrasa la detección.

Y la lista de comprobación de la clase:

- ninguna etiqueta lleva identificadores ni rutas sin normalizar
- hay límite de series por métrica, con alerta al acercarse
- los percentiles se calculan sobre histogramas sumados, no promediados
- los medidores no se promedian entre instancias sin pensarlo
- cada servicio tiene ritmo, errores y duración
- cada recurso tiene uso, saturación y errores
- la saturación está instrumentada en agrupadores, colas y concurrencia

- los intervalos del histograma rodean los valores reales
- hay un intervalo colocado en el objetivo de latencia
- la duración se mide también en el borde, no solo en el servidor
- hay alertas de ausencia, no solo de umbral
- los nombres de métrica y etiqueta se tratan como contrato
- las consultas caras están precalculadas

Y el cierre que enlaza con la clase siguiente: las métricas dicen que algo va mal y no dicen dónde. Seguir una petición por los quince servicios que la atienden, y saber en cuál se fue el tiempo, es la materia de la clase 124.

Ejemplo trabajado

CloudShop tiene un sistema de métricas que cuesta más que el registro, un panel de latencia que muestra números que nadie sufre y una alerta que no se disparó durante una caída de once minutos. Los tres problemas se resuelven en orden.

Problema 1: la explosión de cardinalidad.

```
series activas                                  8,4 millones
coste mensual                                       2.900 €
consultas que tardan más de 10 s                  11 de 34 paneles
caídas del sistema de métricas en 6 meses                2
```

Y las tres métricas responsables:

```
http_peticiones{…, ruta}   con la ruta sin normalizar   4,1 M
  /pedidos/1421, /pedidos/1422, …
cola_mensajes{…, pod}      nombre del pod               2,6 M
  con 40 despliegues al día durante la retención
errores{…, mensaje}        mensaje de error completo    1,1 M
```

Y las tres correcciones:

```
ruta                                  /pedidos/1421     /pedidos/{id}
  series de esa métrica                   4,1 M            18.000
pod como etiqueta                         sí               no (va en la línea
                                                           ancha)
  series de esa métrica                   2,6 M             1.400
mensaje de error                        texto completo    código de causa
  series de esa métrica                   1,1 M               840
```

```
series activas                           8,4 millones      310.000
coste mensual                               2.900 €          280 €
consultas de más de 10 s                   11 de 34         0 de 34
tiempo de evaluación de alertas, p95          41 s            1,2 s
```

La última fila es la consecuencia que nadie esperaba: las alertas tardaban cuarenta y un segundos en evaluarse, así que la detección llegaba con ese retraso añadido.

Y el control que evita la reincidencia:

```
límite de 50.000 series por métrica, con rechazo y alerta
rechazos en 6 meses                                    3
  → los 3 eran etiquetas nuevas con identificadores dentro
```

Problema 2: el percentil que no existía.

El panel principal mostraba la latencia del percentil 99 como media de las instancias:

```
avg(latencia_p99_por_instancia)
```

Durante un incidente, la discrepancia era enorme:

```
lo que mostraba el panel                        340 ms
lo que reclamaban los clientes                  «tarda 4 segundos»
p99 real, calculado sobre histogramas sumados   3.900 ms
```

La causa era la del apartado primero: dos de treinta y cuatro instancias tenían un problema y su percentil quedaba diluido en la media.

```
p99 mostrado en el incidente               340 ms            3.900 ms
diferencia con lo que sufría el cliente    ×11,5                ×1
incidentes en los que el panel dijo
«todo bien» mientras había quejas        4 en 6 meses          0
```

Y el cambio de tipo de métrica fue lo que lo permitió: el sistema emitía resúmenes, que no se pueden agregar. Hubo que pasar a histogramas en los quince servicios.

Problema 3: los intervalos que ocultaban el problema.

Tras pasar a histogramas, el percentil 99 seguía siendo poco informativo:

```
intervalos por defecto     5, 10, 25, 50, 100, 250, 500, 1000, 2500 ms
latencia real del servicio de catálogo   entre 2 y 7 ms
→ el 98 % de las peticiones caía en el primer intervalo
→ el p99 se interpolaba y daba 4,9 ms siempre, pasara lo que pasara
```

```
intervalos                              5…2500 ms       1, 2, 3, 5, 8, 12,
                                                        20, 50, 150, 300, 1000
p99 durante una degradación real          4,9 ms          18 ms
degradaciones detectadas por el panel        0            7 en 4 meses
```

Y el intervalo colocado en el objetivo, anticipando la clase 126:

```
objetivo de latencia del catálogo: 99 % por debajo de 300 ms
→ hay un intervalo exactamente en 300
→ la proporción por debajo del objetivo es una división, no una estimación
```

Problema 4: la caída con el panel en verde.

```
03:12  el proceso de pedidos falla al arrancar tras un despliegue
03:12  deja de emitir métricas
03:12  la alerta «errores > 5 %» no se evalúa: no hay errores ni peticiones
03:12  el panel muestra las últimas series planas y luego nada
03:23  un cliente reclama
duración                                                11 min
alertas disparadas                                          0
```

Es la ley 13 en métricas: ninguna condición sobre un valor se cumple cuando no hay valor.

```
alertas de umbral                               41              41
alertas de ausencia                              0              15
vigilancia de objetivos esperados vs recogidos  no              sí

ensayo: parar un servicio a propósito
tiempo hasta la alerta                    no llegaba        70 s
```

Y el ensayo se añadió a la rutina trimestral, con la misma lógica de la clase 102: parar un servicio a propósito y comprobar que alguien se entera.

El modelo aplicado, y lo que faltaba.

Al revisar los quince servicios con los dos modelos:

```
ritmo de peticiones                      15 de 15          0
proporción de errores                    15 de 15          0
distribución de duración                  6 de 15          9
uso de recursos                          15 de 15          0
SATURACIÓN                                2 de 15         13
errores del recurso                       9 de 15          6
```

Trece de quince servicios no medían saturación, que es el indicador que avisa antes. Se instrumentaron los seis sitios del apartado tercero, y en cuatro meses:

```
incidentes anticipados por saturación antes de que subiera la latencia   6
  agrupador de conexiones al 90 %                                        2
  antigüedad de cola creciendo                                           2
  concurrencia de función cerca del límite                               1
  hilos ocupados                                                         1
de ellos, evitados por completo                                          4
```

A los cuatro meses.

```
series activas                           8,4 millones      310.000
coste mensual de métricas                   2.900 €          280 €
tiempo de evaluación de alertas, p95           41 s          1,2 s
percentiles calculados correctamente           no             sí
incidentes con el panel en verde y quejas   4 / 6 meses         0
servicios con distribución de duración       6 de 15        15 de 15
servicios con saturación instrumentada       2 de 15        15 de 15
alertas de ausencia                             0              15
caídas detectadas solo por clientes          2 / 6 meses         0
```

```
incidentes anticipados por saturación            0          6 / 4 meses
```

La lección que esta clase traslada a la parte 10: el coste bajó de 2.900 € a 280 € quitando tres etiquetas, y a la vez el sistema pasó a decir la verdad, porque las consultas dejaron de tardar cuarenta segundos. Y de los cuatro problemas, dos eran errores de aritmética que llevaban años en producción: un percentil promediado que ocultaba un factor once, y unos intervalos por defecto que hacían que el percentil 99 diera siempre el mismo número.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-10-observability-sre-reliability/123-metricas-cardinalidad-y-modelos-red-y-use/lab.py
```

El laboratorio selecciona el motor de práctica metrics 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 catalogo-metricas en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es métricas definidas, consultables y vinculadas a una decisión. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye catalogo-metricas 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 de métricas es lento o se cae	Explosión de cardinalidad por etiquetas con identificadores, rutas sin normalizar o nombres de pod	Normaliza rutas, saca las dimensiones de muchos valores a traza o línea ancha y pon límite de series por métrica.
El panel muestra buena latencia mientras hay quejas	Se está promediando el percentil de cada instancia, y eso no es un percentil	Emite histogramas y calcula el percentil sobre la suma de intervalos de todas las instancias.
El percentil 99 devuelve siempre el mismo valor	Los intervalos del histograma no rodean los valores reales	Ajusta los intervalos a la latencia real y coloca uno exactamente en el objetivo.
Un servicio cae y no se dispara ninguna alerta	Ley 13: sin datos, ninguna condición sobre el valor se evalúa	Añade alertas de ausencia y vigila objetivos esperados frente a recogidos; ensaya parando un servicio.
La latencia y los errores están bien y el sistema se cae poco después	No se mide saturación, que es el indicador que avisa antes	Instrumenta cola, agrupadores, concurrencia e hilos ocupados frente a sus máximos.
Las alertas tardan casi un minuto en evaluarse	Las consultas son caras por cardinalidad o por falta de agregación previa	Reduce cardinalidad y precalcula las consultas caras como series nuevas.

■ 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 percentiles no se pueden promediar entre instancias?
2. ¿Cómo se calcula la cardinalidad y qué etiquetas la disparan siempre?
3. ¿Qué mide cada uno de los dos modelos y por qué hacen falta los dos?
4. ¿Por qué la saturación avisa antes que la latencia?
5. ¿Qué ocurre con una alerta de umbral cuando la métrica deja de emitirse?

Referencias

- Prometheus (2025). Metric types and naming — contadores, medidores, histogramas y resúmenes. <https://prometheus.io/docs/concepts/metrictypes/>
- Prometheus (2025). Instrumentation best practices: labels and cardinality — coste del producto de etiquetas. <https://prometheus.io/docs/practices/instrumentation/>
- Wilkie, T. (2018). The RED method — ritmo, errores y duración para servicios. <https://grafana.com/blog/2018/08/02/the-red-method-how-to-instrument-your-services/>
- Gregg, B. (2013). The USE method — uso, saturación y errores para recursos. <https://www.brendangregg.com/usemethod.html>
- Google SRE (2025). The four golden signals — latencia, tráfico, errores y saturación. <https://sre.google/sre-book/monitoring-distributed-systems/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 10 en PDF · Manual integral

Anterior	Índice	Siguiente
← 122 · Logging estructurado, correlación y retención	Parte 10 · Programa	124 · Tracing distribuido y OpenTelemetry →

Evaluación — 123 Métricas, cardinalidad y modelos RED y USE

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 catalogo-metricas 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.

124 — Tracing distribuido y OpenTelemetry

Parte: 10 — Observabilidad, SRE y confiabilidad Nivel: avanzado · Horas estimadas: 4 Laboratorio: observability · Estado: EXECUTABLECORE

Propósito

Seguir una petición por los quince servicios que la atienden y saber en cuál se fue el tiempo. La clase enseña lo que ninguna métrica puede dar —la forma del grafo de llamadas de un caso concreto— y se centra en las dos cosas que deciden si sirve: propagar el contexto por los sitios donde siempre se rompe, que son los asíncronos, y saber leer las cuatro formas que adopta una traza cuando algo va mal. Y sitúa el sitio correcto para las dimensiones de mucha cardinalidad que la clase 123 prohibió en las métricas.

Resultados de aprendizaje

Al finalizar podrás:

1. Describir una traza como árbol de tramos, con padres y enlaces.
2. Propagar el contexto en HTTP, en colas y en trabajos programados.
3. Diagnosticar leyendo la forma de la traza, no solo su duración.
4. Añadir atributos de negocio que conviertan la traza en respuesta a preguntas.
5. Decidir el muestreo de forma coherente entre todos los servicios.

Conceptos centrales

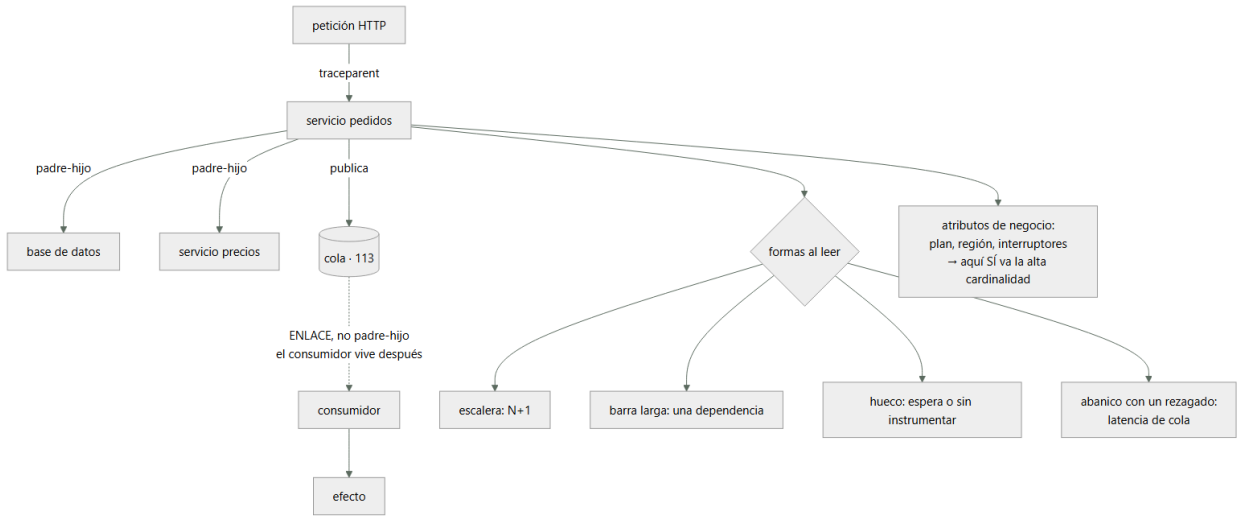
Concepto	Comprensión verificable
tramo	Operación con inicio, duración, estado y atributos. Es la unidad de una traza.
traza	Árbol de tramos que comparten un identificador. Describe el recorrido completo de una unidad de trabajo.
propagación de contexto	Llevar el identificador de traza y la decisión de muestreo de un proceso al siguiente. Sin ella, la traza se parte.
enlace	Relación entre tramos que no es de padre a hijo. Es lo que corresponde al cruzar una cola, donde el consumidor puede vivir mucho después.
hueco	Tiempo dentro de un tramo que ningún tramo hijo explica. Casi siempre es espera en cola, contención o algo sin instrumentar.
muestreo coherente	Que todos los servicios tomen la misma decisión sobre la misma traza. Si no, quedan trazas incompletas que engañan.

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

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 y qué añade

Una traza es un árbol. Cada nodo es un tramo con lo mismo:

```
identificador de traza      común a todo el árbol
identificador del tramo     propio
identificador del padre     de quién cuelga
nombre                      la operación: GET /checkout, SELECT pedidos
inicio y duración
estado                      correcto o error, con causa
atributos                   todo lo que ayude a filtrar
```

Y lo que aporta frente a las otras señales:

```
la métrica dice   «el percentil 99 subió a 3,9 s»
la traza dice     «en esta petición, 3,4 s se fueron en 340 llamadas
                   a la base, todas iguales»
```

La segunda es accionable y la primera no. Y hay tres cosas que solo la traza da:

```
la FORMA del grafo de llamadas
  → aparecen dependencias que nadie sabía que existían
dónde se fue el tiempo dentro de una petición
cómo se comporta la cola de la distribución
  → la petición lenta concreta, no el agregado
```

Y lo que no da:

```
frecuencia, si está muestreada      → eso lo dan las métricas
lo que no se instrumentó            → aparece como hueco
tendencias largas                   → la retención de trazas es corta
```

Y una decisión que enlaza con la clase 123: los atributos de un tramo pueden tener cardinalidad alta sin arruinar nada. El identificador de cliente, de pedido, de sesión, la ruta concreta, el mensaje de error: aquí sí.

```
métrica      dimensiones pocas y acotadas      clase 123
traza        dimensiones muchas y libres       esta clase
línea ancha  igual que la traza                clase 122
```

Y los atributos que más valen no son los técnicos, sino los de negocio:

```
plan del cliente, país, canal, tipo de pago
versión de la aplicación cliente
estado de los interruptores evaluados            clase 105
número de artículos, importe, si venía de campaña
```

Eso convierte la traza en algo que responde preguntas de producto: «¿la lentitud afecta solo a los pedidos con más de veinte líneas?»

2. Propagar el contexto, que es donde se rompe

La traza se mantiene porque cada llamada lleva el contexto. En HTTP es una cabecera estándar:

```
traceparent: 00-a91c4e2b…-b7f2c1…-01
             ■  ■           ■        ■ decisión de muestreo
             ■  ■           ■ tramo padre
             ■  ■ identificador de traza
             ■ versión
```

Y hay tres sitios donde se rompe siempre, los tres ya nombrados en la clase 121 y aquí con su mecanismo:

```
1. AL ENCOLAR
   el contexto vive en la petición y el mensaje viaja solo
   → el contexto se pone en los ATRIBUTOS del mensaje, no en memoria
   → y en el consumidor se restaura al leerlo

2. EN EJECUCIÓN ASÍNCRONA DENTRO DEL PROCESO
   grupos de hilos, tareas planificadas, devoluciones de llamada
   → el contexto se guarda en almacenamiento local del hilo
   → y al saltar de hilo hay que trasladarlo explícitamente

3. EN TRABAJOS PROGRAMADOS
   no hay petición de origen
   → hay que CREAR una traza nueva, con un atributo que diga
     que es programado, y no dejarlos sin traza
```

Y una decisión de modelado en el primer caso que se equivoca a menudo:

```
mal   el tramo del consumidor cuelga como HIJO del productor
      → el padre ya terminó hace 40 minutos
      → la traza dice que la petición duró 40 minutos, y es falso

bien  el tramo del consumidor es raíz de su propia traza,
      con un ENLACE al tramo que publicó
      → y se puede navegar de uno a otro
```

Y con eso hay dos duraciones distintas, y las dos importan:

```
lo que esperó el usuario           la traza síncrona: 140 ms
el proceso completo hasta el efecto  del enlace: 2,6 s
```

Y el hueco, que es la señal más informativa de una traza:

```
tramo padre: 900 ms
tramos hijos: 40 ms + 30 ms
hueco: 830 ms sin explicar
```

Las causas, por frecuencia:

```
espera en cola antes de que un hilo atendiera la petición
contención por un bloqueo
pausa de recolección de memoria
código sin instrumentar: serialización, cifrado, cálculo
tiempo de red no atribuido
```

El primero es el más frecuente y el que más se confunde con «el servicio es lento»: el servicio no tardó; la petición estuvo esperando a entrar. Se distingue instrumentando la espera en cola como tramo propio.

3. Leer una traza: cuatro formas

Diagnosticar con trazas es reconocer formas. Estas cuatro cubren la mayoría de los casos:

```
1. ESCALERA — muchos tramos iguales y seguidos
   ■ SELECT producto  4 ms
   ■ SELECT producto  4 ms
   ■ SELECT producto  4 ms   … ×340
   diagnóstico   una consulta por elemento de una lista
   corrección    traer todo de una vez, o precargar
   nota          cada llamada es rápida; el problema es el número,
                 y por eso ninguna métrica de latencia de consulta
                 lo detecta

2. BARRA LARGA — un tramo consume casi todo
   diagnóstico   una dependencia lenta; mirar SU traza
   corrección    ahí, no aquí
   nota          si la barra no tiene hijos, o no está instrumentada
                 o es tiempo real de cálculo

3. HUECO — el padre dura mucho más que la suma de sus hijos
```

```
   diagnóstico   espera, contención o código sin instrumentar
   corrección    instrumentar la espera antes de suponer nada

4. ABANICO CON REZAGADO — diez llamadas en paralelo y una tarda
   diagnóstico   latencia de cola: se espera al más lento
   corrección    plazo por llamada, respuesta parcial, o pedir a dos
                 y quedarse con el primero              clase 130
   nota          la media de las diez es excelente y el usuario sufre
                 la peor
```

Y dos comprobaciones que conviene hacer siempre al mirar una traza:

```
¿hay tramos que se repiten con el mismo nombre?         forma 1
¿la suma de los hijos explica al padre?                 forma 3
¿hay tramos en paralelo con duraciones muy distintas?   forma 4
¿hay errores marcados en tramos que no fallaron
  la petición?                                          reintentos ocultos
```

La última descubre reintentos que funcionan y que están costando latencia sin que nadie los cuente.

Instrumentación automática y manual, y qué aporta cada una:

```
AUTOMÁTICA   marcos web, clientes HTTP, controladores de base,
             clientes de cola
             → da el esqueleto por casi nada de trabajo
             → y no sabe nada de tu negocio

MANUAL       tramos para operaciones propias que valga la pena medir
             y ATRIBUTOS de negocio en el tramo raíz
             → es lo que convierte la traza en respuesta a preguntas
```

Y una recomendación de dosis: pocos tramos manuales y muchos atributos. Un árbol con doscientos tramos por petición es difícil de leer y caro; el mismo árbol con veinte tramos y treinta atributos responde más.

4. Muestreo coherente, recolector y coste

El muestreo tiene que ser coherente: si el servicio A decide conservar la traza y el B no, queda un árbol con ramas cortadas que induce a conclusiones falsas.

```
la decisión se toma UNA vez y viaja en el contexto
→ es el último bit de la cabecera estándar
→ y todos los servicios la respetan
```

Y las dos estrategias, con lo que la clase 121 ya estableció:

```
EN LA CABEZA   decide el primero; barato; descarta errores por azar
EN LA COLA     decide al terminar, viendo el resultado completo
               → conserva el 100 % de errores y lentos
               → exige retener las trazas hasta decidir
```

Y un detalle del muestreo en la cola que hay que resolver: hay que reunir todos los tramos de una traza antes de decidir, y llegan de servicios distintos en momentos distintos. Eso lo hace un componente intermedio, no la aplicación.

El recolector, que es ese componente y hace más cosas de las que parece:

```
recibe de todos los servicios en un formato común
aplica el muestreo en la cola
depura atributos sensibles antes de enviar        clase 122
añade atributos comunes: entorno, región, versión
reparte a varios destinos a la vez
amortigua picos y reintenta
```

Y las dos ventajas que más se notan:

```
cambiar de proveedor no toca el código de los servicios
y la depuración de datos sensibles ocurre en UN sitio, no en quince
```

Y una advertencia: el recolector es una dependencia más. Si se cae, la aplicación no debe caerse; emisión asíncrona, memoria acotada y descarte, como en la clase 122.

El coste, que se comporta distinto que en métricas:

```
lo caro     el número de tramos y su retención
lo barato   los atributos, comparados con las etiquetas de métrica

palancas
```

```
muestrear, con las excepciones de siempre
reducir tramos por petición: no instrumentar lo trivial
retención corta: 7-15 días suele bastar
y guardar en el lago lo que haga falta conservar más
```

Y un uso de las trazas que se aprovecha poco: medir el grafo de dependencias real. Agregando trazas se obtiene quién llama a quién y con qué frecuencia, que suele contradecir el diagrama de arquitectura.

Y la lista de comprobación de la clase:

- el contexto se propaga en HTTP, en colas y entre hilos
- los trabajos programados crean su propia traza
- el cruce de una cola se modela como enlace, no como padre-hijo
- la espera en cola de entrada está instrumentada como tramo
- hay atributos de negocio en el tramo raíz
- los identificadores de alta cardinalidad van aquí, no en métricas
- la decisión de muestreo se toma una vez y la respetan todos
- el muestreo conserva errores y lentos
- hay recolector, y la depuración de datos sensibles ocurre en él
- la aplicación no se cae si el recolector no responde
- se revisa el grafo de dependencias real frente al que se cree tener

Y el cierre que enlaza con la clase siguiente: con las cuatro señales instrumentadas y correlacionadas, queda decidir qué se mira y qué despierta a alguien. Y ahí el fallo no es medir poco, sino producir tantas señales que dejan de serlo, que es la materia de la clase 125.

Ejemplo trabajado

CloudShop instrumenta trazas en los quince servicios. Lo primero que aparece no es una lentitud: es que el grafo de llamadas no se parece al diagrama de arquitectura. Después vienen cuatro diagnósticos, uno por cada forma.

El grafo real frente al que se creía tener.

```
dependencias en el diagrama de arquitectura                    23
dependencias observadas en las trazas                          41
dependencias que nadie sabía que existían                      18
de ellas, servicio de catálogo → servicio de pagos               sí
  motivo   una comprobación de fraude añadida hacía 8 meses
  efecto   el catálogo dejaba de responder si pagos estaba mal
```

Dieciocho dependencias no documentadas, y una de ellas explicaba por qué una caída del servicio de pagos tumbaba también el catálogo, algo que llevaba meses sin explicación.

Forma 1: la escalera de 340 peldaños.

```
GET /pedidos/{id}/detalle                          3.410 ms
  ■ SELECT pedido                                      6 ms
  ■ SELECT linea (×340)                            3.380 ms
  ■ render                                            24 ms
```

Trescientas cuarenta consultas de 9,9 ms de media. Ninguna métrica lo detectaba: la latencia de cada consulta era excelente y la del endpoint se diluía en el agregado porque solo afectaba a pedidos grandes.

```
consultas por petición, mediana              8              4
consultas por petición, p99                340              5
latencia p99 del endpoint              3.410 ms         112 ms
endpoints con el mismo patrón,
encontrados al buscar la forma               —              6
```

Seis endpoints más con la misma escalera, encontrados buscando la forma en vez de esperar a que alguien se quejara.

Forma 3: el hueco de 830 ms.

```
POST /checkout                                       960 ms
  ■ validar                                            8 ms
  ■ SELECT cliente                                    11 ms
  ■ llamada a precios                                 41 ms
  ■ … suma de hijos: 60 ms
hueco sin explicar                                   900 ms
```

La primera hipótesis fue recolección de memoria. La instrumentación de la espera en cola de entrada dio la respuesta:

```
tiempo esperando un hilo libre                       880 ms
```

```
tiempo de proceso real                                  80 ms
hilos configurados                                        16
concurrencia entrante en el pico                         140
```

El servicio no era lento: estaba lleno. Y la corrección no fue optimizar código:

```
hilos                                      16             64
espera en cola, p99                      880 ms          12 ms
latencia p99 del endpoint                960 ms         104 ms
uso de procesador                          21 %           68 %
```

Y se instrumentó la espera en cola como tramo propio en los quince servicios: apareció en otros cuatro.

Forma 4: el abanico con rezagado.

```
GET /inicio                                          1.240 ms
  ■ recomendaciones     ■■                              78 ms
  ■ promociones         ■                               41 ms
  ■ catálogo            ■■                              96 ms
  ■ valoraciones        ■■■■■■■■■■■■■■■■■■■■■■        1.190 ms   ← rezagado
  ■ carrito             ■                               33 ms
```

Cinco llamadas en paralelo, media 288 ms, y el usuario espera 1.190 ms.

```
plazo para valoraciones                  sin plazo        150 ms
qué pasa al vencer                          —        se sirve sin valoraciones
latencia p99 de la portada               1.240 ms        180 ms
peticiones servidas sin valoraciones        —             4,1 %
quejas por falta de valoraciones            —              0
```

La penúltima fila es el coste asumido conscientemente, y la última dice que valió la pena. Es la respuesta parcial que la clase 130 desarrollará.

Forma 2, y los reintentos ocultos.

Al buscar tramos con error dentro de peticiones que terminaron bien:

```
peticiones correctas que contenían al menos un tramo fallido    11 %
latencia media de esas peticiones                              840 ms
latencia media del resto                                       190 ms
causa      reintentos automáticos contra un servicio con el 4 %
           de errores intermitentes
```

Once por ciento de las peticiones pagaban un reintento, y ninguna métrica de errores lo mostraba, porque el resultado final era correcto. Corregido el servicio intermitente, la latencia media global bajó un 18 %.

La propagación, que faltaba en dos de los tres sitios.

```
HTTP                                     propagado       propagado
colas                                    NO propagado    en atributos
                                                         del mensaje
trabajos programados                     sin traza       traza propia
hilos de fondo                           se perdía       trasladado

trazas completas de extremo a extremo       31 %            96 %
trazas que terminaban en el productor       69 %             0 %
```

Y la corrección del modelado: el consumidor pasó a ser raíz con enlace, en vez de hijo.

```
antes    la traza decía que /checkout duraba 2,6 s
         → porque incluía el consumo asíncrono
después  /checkout dura 140 ms, y un enlace lleva al proceso de 2,6 s
```

El recolector, y el cambio de proveedor.

```
destinos de telemetría                      1 (directo)     2 (por recolector)
depuración de datos sensibles            en 15 servicios    en 1 sitio
cambio de proveedor de trazas           tocar 15 servicios  cambiar 6 líneas
                                                            del recolector
tiempo del cambio de proveedor, real        —               1 tarde
muestreo                                 1 de 100 en cabeza  en cola
```

A los cinco meses.

```
dependencias conocidas                      23              41
trazas completas de extremo a extremo       31 %            96 %
endpoints con consulta por elemento          7               0
servicios con espera en cola instrumentada   0              15
```

latencia p99 de la portada	1.240 ms	180 ms
latencia p99 del detalle de pedido	3.410 ms	112 ms
peticiones con reintentos ocultos	11 %	0,4 %
tiempo medio de diagnóstico de una lentitud	2 h 20	9 min
coste mensual de trazas	—	190 €

La lección que esta clase traslada a la parte 10: las cuatro correcciones —agrupar consultas, subir hilos, poner un plazo y arreglar un servicio intermitente— no las habría encontrado ninguna métrica, porque en los cuatro casos las métricas de cada componente estaban bien. El problema estaba en la relación entre ellos, y eso solo se ve mirando un caso completo. Y el hallazgo con más consecuencias no fue de rendimiento: fue descubrir que el sistema tenía dieciocho dependencias que no estaban en ningún diagrama.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-10-observability-sre-reliability/124-tracing-distribuido-y-opentelemetry/lab.py
```

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

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

Evidencia esperada

El artefacto principal es telemetría correlacionada con una pregunta operativa. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

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

■ Criterio de aceptación

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

Errores frecuentes

Síntoma	Causa probable	Corrección
Las trazas terminan en el servicio que publica y no continúan	El contexto no viaja en el mensaje de la cola	Pon el contexto en los atributos del mensaje y restáuralo en el consumidor.
Una petición de 140 ms aparece como si durara 40 minutos	El tramo del consumidor cuelga como hijo del productor	Modela el cruce de la cola como enlace y haz del consumidor la raíz de su propia traza.
El tramo padre dura mucho más que la suma de sus hijos	Hay espera, contención o código sin instrumentar	Instrumenta la espera en cola de entrada como tramo propio antes de suponer cualquier otra cosa.
Algunas trazas están incompletas y llevan a conclusiones falsas	Cada servicio decide el muestreo por su cuenta	Toma la decisión una vez, propágala en el contexto y respétala en todos los servicios.
La latencia media de un abanico es buena y el usuario espera mucho	Se espera al más lento de las llamadas paralelas	Pon plazo por llamada y sirve respuesta parcial cuando venza.
Cambiar de proveedor de telemetría exige tocar todos los servicios	Los servicios envían directamente al destino	Envía a un recolector y deja en él el reparto, el muestreo y la depuración de datos sensibles.

Seguridad, ética y costo

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

Preguntas de comprobación

1. ¿Qué tres cosas da una traza que ninguna métrica puede dar?
2. ¿Cuáles son los tres sitios donde se rompe la propagación de contexto y cómo se arregla cada uno?
3. ¿Por qué el cruce de una cola se modela como enlace y no como padre-hijo?
4. ¿Qué indica un hueco en una traza y cuál es su causa más frecuente?
5. ¿Por qué la decisión de muestreo debe tomarse una vez y propagarse?

Referencias

- W3C (2025). Trace Context — cabecera estándar de propagación y bit de muestreo. <https://www.w3.org/TR/trace-context/>
- OpenTelemetry (2025). Traces: spans, links and context propagation — modelo de datos y enlaces. <https://opentelemetry.io/docs/concepts/signals/traces/>
- OpenTelemetry (2025). Collector: processors and exporters — muestreo en la cola, depuración y reparto. <https://opentelemetry.io/docs/collector/>
- Sigelman, B. y otros (2010). Dapper, a large-scale distributed systems tracing infrastructure — origen del modelo. <https://research.google/pubs/pub36356/>
- Dean, J. y Barroso, L. (2013). The tail at scale — latencia de cola y respuesta parcial en abanicos. <https://research.google/pubs/pub40801/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 10 en PDF · Manual integral

Anterior	Índice	Siguiente
← 123 · Métricas, cardinalidad y modelos RED y USE	Parte 10 · Programa	125 · Dashboards, alertas accionables y fatiga →

Evaluación — 124 Tracing distribuido y OpenTelemetry

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 traza-distribuida 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.

125 — Dashboards, alertas accionables y fatiga

Parte: 10 — Observabilidad, SRE y confiabilidad Nivel: avanzado · Horas estimadas: 4 Laboratorio: observability · Estado: EXECUTABLECORE

Propósito

Decidir qué despierta a una persona y qué se mira en una pantalla, sabiendo que el fallo característico de esta materia no es medir poco sino producir tanta señal que deja de serlo. La clase da una prueba de cuatro preguntas que descarta la mayoría de las alertas existentes, defiende alertar por síntomas y diagnosticar por causas, mide la fatiga con cifras concretas y sostiene una idea incómoda: el trabajo bien hecho reduce el número de alertas y mejora la detección al mismo tiempo.

Resultados de aprendizaje

Al finalizar podrás:

1. Aplicar la prueba de cuatro preguntas a cada alerta existente.
2. Separar lo que despierta a alguien de lo que abre una tarea o solo se mira.
3. Alertar por síntomas, con las excepciones justificadas.
4. Medir la fatiga y actuar sobre sus causas.
5. Diseñar paneles que respondan una pregunta cada uno.

Conceptos centrales

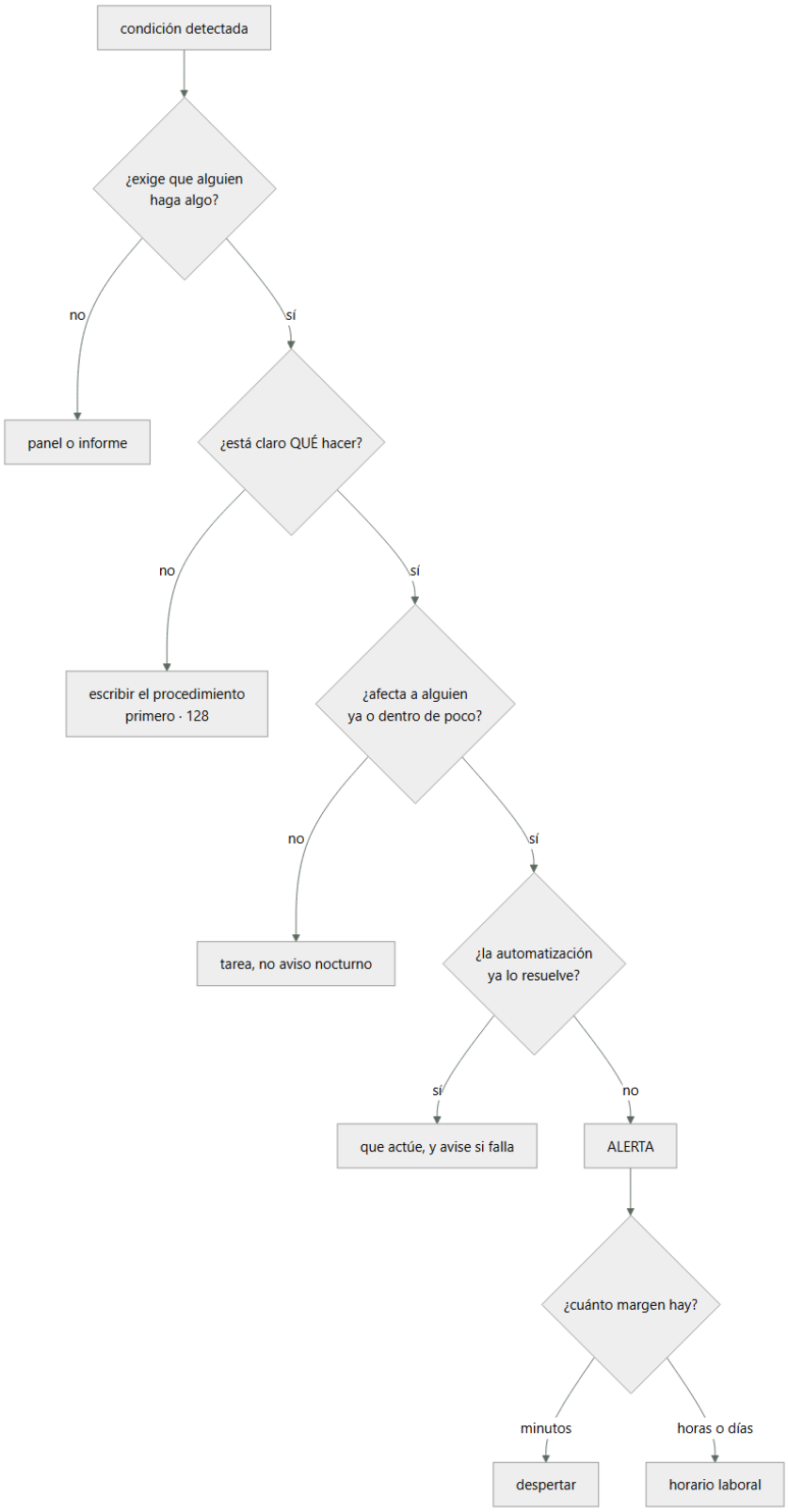
Concepto	Comprensión verificable
alerta accionable	La que exige que una persona haga algo concreto que la automatización no puede hacer. Si no, no es una alerta.
síntoma	Efecto observable para quien usa el sistema: errores, lentitud, funcionalidad ausente. Es lo que debe despertar a alguien.
indicador adelantado	Causa que aún no es síntoma pero lo será con seguridad y con margen para actuar: disco llenándose, certificado que caduca, saturación.
fatiga de alertas	Estado en el que la gente deja de leer los avisos. Se mide, y aparece antes de lo que se cree.
inhibición	Suprimir alertas derivadas cuando ya se ha disparado la causa. Evita cuarenta avisos por un solo fallo.
duración de condición	Tiempo que la condición debe mantenerse antes de avisar. Es lo que separa una alerta de un parpadeo.

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

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 prueba de las cuatro preguntas

Antes de discutir umbrales, conviene descartar. Toda alerta tiene que superar estas cuatro:

1. ¿EXIGE QUE ALGUIEN HAGA ALGO?
   si la respuesta habitual es mirarla y cerrarla, no es una alerta
2. ¿ESTÁ CLARO QUÉ HACER?

```
   si quien la recibe tiene que investigar desde cero qué significa,
   falta el procedimiento (clase 128), no sobra la alerta

3. ¿AFECTA A ALGUIEN, YA O CON SEGURIDAD DENTRO DE POCO?
   si no afecta a nadie, es información

4. ¿LA AUTOMATIZACIÓN NO LO RESUELVE YA?
   si el sistema se recupera solo, que se recupere
   y que avise solo cuando NO lo consiga
```

La cuarta es la que más alertas elimina en sistemas modernos: reinicios automáticos, reintentos, autoescalado y reversiones automáticas resuelven solos lo que antes exigía una persona. Avisar de lo que ya se arregló es ruido.

Y las tres categorías de destino, que hay que separar sin ambigüedad:

```
DESPERTAR       hay que actuar en minutos y afecta a usuarios
                → y esto es un recurso escaso: cada uso gasta atención
TAREA           hay que actuar, y puede esperar al horario laboral
                → certificados, capacidad, deuda operativa
PANEL           no hay que actuar; sirve para investigar
```

Y el criterio que decide entre las dos primeras no es la gravedad, sino el margen: cuánto tiempo hay antes de que sea un problema.

```
disco al 95 %, llenándose en 40 min      despertar
disco al 80 %, llenándose en 6 días      tarea
```

Y una consecuencia organizativa: toda alerta tiene un dueño, sacado del catálogo de la clase 095. Una alerta que llega a un buzón común es una alerta que nadie atiende.

2. Síntomas, causas y las excepciones

La regla general:

```
ALERTAR por síntomas       lo que sufre quien usa el sistema
DIAGNOSTICAR por causas    lo que hay dentro
```

Y el motivo es doble:

```
las causas son muchas y cambian     alertar por cada una es interminable
los síntomas son pocos y estables   errores, lentitud, funcionalidad ausente

y sobre todo:
  una causa sin síntoma no merece despertar a nadie
  un síntoma sin causa conocida SÍ, aunque no sepamos por qué
```

La última línea es la importante: alertar por causas deja fuera todo lo que nadie anticipó, que es exactamente lo que produjo dieciocho de los veintiún problemas de la parte 09.

Ejemplos de la traducción:

```
causa                          síntoma equivalente
uso de procesador > 90 %    →  latencia por encima del objetivo
un pod se reinició          →  proporción de errores
la réplica va retrasada     →  lecturas incorrectas o error visible
memoria al 85 %             →  (normalmente nada: no alertar)
```

Y las excepciones legítimas, que son causas con margen de actuación y consecuencia grave:

```
certificado que caduca en 14 días
disco o cuota que se llenará en horas
credencial que expira
saturación creciendo con latencia todavía normal      clase 123
copia de seguridad que no se ha ejecutado             clase 088
bucle, consumidor o trabajador parado                 ley 13
```

La última categoría merece su propio apartado en cualquier sistema: lo que no se ejecuta no da error, y este programa la ha visto quince veces. Su alerta no mira valores, mira antigüedad.

Y el otro par que hay que tener siempre, aunque no se alerte por causas:

```
ausencia de datos                          clase 123
la propia telemetría caída
```

Porque si el sistema de vigilancia se cae, todo lo demás parece correcto.

3. La fatiga se mide

La fatiga no es una queja: es un estado que se puede medir y que llega antes de lo que la gente cree.

```
alertas por turno de guardia
  0-1     sostenible
  2-4     empieza a costar leerlas
  > 5     se leen por encima; y las importantes se pierden entre las demás

proporción de alertas que exigieron una acción
  < 50 %  el conjunto está mal calibrado

proporción que se resolvió sola antes de que nadie mirara
  > 20 %  falta duración de condición o sobra la alerta

alertas repetidas: las 5 que más se disparan
  suelen ser el 60-80 % del volumen
```

La última es la palanca: arreglar cinco alertas suele arreglar la mayor parte del problema.

Y las cuatro correcciones, en orden de eficacia:

```
1. BORRAR lo que no supera la prueba de cuatro preguntas
2. AGRUPAR e INHIBIR
   cuarenta pods caídos por un nodo → una alerta del nodo
   una dependencia caída → suprimir las de quienes dependen de ella
3. AÑADIR DURACIÓN
   «errores > 5 % durante 5 minutos», no «errores > 5 % ahora»
   → elimina casi todo lo que se resuelve solo
4. ARREGLAR LA CAUSA
   la alerta que se dispara cada noche a las 3 no necesita otro umbral:
   necesita que alguien mire qué pasa a las 3
```

Y dos ajustes técnicos que evitan el parpadeo:

```
histéresis           avisar al 90 % y resolver al 80 %, no al 89,9 %
ventana de silencio  no repetir la misma alerta cada minuto
```

Y dos mecanismos organizativos que mantienen el conjunto sano en el tiempo:

```
REVISIÓN DE LO DISPARADO
  cada alerta que sonó se clasifica: real, ruido, duplicada
  semanal, en diez minutos

LA PREGUNTA DE CADA INCIDENTE
  «¿qué alerta debería haber sonado y no sonó?»
  → es la que hace crecer el conjunto por el lado correcto
```

Las dos juntas son lo que permite que el número baje mientras la detección mejora: se quita lo que no sirve y se añade solo lo que un incidente real ha demostrado que falta.

Y una advertencia sobre las ventanas de mantenimiento: silenciar durante un despliegue es razonable, y silenciar sin fecha de fin es cómo mueren las alertas. Todo silencio caduca.

4. Paneles que responden una pregunta

El panel de sesenta gráficos no lo lee nadie. Y el motivo es el mismo de siempre: no responde ninguna pregunta concreta.

La taxonomía que funciona son tres tipos, con propósitos distintos:

```
1. ¿ESTÁ FUNCIONANDO?   uno por servicio, seis paneles como mucho
   peticiones, errores, latencia (percentiles), saturación,
   estado frente al objetivo (clase 126) y línea de cambios (clase 121)
   → se mira en treinta segundos y se contesta sí o no

2. ¿POR QUÉ NO FUNCIONA?  uno por subsistema, con detalle
   dependencias, colas, agrupadores, recursos
   → se abre cuando el primero dice que no

3. PARA UN INCIDENTE CONCRETO   se crea durante y se conserva
   → y si se vuelve a usar tres veces, se convierte en uno de tipo 2
```

Y las reglas de construcción que más mejoran un panel:

```
el objetivo dibujado como línea, para saber si el valor es bueno
```

```
los cambios superpuestos: despliegues e interruptores
escalas comparables entre gráficos que se leen juntos
nada de medias donde importe la cola: percentiles
el enlace al procedimiento correspondiente
```

La primera es la que convierte un gráfico en información: un valor de 240 ms no dice nada sin saber si el objetivo es 100 o 500.

Y el antipatrón de la pantalla permanente en la pared: se convierte en decoración en dos semanas. Si algo tiene que llamar la atención, es una alerta; si no, no hace falta que esté siempre visible.

Y una comprobación anual muy reveladora, la misma lógica que la clase 121 aplicó a las métricas:

```
¿qué paneles no ha abierto nadie en 90 días?
→ borrarlos; están compitiendo por la atención con los que sirven
```

Y la lista de comprobación de la clase:

- toda alerta supera las cuatro preguntas
- toda alerta tiene dueño en el catálogo
- se alerta por síntomas; las causas alertadas están justificadas por margen
- hay alertas de antigüedad para lo que puede dejar de ejecutarse
- hay alertas de ausencia de datos y de la propia telemetría
- las alertas tienen duración de condición e histéresis
- hay inhibición de alertas derivadas
- todo silencio tiene fecha de fin
- se miden alertas por turno y proporción accionable
- se revisa semanalmente lo disparado y se clasifica
- cada incidente pregunta qué alerta faltaba
- cada servicio tiene un panel de seis gráficos que responde sí o no
- los paneles llevan el objetivo dibujado y los cambios superpuestos

Y el cierre que enlaza con la clase siguiente: la línea del objetivo que estos paneles necesitan no está definida en ninguna parte todavía. Decidir qué significa que el sistema funciona, con un número, y qué margen hay para fallar es la materia de la clase 126.

Ejemplo trabajado

CloudShop tiene 340 alertas configuradas y una guardia que nadie quiere hacer. El ejercicio dura un trimestre y su resultado contradice la intuición: al final hay muchas menos alertas y se detectan muchas más cosas.

Punto de partida.

```
alertas configuradas                                  340
alertas disparadas por semana                         410
alertas por turno de guardia (12 h)                    29
proporción que exigió alguna acción                   11 %
proporción que se resolvió sola antes de mirarla      57 %
silenciadas indefinidamente                            46
incidentes detectados por un cliente antes que por
  una alerta                                  9 de 14
```

Veintinueve avisos por turno y once por ciento accionables. La guardia consistía en cerrar avisos.

Paso 1: las cinco que más suenan.

```
uso de procesador > 80 %                    141            0
un pod se reinició                           88            2
latencia de una consulta > 100 ms            52            0
memoria > 85 %                               37            0
reintento contra el proveedor de pago        31            1
                                            ■■■■■
                                            349  = 85 % del total
```

Cinco alertas producían el 85 % del volumen y tres acciones al mes. Las cuatro primeras eran causas sin síntoma y se borraron; la quinta se convirtió en un panel.

```
alertas disparadas por semana          410            61
```

Paso 2: la prueba de cuatro preguntas sobre las 340.

```
no exigían ninguna acción                    184
no estaba claro qué hacer                     41   → 12 se conservaron
                                                    escribiendo el
                                                    procedimiento;
```

```
                                                          29 se borraron
no afectaban a nadie                                  52
la automatización ya lo resolvía                      38
  reinicios, reintentos, autoescalado, reversión automática
superaron las cuatro                                  25

alertas configuradas                          340             37
  (25 supervivientes + 12 con procedimiento nuevo)
```

Paso 3: síntomas que faltaban.

Y aquí el conjunto creció, con la pregunta de cada incidente aplicada a los veintiún problemas de la parte 09 y a los catorce incidentes del semestre:

```
alertas nuevas, por síntoma                             9
  proporción de errores de cada servicio de cara al cliente
  latencia por encima del objetivo
  pedidos que no avanzan de estado en 30 min          clase 115
  emitido frente a procesado en cada frontera         clase 121

alertas nuevas, por antigüedad (ley 13)                 7
  consumidor sin avanzar
  publicador de la tabla de salida parado             clase 116
  bucle de reconciliación sin sincronizar             clase 103
  cola de tareas del motor sin trabajadores           clase 119
  copia de seguridad no ejecutada
  cola de fallidos con algo dentro                    clase 113
  instantáneas del lago sin caducar                   clase 112

alertas nuevas, indicadores adelantados                 5
  saturación de agrupadores y colas                   clase 123
  certificados y credenciales por caducar
  cuota o disco con proyección de llenado

alertas de ausencia                                    15

alertas configuradas                          340             73
  de las cuales, existían antes                340             25
  nuevas                                         —              48
```

Setenta y tres alertas, de las que casi dos tercios son nuevas. El conjunto no se recortó: se sustituyó.

Paso 4: agrupación, inhibición y duración.

```
disparos por semana                             61             19
qué cambió
  duración de condición en 31 alertas         instantáneo    3-5 min
  inhibición por dependencia caída            no              sí
  agrupación por nodo y por servicio          no              sí
  histéresis                                  no              sí
```

Y el efecto de la inhibición en un incidente real:

```
caída de una zona de disponibilidad
  avisos que se habrían enviado sin inhibición                88
  avisos enviados                                              1
```

Los silencios indefinidos.

```
silencios activos                                          46
de ellos, sin fecha de fin                                 46
el más antiguo                                         19 meses
alertas que estaban silenciadas y eran de las buenas        4
```

Cuatro alertas útiles llevaban meses silenciadas y nadie lo sabía. Desde entonces todo silencio caduca a los 7 días como máximo, y renovarlo exige justificarlo.

Los paneles.

```
paneles                                       118             34
abiertos alguna vez en 90 días                 29             34
gráficos en el panel principal de un servicio  41              6
con el objetivo dibujado                      0 %           100 %
con los cambios superpuestos                  0 %           100 %
pantallas permanentes en la pared               3              0
```

El resultado al cabo del trimestre.

```
alertas configuradas                          340            73
disparadas por semana                         410            19
por turno de guardia                           29           1,4
proporción accionable                         11 %           78 %
proporción resuelta sola antes de mirarla     57 %            6 %
silencios sin fecha                            46             0
incidentes detectados por un cliente
  antes que por una alerta                  9 de 14        1 de 11
incidentes detectados por una alerta        5 de 14        10 de 11
tiempo medio hasta detección                 41 min        3 min 20
```

Y el dato que se recoge para la calificación de la parte:

```
la clase 120 predijo que el trabajo de la parte 10
REDUCIRÍA el número de alertas
→ de 340 a 73 configuradas, y de 410 a 19 disparos semanales
y predijo que la mejora vendría más de síntomas que de causas
→ 10 de las 11 detecciones fueron por alertas de síntoma o de antigüedad;
  1 por indicador adelantado
```

La lección que esta clase traslada a la parte 10: el número bajó de 340 a 73 y la detección subió de 5 de 14 a 10 de 11, y son la misma cosa. Con veintinueve avisos por turno, las cinco que importaban estaban enterradas. Y lo que más aportó no fue borrar —aunque hubo que borrar 315—, sino añadir cuarenta y ocho alertas nuevas que nadie había escrito, casi todas de dos tipos: síntomas visibles para el usuario y cosas que habían dejado de ejecutarse sin dar error.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-10-observability-sre-reliability/125-dashboards-alertas-accionables-y-fatiga/lab.
py
```

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

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

Evidencia esperada

El artefacto principal es telemetría correlacionada con una pregunta operativa. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye tablero-operativo 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 guardia consiste en cerrar avisos sin hacer nada	Las alertas no superan la prueba de las cuatro preguntas	Borra lo que no exige acción, no afecta a nadie o ya lo resuelve la automatización; conserva lo que falta procedimiento escribiéndolo.
Un solo fallo genera decenas de avisos	No hay agrupación ni inhibición de alertas derivadas	Agrupa por nodo y servicio y suprime las de quienes dependen de algo que ya está alertado.
Muchas alertas se resuelven solas antes de que nadie mire	Se alerta sobre el valor instantáneo, sin duración de condición	Exige que la condición se mantenga varios minutos y añade histéresis para evitar parpadeos.
Alertas útiles llevan meses silenciadas y nadie lo sabe	Los silencios no caducan	Pon fecha de fin obligatoria y corta; renovar exige justificarlo.
Solo se detecta lo que alguien anticipó	Se alerta por causas técnicas y no por síntomas visibles	Alerta por errores, latencia y funcionalidad ausente; deja las causas para diagnosticar y para indicadores con margen.
Nadie mira los paneles	Tienen decenas de gráficos y no responden ninguna pregunta concreta	Un panel por servicio con seis gráficos que contesten si funciona, con el objetivo dibujado y los cambios superpuestos.

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 que debe superar una alerta?
2. ¿Por qué se alerta por síntomas y no por causas, y qué causas son excepción?
3. ¿Con qué cifras se mide la fatiga y a partir de cuántas alertas por turno aparece?
4. ¿Por qué todo silencio debe tener fecha de fin?
5. ¿Qué debe contener un panel que responda si un servicio está funcionando?

Referencias

- Google SRE (2025). Practical alerting and symptom-based paging — alertar por síntomas y evitar el ruido. <https://sre.google/sre-book/practical-alerting/>
- Google SRE (2025). Being on-call — carga sostenible de guardia y sus umbrales. <https://sre.google/sre-book/being-on-call/>
- Prometheus (2025). Alerting rules and Alertmanager: grouping, inhibition, silences — agrupación y supresión. <https://prometheus.io/docs/alerting/latest/alertmanager/>
- Rob Ewaschuk (2025). My philosophy on alerting — reglas prácticas para decidir qué merece despertar a alguien. <https://docs.google.com/document/d/199PqyG3UsyXlwieHaqbGiWVa8eMWi8zzAn0YfcApr8Q/preview>
- Grafana (2025). Dashboard design best practices — un panel, una pregunta. <https://grafana.com/docs/grafana/latest/dashboards/build-dashboards/best-practices/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 10 en PDF · Manual integral

Evaluación — 125 Dashboards, alertas accionables y fatiga

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 tablero-operativo 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.

126 — SLI, SLO, SLA y presupuesto de error

Parte: 10 — Observabilidad, SRE y confiabilidad Nivel: avanzado · Horas estimadas: 4 Laboratorio: sre · Estado: EXECUTABLECORE

Propósito

Poner un número a lo que significa que el sistema funciona, y usarlo para decidir. La clase distingue las tres cosas que se confunden —lo que se mide, lo que se promete internamente y lo que se firma—, insiste en que un objetivo de latencia se expresa como proporción y no como percentil, y desarrolla el mecanismo que convierte todo esto en algo útil: el presupuesto de error, que sirve para dos cosas concretas —decidir si se sigue entregando o se para a arreglar, y sustituir un montón de alertas de umbral por unas pocas basadas en el ritmo de consumo—.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir indicadores que midan lo que experimenta quien usa el sistema.
2. Expresar objetivos de latencia como proporción por debajo de un umbral.
3. Calcular el presupuesto de error y convertirlo en una regla de decisión.
4. Alertar por ritmo de consumo en varias ventanas, no por umbral instantáneo.
5. Fijar un objetivo alcanzable dadas las dependencias, y no por aspiración.

Conceptos centrales

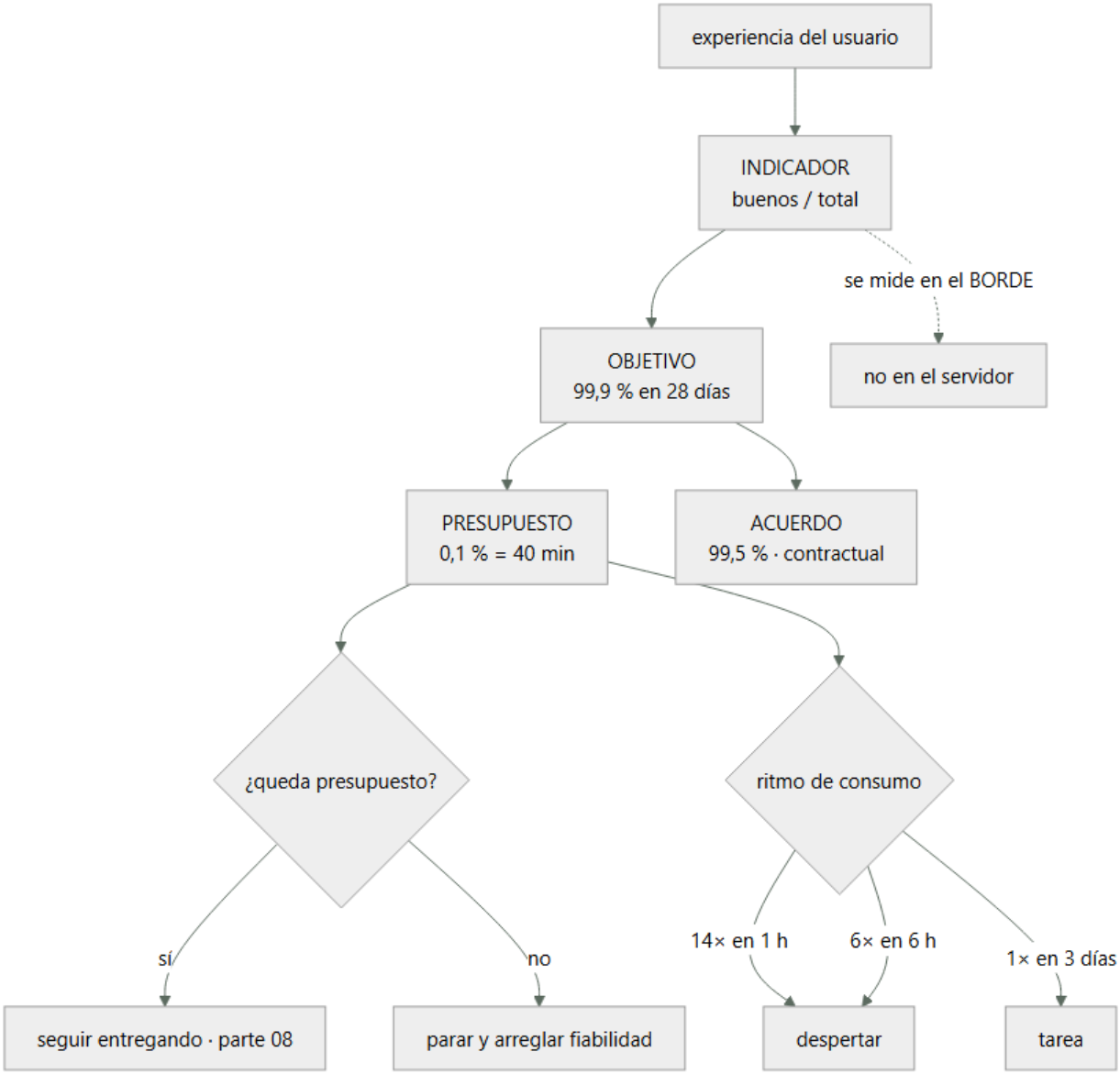
Concepto	Comprensión verificable
indicador	Medida de la experiencia de quien usa el sistema, expresada como proporción de eventos correctos sobre el total.
objetivo	Valor que ese indicador debe alcanzar en una ventana de tiempo. Es un compromiso interno y decide prioridades.
acuerdo	Promesa contractual con consecuencias. Siempre menos exigente que el objetivo interno, para tener margen.
presupuesto de error	Lo que queda del 100 % tras el objetivo. Es una cantidad de fallo permitida y se gasta.
ritmo de consumo	Velocidad a la que se agota el presupuesto respecto de lo previsto. Es la base de las alertas útiles sobre objetivos.
ventana móvil	Los últimos N días, actualizados continuamente. Sirve para decidir; el mes natural sirve para informar.

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

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

■ Flujo de razonamiento



Desarrollo

1. Tres cosas distintas, y cómo se elige la primera

```
INDICADOR   lo que se mide
            «proporción de peticiones correctas y por debajo de 300 ms»
OBJETIVO    lo que se promete internamente
            «99,9 % en los últimos 28 días»
ACUERDO     lo que se firma, con consecuencias
            «99,5 % mensual, o se descuenta de la factura»
```

Y la relación entre ellos no es negociable:

```
acuerdo  <  objetivo
```

Si se firma lo mismo que se persigue internamente, cualquier desviación es un incumplimiento contractual. El margen entre los dos es lo que permite reaccionar antes.

Elegir el indicador es la parte difícil, y tiene tres condiciones a la vez:

```
1. mide lo que EXPERIMENTA quien usa el sistema
2. se puede medir de verdad, con lo que hay
3. depende de nosotros lo suficiente como para actuar
```

Y la forma canónica es siempre la misma, y conviene forzarla:

```
indicador = eventos buenos / eventos válidos
```

Y de ahí sale la corrección más importante de esta clase:

```
mal   «latencia del percentil 99 por debajo de 300 ms»
bien  «proporción de peticiones por debajo de 300 ms ≥ 99 %»
```

Son casi lo mismo y no del todo: el segundo se puede sumar entre instancias y entre ventanas, se puede comparar, y encaja con el presupuesto. El primero es una métrica y arrastra los problemas de agregación de la clase 123.

Y el intervalo de histograma colocado en el objetivo, que la clase 123 anticipó, es exactamente lo que hace que esa proporción sea una división exacta.

Los tipos de indicador que cubren casi todo:

```
DISPONIBILIDAD   peticiones sin error / peticiones válidas
LATENCIA         peticiones por debajo del umbral / total
FRESCURA         datos con antigüedad menor que X / total
                 → para lo asíncrono y los informes
CORRECCIÓN       resultados correctos / total
                 → cuando se puede comprobar; muy valioso y poco usado
COBERTURA        elementos procesados / elementos que debían procesarse
                 → es la métrica de conservación de la clase 121
```

Y dos decisiones que cambian el número por completo:

```
DÓNDE SE MIDE     en el borde o en el cliente, no en el servidor
                  → el servidor no ve la cola de entrada ni la red
QUÉ ES «VÁLIDO»   los errores por petición mal formada del cliente
                  no cuentan como fallo nuestro; hay que escribirlo
```

Y una advertencia sobre el número de indicadores: entre dos y cuatro por servicio de cara al usuario. Más no se pueden gobernar, y la mayoría de los servicios internos no necesitan ninguno propio.

2. El presupuesto, y para qué sirve de verdad

El objetivo deja un resto, y ese resto es una cantidad concreta de fallo permitido:

```
objetivo    presupuesto      en 28 días        en 1 día
99      %       1 %          6 h 43 min        14 min
99,5    %       0,5 %        3 h 22 min         7 min
99,9    %       0,1 %          40 min          1,4 min
99,95   %       0,05 %         20 min           43 s
99,99   %       0,01 %          4 min            9 s
```

Y la fila de abajo enseña por qué un objetivo aspiracional es un problema: con cuatro minutos al mes, un solo despliegue fallido lo consume entero, y ningún proceso humano reacciona en ese tiempo.

Y lo importante no es la tabla, sino la regla de decisión que la acompaña:

```
queda presupuesto     → se sigue entregando funcionalidad
                        con las prácticas de la parte 08
presupuesto agotado   → se para la funcionalidad nueva y se dedica
                        el esfuerzo a fiabilidad, hasta recuperarlo
```

Eso es lo que convierte el objetivo en algo más que un informe. Y tiene una condición previa sin la cual no funciona:

```
la regla la acepta POR ADELANTADO quien puede decidir prioridades
→ si se discute cuando el presupuesto se agota, no hay regla
```

Y dos matices honestos:

```
el presupuesto no se «ahorra» para gastarlo de golpe
  → es un límite, no una hucha
un mes con presupuesto intacto es también una señal
  → o el objetivo es demasiado flojo, o se está siendo demasiado
    conservador entregando
```

La segunda es la menos intuitiva y la más útil: un sistema que nunca gasta presupuesto probablemente está entregando demasiado despacio.

Y qué gasta presupuesto y qué no:

```
gasta     todo fallo que sufre el usuario, sea de quien sea la culpa
          incluidas las caídas del proveedor
no gasta  el mantenimiento anunciado, si el usuario lo acepta
          y las peticiones inválidas, si así se definió
```

La primera línea sorprende y es deliberada: al usuario le da igual de quién sea la culpa. Si una dependencia del proveedor tumba el servicio, el presupuesto se gasta, y eso es lo que obliga a diseñar para tolerarlo.

3. Alertar por ritmo de consumo

Este apartado sustituye buena parte de las alertas de umbral de la clase 125 por unas pocas mejores.

El problema del umbral fijo:

```
«avisar si los errores superan el 5 %»
  → un pico del 6 % durante 30 segundos: irrelevante, y avisa
  → un 1,5 % sostenido durante dos días: gasta el presupuesto entero,
    y no avisa nunca
```

La alternativa es medir a qué velocidad se gasta el presupuesto:

```
ritmo 1×    se agota justo al final de la ventana: lo previsto
ritmo 14,4× se agota en 2 días: hay que actuar ya
ritmo 0,5×  sobra presupuesto
```

Y las alertas se construyen combinando una ventana corta y otra larga, para tener sensibilidad sin falsos avisos:

```
DESPERTAR   ritmo > 14,4× en 1 h  Y  > 14,4× en 5 min
            → consume el 2 % del presupuesto en una hora

DESPERTAR   ritmo > 6× en 6 h     Y  > 6× en 30 min
            → consume el 5 % en seis horas

TAREA       ritmo > 3× en 1 día   Y  > 3× en 2 h
TAREA       ritmo > 1× en 3 días  Y  > 1× en 6 h
```

La condición de la ventana corta es la que evita avisar por algo que ya terminó: si la ventana larga está alta pero la corta ya bajó, el problema pasó.

Y lo que se gana:

```
cuatro alertas por servicio en vez de veinte umbrales
cubre errores, latencia y cualquier otro indicador con la misma forma
su gravedad se corresponde con el daño real al usuario
y no hay que ajustar umbrales a mano por servicio
```

La ventana del objetivo, que también decide:

```
móvil de 28 días     para decidir: siempre refleja el estado actual
mes natural          para informar y para el acuerdo contractual
```

Y veintiocho en lugar de treinta por un motivo práctico: contiene siempre el mismo número de fines de semana, así que las comparaciones entre ventanas no se distorsionan.

Y una advertencia sobre la ventana móvil: un incidente sale del cálculo de golpe cuando cumple 28 días, y el indicador «mejora» sin que nadie haya hecho nada. Conviene saberlo antes de celebrarlo.

4. Poner el número, con las dependencias delante

El objetivo no se elige por aspiración. Los tres insumos:

```
1. LO QUE HAY AHORA
   medir el indicador durante 4-6 semanas sin objetivo
   → si ahora es 99,2 %, poner 99,99 % no es un objetivo: es un deseo

2. LO QUE NOTA EL USUARIO
   ¿a partir de qué punto se queja, se va o llama?
   → suele haber un escalón, y ese es el sitio del objetivo

3. LO QUE PERMITEN LAS DEPENDENCIAS
```

Y la tercera se calcula, y es la que más ilusiones rompe:

```
si una petición necesita 5 dependencias en serie, cada una al 99,9 %
disponibilidad máxima = 0,999^5 = 99,50 %

→ no se puede prometer 99,9 % sin cambiar el diseño
```

Y las formas de romper esa cadena son las que la clase 130 desarrolla:

```
hacer la dependencia opcional: servir sin ella   clase 124
cachear su respuesta para sobrevivir a su caída  clase 111
```

```
hacerla asíncrona: encolar en vez de llamar      clase 113
redundancia: dos proveedores o dos regiones
```

Y hay una decisión de coste explícita, porque cada nueve cuesta más que el anterior:

```
de 99   a 99,9    suele ser trabajo de ingeniería normal
de 99,9 a 99,99   suele exigir redundancia entre regiones y automatismos
de 99,99 en adelante   exige que nada dependa de una intervención humana
```

Y la pregunta que evita gastar de más: ¿qué pierde el negocio por cada nueve que falta? Si nadie sabe responderla, el objetivo lo está eligiendo la moda.

El acuerdo contractual, que es otra cosa:

```
se mide como lo mide el CLIENTE, no como nos conviene
lleva consecuencias: descuentos, penalizaciones
y por eso va varios puntos por debajo del objetivo interno
y debe definir con precisión qué cuenta, cómo se mide y quién arbitra
```

Y un aviso sobre qué no debe tener objetivo formal:

```
todo servicio interno con dos consumidores
cada componente por separado
todo lo que no se traduzca en algo que alguien note
```

Poner objetivos a todo es la ley 15 aplicada a los objetivos: con cuarenta, ninguno decide nada.

Y la lista de comprobación de la clase:

- cada indicador se expresa como buenos entre válidos
- los de latencia son proporción bajo umbral, no percentiles
- está escrito qué eventos son válidos y cuáles no cuentan
- se mide en el borde o en el cliente
- hay entre dos y cuatro indicadores por servicio de cara al usuario
- el objetivo se fijó midiendo antes, no por aspiración
- está calculado el máximo que permiten las dependencias
- el acuerdo contractual es menos exigente que el objetivo
- la regla de decisión al agotar el presupuesto está aceptada de antemano
- las alertas son de ritmo de consumo, con ventana corta y larga
- la ventana de decisión es móvil, y la de informe es natural
- el estado del presupuesto está en el panel principal del servicio

Y el cierre que enlaza con la clase siguiente: cuando el presupuesto se consume de golpe, hay un incidente. Cómo se declara, quién manda mientras dura, qué se comunica y qué se hace después es la materia de la clase 127.

Ejemplo trabajado

CloudShop define objetivos para los cinco servicios de cara al cliente. El ejercicio tiene tres momentos: la aspiración que resultó imposible, las alertas de umbral que se sustituyeron, y el mes en que la regla del presupuesto se aplicó de verdad.

Momento 1: el objetivo aspiracional que las matemáticas descartaron.

La dirección propuso 99,99 % para el flujo de compra. Antes de aceptarlo se hicieron las tres cuentas del apartado cuarto.

```
LO QUE HAY AHORA, medido 6 semanas sin objetivo
  disponibilidad del flujo de compra                    99,21 %
  proporción por debajo de 500 ms                       97,4 %

LO QUE PERMITEN LAS DEPENDENCIAS
  identidad            99,95 %
  catálogo             99,90 %
  precios              99,90 %
  inventario           99,80 %
  pago (externo)       99,50 %
  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  producto             99,05 %   ← techo actual
```

Noventa y nueve coma cero cinco. El objetivo propuesto era físicamente imposible sin cambiar el diseño, y con la configuración de entonces ni siquiera se podía prometer 99,5 %.

Se rediseñaron tres dependencias con las técnicas del apartado cuarto:

```
catálogo                          llamada síncrona   caché con valor
```

```
                                                caducado servible
precios                          llamada síncrona   caché, y precio
                                                    base si falla
pago                             síncrono en la
                                 petición           encolado, con
                                                    confirmación posterior

techo por dependencias             99,05 %           99,74 %
```

Y el objetivo se fijó en 99,5 %, con el acuerdo contractual en 99,0 %.

```
objetivo interno    99,5 %   → presupuesto: 3 h 22 min / 28 días
acuerdo firmado     99,0 %   → margen: 3 h 22 min más antes de
                               consecuencias contractuales
```

La corrección del indicador de latencia.

```
primera versión    «percentil 99 por debajo de 500 ms»
problema           se calculaba promediando instancias (clase 123)
                   y no se podía combinar con el presupuesto

versión final      «proporción de peticiones por debajo de 500 ms ≥ 99 %»
                   con un intervalo de histograma exactamente en 500 ms
```

Y dónde se mide, que cambió el número:

```
proporción bajo 500 ms                    99,4 %            97,4 %
diferencia                       la cola de entrada y la red,
                                 que el servidor no ve
```

Dos puntos de diferencia. El servidor decía que se cumplía el objetivo y el usuario decía que no.

Momento 2: las alertas de ritmo sustituyen a veinte umbrales.

```
alertas de umbral sobre errores y latencia      20              0
alertas de ritmo de consumo                      0              4
ventanas                                         —      1 h/5 min, 6 h/30 min,
                                                        1 d/2 h, 3 d/6 h
```

Y el comportamiento comparado, con seis meses de datos:

```
avisos por mes                                  31                6
de ellos, con daño real al usuario               7                6
falsos avisos por picos breves                  19                0
degradaciones lentas no detectadas               5                0
```

Las dos últimas filas son el argumento entero. Los umbrales avisaban diecinueve veces por picos irrelevantes y se perdían cinco degradaciones lentas:

```
ejemplo real  1,4 % de errores sostenido durante 3 días
  umbral del 5 %          nunca se disparó
  ritmo de consumo        avisó como tarea a las 6 h
  presupuesto consumido   84 % antes de que nadie lo hubiera notado
```

Momento 3: el mes en que se agotó el presupuesto.

```
día 3   despliegue con un defecto: 41 min de errores parciales
        presupuesto consumido: 38 %
día 11  caída del proveedor de pago: 1 h 12 min
        presupuesto consumido acumulado: 74 %
día 17  incidente de conexiones (clase 109): 27 min
        presupuesto consumido acumulado: 88 %
día 19  degradación por saturación: 34 min
        presupuesto consumido acumulado: 105 %
```

Y entonces se aplicó la regla, que estaba aceptada por escrito desde el principio:

```
funcionalidad nueva                       parada 9 días
esfuerzo dedicado a fiabilidad            el del equipo entero
trabajo hecho en esos 9 días
  agrupador de conexiones y techos          clase 109
  plazos y respuesta parcial en 4 llamadas  clase 124
  caché con valor caducado servible         clase 111
  reversión automática por ritmo de consumo clase 102
```

Y la discusión que no hubo:

```
tiempo dedicado a discutir si parar o no                      0
```

```
motivo    la regla estaba aceptada por adelantado, con nombre
          y firma, desde la definición del objetivo
```

Y el efecto en los tres meses siguientes:

```
presupuesto consumido                        105 %              31 %
despliegues por semana                       6,4 (y luego 0)     7,1
incidentes                                    4                  1
```

Los despliegues no bajaron al mes siguiente: el equipo volvió a entregar más que antes, sobre un sistema que ya no gastaba el presupuesto.

Y el mes contrario, que también enseñó algo.

```
mes 7   presupuesto consumido: 4 %
```

Cuatro por ciento. Según el apartado segundo, eso es también una señal, y al revisarlo:

```
causa    dos equipos habían dejado de desplegar los viernes
         y estaban acumulando cambios «por prudencia»
efecto   lotes más grandes, y el incidente del mes 8 fue el mayor del año
decisión volver a desplegar a diario; el presupuesto está para gastarse
```

A los seis meses.

```
indicadores definidos                           0            11
servicios con objetivo                        0 de 5       5 de 5
objetivo del flujo de compra              aspiración 99,99 %   99,5 %
techo por dependencias                       99,05 %         99,74 %
disponibilidad real                          99,21 %         99,68 %
proporción bajo 500 ms (en el borde)          97,4 %          99,3 %
alertas de umbral sobre errores y latencia      20              0
alertas de ritmo de consumo                      0              4
avisos por mes                                  31              6
falsos avisos por picos breves                  19              0
degradaciones lentas no detectadas          5 / 6 meses         0
meses con la regla del presupuesto aplicada      —              1
discusiones sobre si aplicarla                   —              0
```

La lección que esta clase traslada a la parte 10: lo más valioso del ejercicio ocurrió antes de medir nada. Multiplicar las disponibilidades de las cinco dependencias demostró en diez minutos que el objetivo que se quería prometer era imposible, y eso convirtió la conversación de «esforzaos más» en «hay que hacer opcionales tres dependencias». Y el mecanismo que más cambió el día a día no fue el objetivo, sino su resto: una regla de parada aceptada por adelantado eliminó por completo la discusión que suele consumir los días posteriores a un mal mes.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-10-observability-sre-reliability/126-sli-slo-sla-y-presupuesto-de-error/lab.py
```

El laboratorio selecciona el motor de práctica sre 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 slo-servicio 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 SLO con presupuesto de error y política de acción. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye slo-servicio 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 objetivo es inalcanzable por más que se esfuerce el equipo	Se fijó por aspiración, sin calcular el techo que imponen las dependencias	Multiplica las disponibilidades de la cadena; si no llega, haz dependencias opcionales, cacheables o asíncronas antes de prometer nada.
El indicador de latencia no se puede combinar ni comparar	Está expresado como percentil en vez de como proporción bajo umbral	Define proporción de peticiones por debajo del umbral y coloca un intervalo de histograma justo ahí.
El servidor cumple el objetivo y el usuario se queja	Se mide donde no se ven la cola de entrada ni la red	Mide en el borde o en el cliente.
Una degradación leve y sostenida consume el presupuesto sin disparar nada	Las alertas son de umbral instantáneo	Alerta por ritmo de consumo del presupuesto, combinando una ventana larga con una corta.
Al agotarse el presupuesto se discute durante días qué hacer	La regla de decisión no estaba aceptada de antemano por quien prioriza	Acuerda y firma la regla al definir el objetivo, no cuando se incumple.
Hay objetivos para cuarenta componentes y ninguno decide nada	Ley 15 aplicada a los objetivos	De dos a cuatro indicadores por servicio de cara al usuario; los componentes internos no necesitan objetivo propio.

■ 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 relación debe haber entre el acuerdo contractual y el objetivo interno, y por qué?
2. ¿Por qué un objetivo de latencia se expresa como proporción y no como percentil?
3. ¿Para qué dos cosas concretas sirve el presupuesto de error?
4. ¿Por qué las alertas de ritmo de consumo combinan una ventana larga y una corta?
5. ¿Cómo se calcula el techo de disponibilidad que imponen las dependencias?

Referencias

- Google SRE (2025). Service level objectives — indicadores, objetivos y presupuesto de error. <https://sre.google/sre-book/service-level-objectives/>

- Google SRE (2025). Alerting on SLOs: multiwindow, multi-burn-rate — ventanas y ritmos de consumo. <https://sre.google/workbook/alerting-on-slos/>
- Beyer, B. y otros (2018). The Site Reliability Workbook, cap. 2 — cómo elegir indicadores útiles. <https://sre.google/workbook/implementing-slos/>
- Wilkinson, A. (2025). SLO implementation patterns — proporción de eventos buenos y ventanas móviles. <https://github.com/google/slo-generator>
- Prometheus (2025). Histograms and quantiles — por qué el intervalo colocado en el objetivo importa. <https://prometheus.io/docs/practices/histograms/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 10 en PDF · Manual integral

Anterior	Índice	Siguiente
← 125 · Dashboards, alertas accionables y fatiga	Parte 10 · Programa	127 · Incidentes, severidad, comando y comunicación →

Evaluación — 126 SLI, SLO, SLA y presupuesto de error

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 slo-servicio 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.

127 — Incidentes, severidad, comando y comunicación

Parte: 10 — Observabilidad, SRE y confiabilidad Nivel: avanzado · Horas estimadas: 4 Laboratorio: incident · Estado: EXECUTABLECORE

Propósito

Organizar lo que pasa entre que suena la alerta y que el sistema vuelve a funcionar, y lo que pasa después. La clase defiende cuatro cosas concretas: declarar pronto, porque declarar es barato y no declarar es caro; mitigar antes que diagnosticar, porque entender la causa puede llevar horas y el usuario está sufriendo ahora; un cambio cada vez, anunciado, que es la regla que separa un incidente ordenado de uno que se alarga solo; y una revisión posterior que busca por qué el sistema lo permitió, no quién lo hizo.

Resultados de aprendizaje

Al finalizar podrás:

1. Declarar un incidente con criterios escritos y sin pedir permiso.
2. Repartir los papeles, incluido cuando los asume una sola persona.
3. Mitigar con el catálogo de acciones construido en las partes anteriores.
4. Comunicar dentro y fuera con una cadencia fija.
5. Convertir la revisión posterior en cambios que se completan.

Conceptos centrales

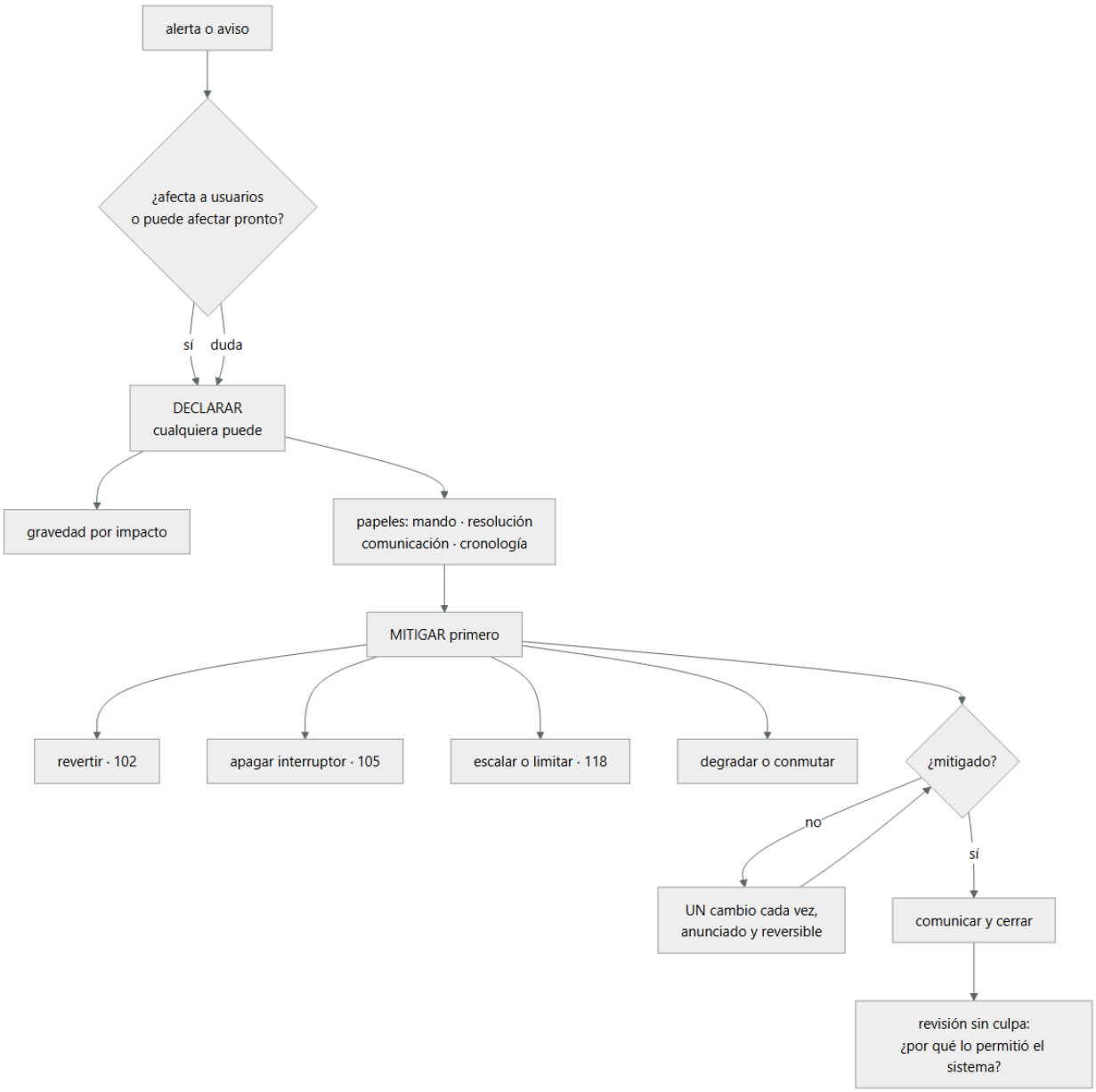
Concepto	Comprensión verificable
declaración	Acto explícito de decir que hay un incidente. Abre el canal, los papeles y el registro; cualquiera puede hacerlo.
mando del incidente	Quien coordina, decide y mantiene el orden. No es quien arregla ni tiene por qué ser quien más sabe.
mitigación	Acción que reduce el daño sin explicar la causa. Es siempre lo primero.
un cambio cada vez	Regla de oro durante un incidente: una modificación, anunciada, con forma de deshacerla, antes de la siguiente.
cronología	Registro de qué se supo, qué se hizo y cuándo. Se escribe durante, no después, porque después nadie lo recuerda.
revisión sin culpa	Análisis que busca por qué el sistema permitió el fallo. Si acaba en «alguien se equivocó», no ha terminado.

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

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. Declarar pronto

El error más común no es gestionar mal un incidente: es no llamarlo incidente hasta que lleva cuarenta minutos.

```
coste de declarar y que no fuera nada
  quince minutos de tres personas y un mensaje diciendo que ya está

coste de no declarar y que sí lo fuera
  nadie coordina, nadie avisa fuera, nadie apunta nada
  y la información se pierde para la revisión posterior
```

Y de ahí tres reglas:

```
1. cualquiera puede declarar, sin pedir permiso a nadie
2. ante la duda, se declara
3. cerrar un incidente que no era nada NO se penaliza ni se comenta
```

La tercera es la que hace posibles las dos primeras.

La gravedad se define por impacto, y con tres niveles basta:

```
ALTA    afecta a la mayoría de usuarios o al flujo principal, o hay
        pérdida de datos, o hay un problema de seguridad
```

```
        → respuesta inmediata, a cualquier hora, con comunicación externa

MEDIA   afecta a una funcionalidad secundaria o a un grupo de usuarios,
        o hay degradación clara
        → respuesta en horas, en horario laboral ampliado

BAJA    impacto pequeño o potencial, con margen
        → tarea priorizada
```

Y dos precisiones que evitan discusiones:

```
la gravedad se puede subir y bajar durante el incidente
se define por IMPACTO, no por nerviosismo ni por quién reclama
```

Y una advertencia que viene de la clase 107: redefinir qué cuenta como incidente es la forma más fácil de mejorar las cifras sin mejorar nada. Los criterios se escriben una vez y cambiarlos es un cambio, no una interpretación.

Y lo que la declaración pone en marcha, que debe ser automático y no una lista que alguien recuerda:

```
un canal dedicado, con nombre del incidente
el registro abierto, con marcas de tiempo
aviso a los papeles necesarios
y un documento donde va todo
```

2. Papeles, y los dos fallos que evitan

```
MANDO           coordina, decide, reparte y mantiene el foco
                NO arregla; si se pone a arreglar, deja de coordinar
RESOLUCIÓN      quien investiga y aplica cambios
COMUNICACIÓN    informa dentro y fuera con cadencia fija
CRONOLOGÍA      escribe qué se supo, qué se hizo y cuándo
```

Y los papeles existen para evitar dos fallos concretos, que son los que alargan los incidentes:

```
TODOS EXCAVANDO EN EL MISMO SITIO
  seis personas mirando la misma gráfica y nadie mirando las otras cinco
  → el mando reparte hipótesis: «tú mira la base, tú los despliegues»

NADIE HABLA CON EL EXTERIOR
  el equipo trabaja bien y soporte, comercio y clientes no saben nada
  → y a los veinte minutos llegan interrupciones que roban atención
```

Y dos precisiones prácticas:

```
el mando NO es quien más sabe
  quien más sabe suele ser quien mejor resuelve, y ese es otro papel
en un incidente pequeño una persona asume todos los papeles
  y lo dice en voz alta: «asumo mando y resolución»
  → lo peligroso no es acumular papeles, es que nadie sepa quién los tiene
```

Y la regla que más incidentes ha salvado, y que hay que enunciar sin matices:

```
UN CAMBIO CADA VEZ
  se anuncia en el canal antes de hacerlo
  se sabe cómo deshacerlo
  se espera a ver el efecto
  y solo entonces viene el siguiente
```

El motivo es que, con tres cambios simultáneos, si el sistema mejora nadie sabe por cuál, y si empeora tampoco. Y en un incidente hay prisa, que es exactamente cuando esto se incumple.

Y su corolario: nada se toca fuera del canal. Un cambio hecho en silencio por alguien que quería ayudar es la causa de una parte considerable de los incidentes que se alargan.

Y el relevo, para los largos:

```
se traspasa explícitamente: estado, hipótesis descartadas, cambios hechos,
qué está en marcha y qué se ha comunicado
y quien entra repite en voz alta lo que ha entendido
```

3. Mitigar primero

La tentación durante un incidente es entender qué pasa. Y entender puede llevar horas mientras el usuario sufre.

```
primero   que deje de doler
después   por qué dolía
```

Y eso es viable porque las partes anteriores han construido un catálogo de mitigaciones que no requieren saber la causa:

```
revertir al artefacto anterior                      clase 102
apagar un interruptor de funcionalidad              clase 105
revertir la confirmación del entorno                clase 103
escalar el servicio o el consumidor                 clases 113, 117
limitar el ritmo de un cliente concreto             clase 118
degradar: servir del caché, quitar una sección      clases 111, 124
conmutar a otra réplica o región                    clase 109
detener un proceso por lotes que compite            clase 117
vaciar o desviar una cola                           clase 113
```

Y la pregunta que ordena la elección, que es la de la clase 121:

```
¿qué cambió en los últimos treinta minutos?
  → si hay un cambio reciente, revertirlo es la primera hipótesis
  → y es la mitigación más rápida y más segura que existe
```

Y la excepción que hay que tener presente, de la clase 102: revertir no sirve si se cruzó el punto de no retorno. Ahí la mitigación es apagar, no volver atrás.

Y dos trampas frecuentes:

```
«ya casi lo tengo»          lleva 40 minutos y hay una reversión
                            disponible desde el minuto 3
reiniciar sin mirar         a veces mitiga y destruye la evidencia:
                            capturar antes lo que se pueda
```

La segunda merece un procedimiento: antes de reiniciar, guardar lo que se perderá —volcado de memoria, estado de las colas, últimas trazas—. Cuesta un minuto y es la diferencia entre poder explicarlo después o no.

Y el cierre del incidente, que también se declara:

```
MITIGADO      el usuario ya no sufre; puede que la causa siga ahí
RESUELTO      la causa está corregida
CERRADO       hecha la revisión y creadas las acciones
```

Y se comunica el paso a mitigado, no el paso a resuelto: lo que importa fuera es que ya funciona.

4. Comunicar, y revisar sin culpa

Fuera. La comunicación externa tiene tres reglas y se incumplen las tres:

```
1. CADENCIA FIJA, aunque no haya novedades
   cada 30 minutos en gravedad alta
   «seguimos investigando, próxima actualización a las 11:30»
   → el silencio se interpreta siempre como abandono

2. NUNCA prometer una hora de resolución
   se promete la hora de la SIGUIENTE actualización, que sí se controla

3. DECIR EL IMPACTO en términos del usuario
   «no se pueden completar pedidos», no «el servicio de pagos devuelve 503»
```

Y el contenido de cada mensaje:

```
qué está afectado, en términos de lo que la gente no puede hacer
desde cuándo
qué se sabe y qué no
si hay alternativa
cuándo será la próxima actualización
```

Dentro. Un canal por incidente, y una disciplina:

```
todo cambio se anuncia antes
las decisiones se escriben, no solo se dicen
las hipótesis descartadas se apuntan, para que nadie las repita
y las conversaciones paralelas vuelven al canal
```

La revisión posterior, que es donde el incidente se convierte en algo útil.

La regla que la define: buscar por qué el sistema lo permitió.

```
si la conclusión es «alguien se equivocó», la revisión no ha terminado

¿por qué era posible hacer eso?
```

```
¿por qué no lo detectó nada?
¿por qué tardó tanto en detectarse?
¿por qué la mitigación tardó lo que tardó?
¿qué hizo que fuera difícil de diagnosticar?
```

Y no por bondad: si se busca culpable, la gente deja de contar lo que pasó, y sin información no hay corrección posible. Es una decisión práctica antes que ética.

Y el contenido mínimo:

```
cronología con marcas de tiempo
impacto medido: usuarios, duración, presupuesto de error consumido
cómo se detectó   ← y si fue por un cliente, eso es un hallazgo
qué se intentó y qué no funcionó
acciones, con dueño y fecha
```

Y la única medida honesta del proceso de revisión:

```
proporción de acciones completadas en plazo
  < 50 %  las revisiones son teatro
```

Y tres tipos de acción, en orden de valor:

```
1. que no pueda volver a ocurrir      lo mejor, y lo más caro
2. que se detecte en un minuto        casi siempre lo más rentable
3. que se mitigue más rápido          procedimiento, automatización
```

Y la pregunta obligatoria de la clase 125: ¿qué alerta debería haber sonado y no sonó?

Y la lista de comprobación de la clase:

- los criterios de declaración y gravedad están escritos
- cualquiera puede declarar y nadie es penalizado por hacerlo
- declarar abre canal, registro y papeles de forma automática
- los papeles se anuncian en voz alta, aunque los asuma una persona
- el mando coordina y no arregla
- un cambio cada vez, anunciado y reversible
- nada se toca fuera del canal
- existe catálogo de mitigaciones que no requieren saber la causa
- se captura evidencia antes de reiniciar
- comunicación externa con cadencia fija y sin prometer horas de resolución
- el relevo se traspasa explícitamente
- la revisión pregunta por qué el sistema lo permitió
- se mide la proporción de acciones completadas en plazo

Y el cierre que enlaza con la clase siguiente: la mitad de las acciones de una revisión son procedimientos que alguien tendrá que ejecutar de madrugada. Cómo se escriben para que sirvan, y cuáles conviene automatizar del todo, es la materia de la clase 128.

Ejemplo trabajado

CloudShop tenía incidentes sin proceso: se resolvían en un canal general, sin papeles y sin registro. El ejercicio compara dos incidentes casi idénticos, separados por cinco meses y por la implantación del proceso.

Incidente A, antes del proceso. Duración: 2 h 51.

```
09:14  suben los errores del flujo de compra
09:14  tres personas empiezan a mirar por su cuenta
09:21  alguien pregunta en el canal general si le pasa a alguien más
09:29  se reconoce que «esto es gordo»; no se declara nada formalmente
09:31  alguien reinicia el servicio de precios (sin anunciarlo)
09:33  alguien sube las réplicas del catálogo (sin anunciarlo)
09:34  alguien revierte un despliegue de la mañana (sin anunciarlo)
09:38  el sistema mejora un poco; nadie sabe por cuál de los tres
09:41  empeora otra vez
10:05  soporte lleva 50 min recibiendo llamadas sin información
10:12  comercio interrumpe al equipo por chat para preguntar
10:40  se descubre que el reinicio de las 09:31 borró un caché caliente
       y añadió carga a la base
11:30  se identifica la causa real: un cambio de interruptor a las 09:10
12:05  mitigado

cómo se detectó                    por la subida de errores, mirando
cambios simultáneos sin anunciar              3
```

```
comunicación externa                    ninguna hasta las 10:40
cronología                             reconstruida después, incompleta
revisión posterior                     una reunión de 30 min, 4 acciones
acciones completadas a los 3 meses          1 de 4
```

Lo que se implantó.

```
criterios de declaración y tres niveles de gravedad, escritos
declaración con un comando que abre canal, documento y avisa
cuatro papeles, anunciados en voz alta
regla de un cambio cada vez
catálogo de mitigaciones con enlaces directos
plantilla de comunicación externa con cadencia de 30 min
revisión sin culpa obligatoria en gravedad alta y media
```

Incidente B, cinco meses después. Mismo tipo de fallo. Duración: 19 minutos.

```
14:02  alerta de ritmo de consumo del presupuesto (clase 126)
14:03  declarado por quien estaba de guardia; gravedad alta
14:03  canal abierto, documento creado, avisos enviados
14:04  «asumo mando y comunicación; X en resolución»
14:05  primera pregunta: ¿qué cambió en los últimos 30 min?
       → la línea de cambios muestra un interruptor al 100 % a las 13:58
14:06  anuncio: «voy a revertir el interruptor recomendaciones-v4»
14:07  revertido
14:09  errores bajando; se confirma
14:12  primera comunicación externa, con impacto y próxima actualización
14:21  mitigado y comunicado

cambios aplicados                              1
cambios sin anunciar                           0
tiempo hasta la primera hipótesis              3 min
tiempo hasta la mitigación                    18 min
comunicación externa                    a los 10 min, y al cierre
cronología                             escrita durante, completa
```

La comparación.

```
tiempo hasta declarar                  nunca          1 min
tiempo hasta la primera hipótesis      2 h 16         3 min
cambios simultáneos                       3             1
cambios sin anunciar                      3             0
primera comunicación externa           1 h 26         10 min
interrupciones al equipo desde fuera      11             0
duración total                         2 h 51         19 min
presupuesto de error consumido           84 %          9 %
```

Y la parte del ahorro atribuible a cada cosa, según la revisión:

```
preguntar primero qué cambió              ~2 h
un cambio cada vez (evitó el reinicio dañino) ~35 min
comunicación con cadencia                 evitó 11 interrupciones
```

El incidente que no se declaró, y que enseñó por qué la regla 2 importa.

```
mes 3   una persona duda si declarar por una degradación del 4 %
        decide no declarar «por no molestar»
        3 h 20 después es una caída total
        presupuesto consumido: 100 % en un solo suceso
```

Y la corrección no fue reprender a nadie, sino cambiar el sistema:

```
se midió cuántos incidentes se declaraban y resultaban ser nada
  antes de la medida:     0
  objetivo declarado:     que hubiera algunos
  a los 6 meses:          7 de 34 declaraciones fueron falsas alarmas
```

Siete falsas alarmas se consideran una señal buena, no un problema: significa que la gente declara ante la duda. Y desde entonces no ha vuelto a haber una degradación de tres horas sin declarar.

Las revisiones y sus acciones.

```
revisiones hechas                      2 de 14        27 de 27
acciones creadas                           8             94
acciones con dueño y fecha               2 de 8        94 de 94
completadas en plazo                     1 de 8         79 de 94  (84 %)
revisiones que concluían «error humano»  2 de 2          0 de 27
```

Y el reparto de las noventa y cuatro acciones por tipo:

```
que no pueda volver a ocurrir                  21
que se detecte en un minuto                    48   ← 40 alertas nuevas
que se mitigue más rápido                      25
```

Cuarenta y ocho de noventa y cuatro fueron detección, y salieron de la pregunta obligatoria: ¿qué alerta debería haber sonado?

A los seis meses.

```
incidentes declarados                       14 (informal)     34
de ellos, falsas alarmas                          0            7
tiempo medio hasta declarar                  41 min          2 min
tiempo medio hasta mitigar                   1 h 52         14 min
cambios simultáneos por incidente              2,4            1,0
incidentes con comunicación externa          3 de 14        27 de 27
interrupciones al equipo durante incidentes  11 / incidente   0,3
revisiones con acciones y dueño              2 de 14        27 de 27
acciones completadas en plazo                 13 %            84 %
```

La lección que esta clase traslada a la parte 10: el tiempo de mitigación pasó de 1 h 52 a 14 minutos y nada de esa mejora vino de resolver más rápido. Vino de declarar antes, de preguntar primero qué había cambiado y de aplicar un cambio cada vez en lugar de tres. Y el dato que más dice sobre el proceso es el que parece un fallo: siete de treinta y cuatro declaraciones no eran nada, y esa proporción es exactamente lo que garantiza que no se repita la degradación de tres horas que nadie se atrevió a declarar.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-10-observability-sre-reliability/127-incidentes-severidad-comando-y-comunicacion/
lab.py
```

El laboratorio selecciona el motor de práctica incident 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 plan-incidente 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 cronología, roles, comunicación y aprendizaje. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye plan-incidente 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 incidentes tardan mucho en reconocerse como tales	Declarar se percibe como molestar o como admitir un fallo	Cualquiera puede declarar, ante la duda se declara, y las falsas alarmas no se penalizan ni se comentan.
El sistema mejora y nadie sabe por qué	Se aplicaron varios cambios a la vez sin anunciarlos	Un cambio cada vez, anunciado en el canal, con forma de deshacerlo, y nada se toca fuera del canal.
El equipo trabaja bien y recibe interrupciones constantes desde fuera	No hay papel de comunicación ni cadencia fija	Asigna comunicación desde el principio y publica actualizaciones cada 30 minutos aunque no haya novedades.
Se dedica una hora a entender la causa mientras los usuarios sufren	Se diagnostica antes de mitigar	Mitiga con el catálogo que no requiere saber la causa, empezando por revertir el cambio más reciente.
Después del incidente no se puede explicar qué pasó	La cronología se intentó reconstruir al día siguiente, y se reinició sin capturar evidencia	Papel de cronología escribiendo durante, y captura de estado antes de reiniciar nada.
Las revisiones no cambian nada	Las acciones no tienen dueño ni fecha, o la conclusión fue que alguien se equivocó	Pregunta por qué el sistema lo permitió, asigna dueño y fecha a cada acción y mide la proporción completada en plazo.

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 declarar pronto sale más barato que no declarar, y qué regla lo hace posible?
2. ¿Qué dos fallos concretos evitan los papeles durante un incidente?
3. ¿Por qué se mitiga antes de diagnosticar y con qué catálogo?
4. ¿Qué se promete en una comunicación externa y qué no se promete nunca?
5. ¿Cuál es la única medida honesta de que las revisiones sirven para algo?

Referencias

- Google SRE (2025). Managing incidents — papeles, mando y separación entre coordinar y resolver. <https://sre.google/sre-book/managing-incidents/>
- Google SRE (2025). Postmortem culture: learning from failure — revisión sin culpa y acciones. <https://sre.google/sre-book/postmortem-culture/>
- PagerDuty (2025). Incident response documentation — severidad, comunicación y relevos. <https://response.pagerduty.com/>
- Allspaw, J. (2012). Blameless postmortems and a just culture — por qué buscar culpables destruye la información. <https://www.etsy.com/codeascraft/blameless-postmortems/>
- Atlassian (2025). Incident communication best practices — cadencia fija y contenido de las actualizaciones. <https://www.atlassian.com/incident-management/incident-communication>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 10 en PDF · Manual integral

Evaluación — 127 Incidentes, severidad, comando y comunicación

Preguntas

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

Reto verificable

Entrega plan-incidente 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.

128 — Runbooks, playbooks y automatización operativa

Parte: 10 — Observabilidad, SRE y confiabilidad Nivel: avanzado · Horas estimadas: 4 Laboratorio: operations · Estado: EXECUTABLECORE

Propósito

Escribir los procedimientos que la clase 125 exigía para que una alerta fuera accionable, y decidir cuáles conviene automatizar. La clase parte de un criterio de diseño concreto —se escribe para quien tiene menos contexto, a las tres de la madrugada— y sostiene dos ideas que suelen sorprender: que automatizar un procedimiento raro y peligroso suele ser peor que documentarlo, porque la automatización sin usar se pudre igual que la documentación pero falla en el peor momento; y que toda reparación automática necesita un contador y un límite, o acabará tapando un problema que empeora.

Resultados de aprendizaje

Al finalizar podrás:

1. Escribir un procedimiento utilizable por quien no lo escribió.
2. Mantenerlo vivo con mecanismos que no dependan de la buena voluntad.
3. Situar cada tarea en la escala de automatización con un criterio explícito.
4. Acotar las reparaciones automáticas para que no oculten deterioro.
5. Medir el trabajo repetitivo y reducirlo por donde más pesa.

Conceptos centrales

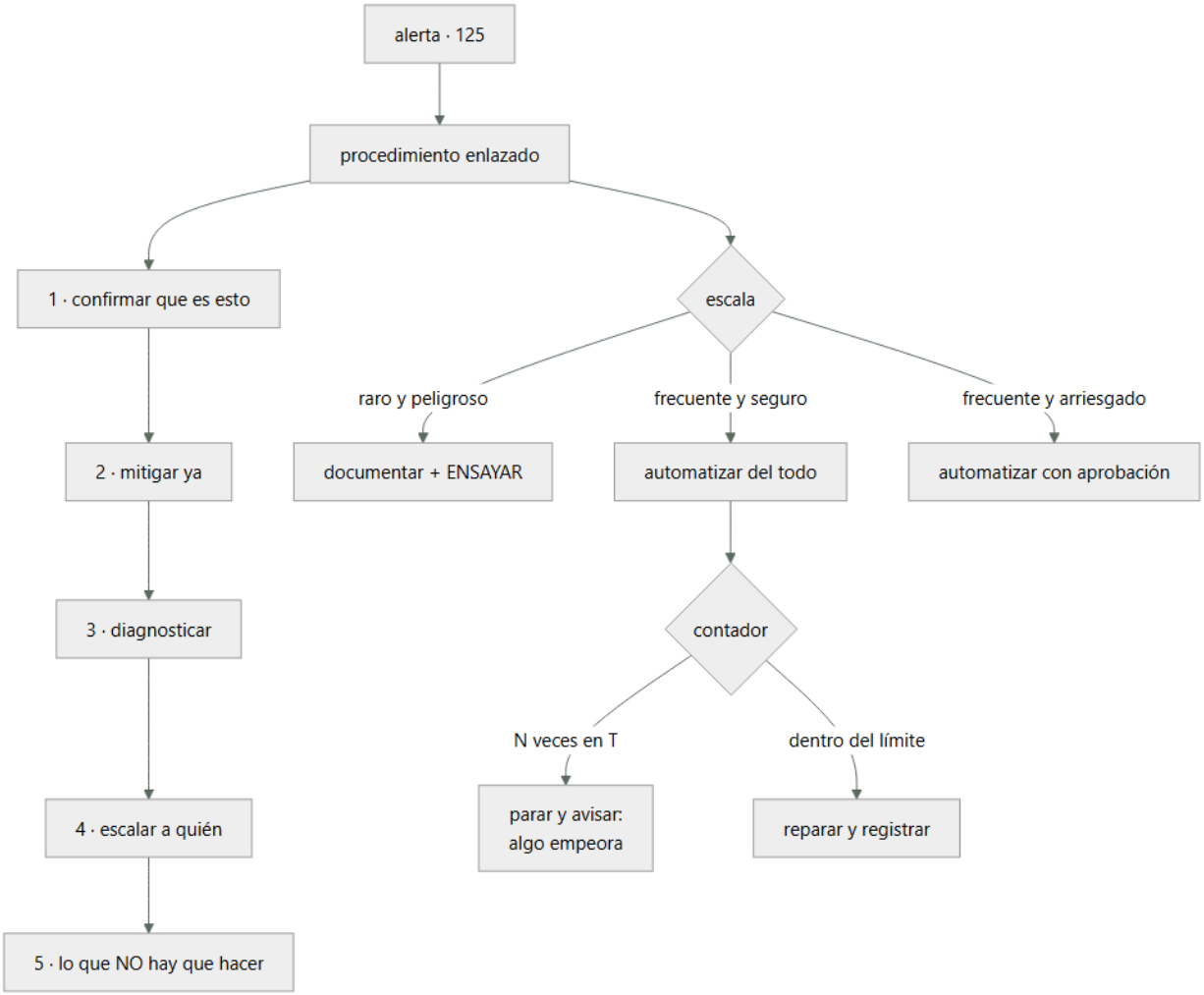
Concepto	Comprensión verificable
procedimiento	Guía para responder a una situación concreta. Su calidad se mide por si sirve a quien no lo escribió, no por su detalle.
prueba de las tres de la madrugada	Que alguien que no lo escribió lo ejecute en un ensayo. Es la única comprobación válida.
escala de automatización	Documentar, guionizar, automatizar con aprobación, automatizar del todo. Se sube según frecuencia y riesgo.
reparación automática	Acción correctiva sin intervención humana. Necesita contador y límite, o esconde el deterioro.
trabajo repetitivo	Manual, repetido, automatizable, sin valor duradero y que crece con el sistema. Se mide como proporción del tiempo.
lo que no se automatiza	Lo que exige juicio, lo que puede destruir datos y lo que no se puede probar.

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

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. Escrito para quien tiene menos contexto

Un procedimiento no se escribe para quien conoce el sistema: se escribe para quien está de guardia a las tres de la madrugada, no construyó ese servicio y acaba de despertarse.

De ahí salen las reglas de redacción:

```
comandos exactos, copiables, sin variables que haya que deducir
lo que se espera ver al ejecutarlos, para saber si va bien
sin siglas ni nombres internos sin explicar
sin «obviamente» ni «basta con»
rutas y enlaces completos, no «en el panel de siempre»
```

Y la estructura que funciona, en este orden:

```
1. ¿ES ESTO?
   cómo confirmar en dos minutos que la situación es la que dice el título
   → evita aplicar el procedimiento equivocado, que es peor que no aplicar
     ninguno

2. MITIGAR
   lo primero, siempre; con el catálogo de la clase 127

3. DIAGNOSTICAR
   consultas concretas, enlaces a los paneles, qué mirar y en qué orden

4. ESCALAR
   a quién, cómo se le localiza y con qué información

5. LO QUE NO HAY QUE HACER
```

```
    la sección más útil y la que casi nunca está
```

Y la quinta merece detalle porque evita el daño irreversible:

```
«no reinicies la base: la conmutación tarda 60 s y el reinicio no arregla esto»
«no borres la cola: contiene pedidos sin procesar»
«no subas el límite de conexiones: empeora, ver clase 109»
«no ejecutes la migración a medias: no es reversible»
```

Y el largo, que decide si se usa:

```
una pantalla para mitigar
lo demás debajo
→ si hay que leer tres páginas antes de la primera acción, no se usará
```

La prueba que valida un procedimiento es una sola, y no es revisarlo:

```
que alguien que NO lo escribió lo ejecute, en un ensayo,
sobre un entorno donde se pueda
y que anote cada punto donde tuvo que preguntar
→ cada pregunta es un defecto del procedimiento
```

Casi siempre aparecen tres cosas: un permiso que quien lo escribió tenía y el resto no, un nombre que solo conoce el autor, y un paso que en realidad son cuatro.

2. Que no se pudran

Un procedimiento escrito hace dos años describe un sistema que ya no existe, y no da ningún error. Es la ley 13 en su versión documental.

Los mecanismos que lo evitan, y ninguno depende de que alguien se acuerde:

```
ENLAZADO DESDE LA ALERTA
  si no se encuentra en cinco segundos, no existe
  → la alerta lleva el enlace, y una alerta sin enlace no se aprueba

CORREGIDO AL USARSE
  quien lo ejecuta lo arregla en el momento, no «lo apunta para luego»
  → y esa corrección es parte de cerrar el incidente

CON FECHA VISIBLE
  «última verificación: 2026-06-14»
  → y una revisión cuando pasa de N meses

BORRADO CON SU ALERTA
  si la alerta desaparece, el procedimiento también
  → 315 alertas borradas en la clase 125 dejaron procedimientos huérfanos

VERSIONADO CON EL CÓDIGO
  vive en el repositorio del servicio, no en un documento suelto
  → así el cambio que lo invalida y su corrección van juntos
```

La segunda es la que más funciona en la práctica y la que hay que exigir: la persona que acaba de sufrir el procedimiento es la única que sabe exactamente qué le faltaba, y dentro de dos días ya no se acordará.

Y una medida sencilla del estado del conjunto:

```
procedimientos sin verificar en 6 meses
alertas sin procedimiento enlazado
procedimientos ejecutados en el último trimestre
  → los que nunca se ejecutan son candidatos a ensayo o a borrado
```

Y una distinción que ordena la biblioteca:

```
PARA UN SÍNTOMA CONCRETO   «errores 5xx en el flujo de compra»
                           es lo que enlaza una alerta
PARA UNA TAREA             «rotar la clave del proveedor de pago»
                           no responde a una alerta; se ejecuta a propósito
DE REFERENCIA              cómo está montado esto, dónde está cada cosa
                           no es un procedimiento: es documentación
```

Mezclarlos produce documentos de veinte páginas que no sirven para ninguna de las tres cosas.

3. La escala de automatización

```
1. DOCUMENTADO       una persona lee y ejecuta
2. GUIONIZADO        una persona ejecuta un guion que hace los pasos
3. CON APROBACIÓN    el sistema propone y una persona confirma
4. AUTOMÁTICO        el sistema actúa y avisa si no lo consigue
```

Y el criterio para subir un escalón, que son dos variables:

```
frecuente           automatizar del todo   automatizar con aprobación
raro                guionizar              DOCUMENTAR y ENSAYAR
```

Y la casilla de abajo a la derecha es la que se hace mal más a menudo. La tentación es automatizar el procedimiento raro y peligroso, y es mala idea:

```
se ejecuta dos veces al año
→ la automatización no se prueba en ese tiempo
→ el sistema cambia y ella no
→ y falla justo cuando se necesita, con permisos amplios y en el peor momento
```

Lo que sí funciona en esa casilla: documentar bien y ensayar cada trimestre, que es lo que la clase 131 convertirá en rutina.

Y tres cosas que no se automatizan, sea cual sea la frecuencia:

```
lo que exige juicio            decidir si se acepta un riesgo, comunicar
                               a un cliente, elegir entre dos males
lo que puede destruir datos    borrados, migraciones destructivas,
                               restauraciones sobre datos vivos
lo que no se puede probar      si no hay forma de ensayarlo, la
                               automatización es un riesgo, no un ahorro
```

Y una recomendación intermedia muy rentable: automatizar el diagnóstico y dejar la acción a la persona. Un guion que reúne en veinte segundos todo lo que habría que mirar a mano ahorra la mayor parte del tiempo sin ninguno de los riesgos.

```
# diagnóstico automático adjunto a la alerta
$ diagnostico pedidos --ultimos 30m
  cambios recientes .......... interruptor pago-v2 al 100 % hace 12 min
  saturación ................. agrupador 41 %, cola 12 s
  dependencias ............... precios 99,9 %, pago 94,1 %  ← degradado
  errores por tipo ........... 71 % tiempo de espera hacia pago
  sugerencia ................. revisar el cambio de interruptor
```

Eso se adjunta al aviso y quien está de guardia empieza con el trabajo hecho.

4. Reparación automática y trabajo repetitivo

La reparación automática es el escalón cuatro, y tiene un peligro específico: oculta el deterioro.

```
el servicio se queda sin memoria y se reinicia solo
→ nadie se entera
→ y la fuga de memoria empeora durante meses
→ hasta que reiniciar cada 40 minutos ya no basta
```

La regla que lo evita, y que hay que aplicar a toda reparación automática:

```
CONTADOR Y LÍMITE
  se cuenta cada actuación
  si supera N veces en T, deja de reparar y avisa
  → «esto se ha reparado solo 14 veces hoy» es la información valiosa
```

Y las tres cosas que hay que registrar siempre:

```
cuántas veces actuó, por causa
si lo consiguió
y la tendencia: si sube, algo empeora aunque nadie lo note
```

Y los ejemplos que este programa ya tiene repartidos:

```
reinicio por sonda fallida                    clase 079
reintento con espera creciente                clase 113
autoescalado                                  clases 078, 117
reversión automática por análisis de canario  clase 102
reconciliación que corrige la deriva          clase 103
```

Los cinco reparan solos. Los cinco pueden estar tapando algo, y por eso los cinco necesitan su contador.

El trabajo repetitivo, que es el objetivo de fondo de esta clase. Su definición tiene cinco rasgos, y hacen falta varios:

```
manual · repetitivo · automatizable · sin valor duradero ·
y crece con el tamaño del sistema
```

El último es el decisivo: si al doblar el número de servicios ese trabajo se dobla, es trabajo repetitivo; si no crece, probablemente sea trabajo normal.

Y se mide, aunque sea a mano:

```
proporción del tiempo del equipo dedicada a ello
  > 50 %   no queda tiempo para arreglarlo, y la situación se agrava sola
  ~ 30 %   habitual
  < 20 %   objetivo razonable
```

Y se reduce por donde pesa, no por donde es más fácil:

```
registrar cada tarea repetitiva durante dos semanas
ordenar por tiempo total: frecuencia × duración
automatizar las tres primeras
→ suelen ser el 60 % del total
```

Y una advertencia: automatizar el síntoma perpetúa la causa. Un guion que crea usuarios en cuatro sistemas ahorra tiempo, y la pregunta mejor es por qué hay cuatro sistemas de usuarios.

Y la lista de comprobación de la clase:

- toda alerta enlaza su procedimiento
- cada procedimiento empieza confirmando que es esa situación
- la mitigación cabe en una pantalla
- hay sección de lo que no hay que hacer
- los comandos son exactos y se dice qué se espera ver
- alguien que no lo escribió lo ha ejecutado en un ensayo
- quien lo usa lo corrige en el momento
- lleva fecha de última verificación y se revisa
- vive en el repositorio del servicio
- cada tarea está situada en la escala por frecuencia y riesgo
- lo raro y peligroso está documentado y se ensaya, no automatizado
- el diagnóstico automático se adjunta a las alertas
- toda reparación automática tiene contador, límite y registro
- se mide la proporción de tiempo en trabajo repetitivo

Y el cierre que enlaza con la clase siguiente: varios de estos procedimientos existen porque el sistema se queda sin capacidad. Saber cuánta hay, cuándo se agota y cómo comprobarlo antes de que ocurra es la materia de la clase 129.

Ejemplo trabajado

CloudShop escribe procedimientos para las setenta y tres alertas que sobrevivieron a la clase 125. El ejercicio tiene tres hallazgos: lo que la prueba de ejecución reveló, la reparación automática que llevaba cuatro meses tapando una fuga, y de dónde salía el trabajo repetitivo.

Hallazgo 1: la prueba de las tres de la madrugada.

Se escribieron 73 procedimientos. Después, cada uno lo ejecutó alguien distinto de quien lo escribió, sobre preproducción.

```
procedimientos escritos                          73
ejecutados por otra persona en el ensayo         73
completados sin preguntar nada                    9
con al menos una pregunta                        64
preguntas totales                               211
```

Y las preguntas, clasificadas:

```
faltaba un permiso que el autor tenía            58
nombre o sigla que solo conocía el autor         47
un paso que en realidad eran varios              39
no se decía qué se esperaba ver                  31
el enlace llevaba a un panel que ya no existía   22
el comando tenía una variable sin explicar       14
```

Cincuenta y ocho problemas de permisos. Los procedimientos eran correctos y la mitad del equipo no podía ejecutarlos. Eso se arregló antes que ningún texto.

```
completados sin preguntar                    9 de 73          68 de 73
tiempo medio de ejecución                    22 min            6 min
los 5 restantes                          requerían juicio: se marcaron
                                         como «escalar siempre»
```

Hallazgo 2: el reinicio automático que tapaba una fuga.

Al instrumentar contadores en las reparaciones automáticas, el primer día:

```
reinicios automáticos por sonda de memoria, últimas 24 h        4.180
servicios afectados                                                 3
del servicio de recomendaciones                                 3.940
frecuencia                                                  cada 22 s
```

Cuatro mil ciento ochenta reinicios en un día, y el panel del servicio estaba verde, porque el reinicio funcionaba: la disponibilidad medida era del 99,7 %.

```
tiempo que llevaba ocurriendo                                4 meses
causa                       fuga de memoria introducida en la versión 3.1
cómo se detectó antes                                       no se detectó
efecto colateral        arranque en frío constante; latencia p99 ×3
                        y el caché local nunca se llenaba
```

Y con contador y límite:

```
contador de reparaciones automáticas            no            sí
límite                                      sin límite   5 en 30 min
qué ocurre al superarlo                        nada      deja de reparar
                                                         y avisa
tiempo hasta detectar una fuga nueva         4 meses         38 min
reinicios diarios tras corregir la fuga       4.180            2
latencia p99 del servicio                      840 ms         210 ms
```

Y se aplicó el mismo tratamiento a las otras cuatro reparaciones automáticas:

```
reintentos hacia el proveedor de pago        14.200      un 4 % de errores
                                                         intermitentes
                                                         crónicos
autoescalado                                    41       un trabajo por lotes
                                                         mal programado
reversión automática de canario                 0,3      correcto
reconciliación corrigiendo deriva                 9      alguien tocaba a mano
                                                         un recurso cada día
```

Tres de las cuatro estaban tapando algo.

Hallazgo 3: de dónde salía el trabajo repetitivo.

Dos semanas registrando cada tarea manual:

```
dar acceso a alguien a algo                    38        22        14 h
reiniciar o desbloquear un consumidor          61         8         8 h
reprocesar mensajes de la cola de fallidos     22        35        13 h
crear un entorno para depurar                  11        40         7 h
rotar una credencial                            6        50         5 h
responder «¿por qué este pedido está así?»     44        12         9 h
el resto (31 tareas distintas)                  —         —        18 h
                                                                  ■■■■■
                                                                  74 h
proporción del tiempo del equipo                                   34 %
```

Y las tres primeras eran el 47 % del total. Su tratamiento, con el criterio del apartado tercero:

```
dar acceso           frecuente, riesgo medio → automatizar con aprobación
                     grupos del catálogo (clase 106)
                     14 h/mes → 1 h/mes

desbloquear consumidor  frecuente, riesgo bajo → automatizar del todo
                     con contador y límite
                     8 h/mes → 0, y el contador reveló que 41 de los 61
                     eran el mismo defecto, que se corrigió

reprocesar fallidos  frecuente, riesgo medio → automatizar con aprobación
                     la herramienta por lotes de la clase 113
                     13 h/mes → 2 h/mes

«¿por qué este pedido está así?»  → no era trabajo repetitivo:
```

```
                    era falta de la vista por sujeto (clase 115)
                    9 h/mes → 0,5 h/mes
```

Y una que se decidió no automatizar:

```
rotar una credencial       6 veces al mes, riesgo alto
decisión                   guionizar los pasos, no automatizar el conjunto
motivo                     toca permisos de producción y una rotación mal
                           hecha corta el servicio
añadido                    ensayo trimestral con alguien distinto
```

El diagnóstico automático adjunto a las alertas.

```
información al recibir un aviso            el título     título + diagnóstico
tiempo hasta la primera hipótesis            9 min          80 s
avisos resueltos sin abrir ningún panel        0 %           41 %
```

A los seis meses.

```
alertas con procedimiento enlazado            12 de 73      73 de 73
procedimientos ejecutables por cualquiera      9 de 73      68 de 73
tiempo medio de ejecución                     22 min         6 min
procedimientos sin verificar en 6 meses          —             4
reparaciones automáticas con contador         0 de 5        5 de 5
fugas o defectos tapados por reparación
  automática, descubiertos                       —             3
tiempo hasta detectar una fuga nueva          4 meses        38 min
trabajo repetitivo, proporción del tiempo       34 %          11 %
horas al mes en tareas manuales                   74            24
tareas automatizadas                               0             4
tareas deliberadamente no automatizadas           —             1
```

La lección que esta clase traslada a la parte 10: el hallazgo más caro no lo dio ningún procedimiento nuevo, sino poner un contador a algo que ya funcionaba. Cuatro mil ciento ochenta reinicios automáticos al día durante cuatro meses, con el panel en verde y la disponibilidad cumpliendo el objetivo: la reparación automática estaba haciendo su trabajo tan bien que nadie podía ver que el sistema se estaba deteriorando. Y la mayor reducción de trabajo repetitivo no vino de automatizar: nueve horas al mes desaparecieron al construir una consulta que la clase 115 ya había pedido.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-10-observability-sre-reliability/128-runbooks-playbooks-y-automatizacion-operativa/lab.py
```

El laboratorio selecciona el motor de práctica operations 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 runbook-verificado 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 runbook probado por otra persona. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye runbook-verificado 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 procedimiento existe y quien está de guardia no puede ejecutarlo	Está escrito por quien tenía permisos y contexto que el resto no tiene	Prueba de ejecución por otra persona; cada pregunta que hace es un defecto, y los permisos se arreglan antes que el texto.
Los procedimientos describen un sistema que ya no existe	Ley 13: un documento desactualizado no produce ningún error	Enlazado desde la alerta, corregido por quien lo usa en el momento, con fecha de verificación y viviendo en el repositorio del servicio.
Una automatización crítica falla justo cuando se necesita	Se automatizó un procedimiento raro, que no se ejecuta lo bastante como para detectar que se pudrió	Lo raro y peligroso se documenta y se ensaya cada trimestre; lo frecuente y seguro se automatiza.
Un servicio se deteriora durante meses con los paneles en verde	Una reparación automática lo compensa y nadie cuenta cuántas veces actúa	Contador, límite y registro en toda reparación automática; al superar el límite, deja de reparar y avisa.
El equipo no tiene tiempo de mejorar nada	El trabajo repetitivo supera la mitad del tiempo y crece con el sistema	Registra dos semanas, ordena por tiempo total y automatiza las tres primeras tareas; suelen ser la mitad.
Se automatiza una tarea y el problema de fondo sigue creciendo	Se automatizó el síntoma	Antes de guionizar, pregunta por qué existe esa tarea; a veces la respuesta elimina la tarea entera.

■ 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. ¿Para quién se escribe un procedimiento y qué prueba lo valida?
2. ¿Por qué la sección de lo que no hay que hacer es tan valiosa?
3. ¿Qué criterio decide si una tarea se automatiza, y qué casilla es la trampa?
4. ¿Por qué toda reparación automática necesita contador y límite?
5. ¿Qué cinco rasgos definen el trabajo repetitivo y cuál es el decisivo?

Referencias

- Google SRE (2025). Eliminating toil — definición, medición y objetivo de reducción. <https://sre.google/sre-book/eliminating-toil/>
- Google SRE (2025). Emergency response and playbooks — contenido y mantenimiento de los procedimientos. <https://sre.google/sre-book/emergency-response/>
- PagerDuty (2025). Runbook documentation — estructura y enlace desde la alerta. <https://response.pagerduty.com/oncall/runbooks/>

- Beyer, B. y otros (2018). The Site Reliability Workbook, cap. 6 — escala de automatización y sus riesgos. <https://sre.google/workbook/eliminating-toil/>
- Woods, D. y otros (2010). Behind Human Error — por qué los automatismos que compensan ocultan el deterioro. <https://www.taylorfrancis.com/books/mono/10.1201/9781315568935/behind-human-error>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 10 en PDF · Manual integral

Anterior	Índice	Siguiente
← 127 · Incidentes, severidad, comando y comunicación	Parte 10 · Programa	129 · Capacidad, rendimiento y pruebas de carga →

Evaluación — 128 Runbooks, playbooks y automatización operativa

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 runbook-verificado 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.

129 — Capacidad, rendimiento y pruebas de carga

Parte: 10 — Observabilidad, SRE y confiabilidad Nivel: avanzado · Horas estimadas: 4 Laboratorio: performance · Estado: EXECUTABLECORE

Propósito

Saber cuánta capacidad hay, cuándo se agota y comprobarlo antes de que ocurra. La clase se apoya en una propiedad de las colas que explica la mitad de los incidentes de este programa —la latencia no crece de forma proporcional al uso: se dispara cerca de la saturación— y en un defecto de método que hace que la mayoría de las pruebas de carga mientan: si el generador espera la respuesta antes de enviar la siguiente petición, oculta precisamente la degradación que se buscaba medir.

Resultados de aprendizaje

Al finalizar podrás:

1. Separar rendimiento, capacidad y escalabilidad, y saber cuál se está midiendo.
2. Explicar por qué la latencia se dispara cerca de la saturación y qué margen dejar.
3. Elegir el tipo de prueba según la pregunta.
4. Generar carga sin ocultar la degradación ni falsear la distribución.
5. Planificar capacidad contando los límites que no escalan solos.

Conceptos centrales

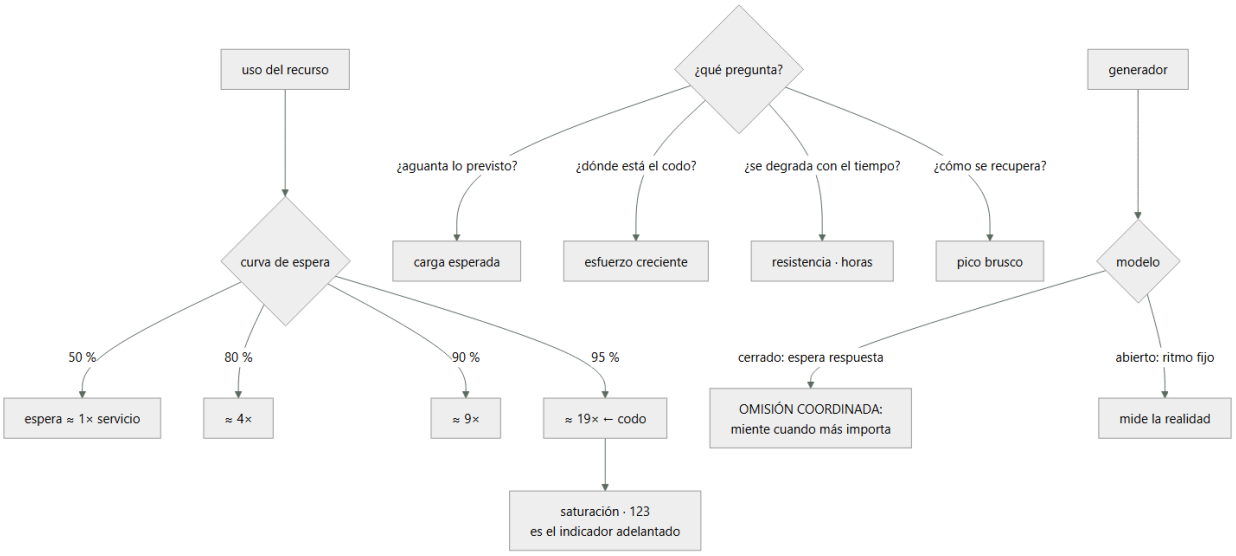
Concepto	Comprensión verificable
rendimiento	Lo que tarda una operación cuando el sistema está tranquilo. Es una propiedad del código y del camino.
capacidad	Cuánto trabajo simultáneo admite antes de degradarse. Es una propiedad del sistema entero.
codo	Nivel de uso a partir del cual la latencia se dispara. Está bastante antes del 100 %, y es el límite real.
modelo abierto	El generador envía peticiones a un ritmo fijo, responda el sistema o no. Es lo que hacen los usuarios reales.
omisión coordinada	Defecto del modelo cerrado: al esperar cada respuesta, se dejan de enviar peticiones justo cuando el sistema va lento, y la latencia medida sale mucho mejor de lo que es.
prueba de resistencia	Carga sostenida durante horas o días. Es la que encuentra fugas, acumulaciones y degradación lenta.
margen	Distancia entre la carga habitual y el codo. Es lo que da tiempo a reaccionar.

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

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

■ Flujo de razonamiento



Desarrollo

1. Tres cosas distintas, y la curva que las une

```
RENDIMIENTO    cuánto tarda una operación sin competencia
               se mejora con código, consultas, índices, caché
CAPACIDAD      cuánto trabajo simultáneo admite antes de degradarse
               se mejora con recursos, paralelismo y quitando cuellos
ESCALABILIDAD  cómo cambian las dos al añadir recursos
               puede ser buena, mala o negativa
```

Y el error habitual es optimizar la primera cuando el problema es la segunda: una consulta de 4 ms no mejora nada si la petición espera 800 ms para conseguir un hilo, que es exactamente el hueco de la clase 124.

La curva de espera es lo que hay que tener en la cabeza, porque explica casi todo:

```
uso del recurso        espera relativa al tiempo de servicio
  10 %                   0,11×
  50 %                   1×
  70 %                   2,3×
  80 %                   4×
  90 %                   9×
  95 %                  19×
  99 %                  99×
```

Y las consecuencias prácticas:

```
entre el 50 % y el 70 % apenas se nota nada
entre el 90 % y el 95 % la latencia se DOBLA
→ el sistema pasa de «bien» a «caído» sin estados intermedios
→ y por eso el uso medio es un indicador engañoso y la saturación no
```

Y de ahí sale la regla de margen:

```
planificar para el 60-70 % de uso en el pico habitual
→ no por prudencia vaga, sino porque el margen es el TIEMPO
  que hay para reaccionar
```

Y dos matices que la hacen más útil:

```
la variabilidad empeora la curva
  con tiempos de servicio muy dispares, el codo llega antes
el paralelismo la mejora
  varios servidores idénticos toleran más uso que uno solo
```

Y la ley que relaciona las tres magnitudes, la misma que la clase 117 usó para las funciones:

```
trabajo en curso = ritmo de llegada × tiempo de permanencia

1.200 peticiones/s × 0,25 s = 300 en curso
→ y si solo hay 200 hilos, 100 esperan
```

Esa cuenta, hecha antes, evita la mayoría de los incidentes de saturación.

2. Qué prueba responde qué

```
HUMO            carga mínima; comprueba que el montaje funciona
                se ejecuta en cada despliegue

CARGA ESPERADA  ¿aguanta lo previsto con la latencia del objetivo?
                → responde sí o no; no busca el límite

ESFUERZO CRECIENTE  sube hasta que se degrada
                → encuentra el CODO y el cuello de botella
                → y el cuello siempre está en un sitio: conexiones, hilos,
                  una dependencia, un bloqueo

RESISTENCIA     la carga esperada durante horas o días
                → encuentra fugas de memoria, acumulación de conexiones,
                  crecimiento de tablas, colas que no vacían

PICO BRUSCO     de 0 a la carga máxima en segundos
                → mide cuánto tarda en escalar y CÓMO SE RECUPERA
                → aquí aparecen las tormentas de reintentos (clase 113)
```

Y la que más valor da en un sistema como el de las partes anteriores es la de resistencia, aunque sea la que menos se hace:

```
los problemas de la parte 09 que solo aparecen con el tiempo
  fuga de memoria oculta por reinicios          clase 128
  conexiones que no se devuelven al agrupador   clase 109
  historial que crece y ralentiza               clase 119
  cola que crece despacio                       clase 113
  tabla de salida sin limpiar                   clase 116
```

Ninguno de esos se ve en veinte minutos de carga. Todos se ven en ocho horas.

Y lo que hay que medir durante la prueba, además de latencia y errores:

```
memoria y su tendencia, no su valor
conexiones en uso y en espera
profundidad y antigüedad de colas
tiempo de recolección de memoria
saturación de todos los agrupadores
y la línea de cambios, por si alguien despliega a la vez
```

La segunda línea es la clave de una prueba de resistencia: lo que importa es la pendiente, no el valor. Una memoria que sube 30 MB por hora está bien a las dos horas y mal a las cuarenta.

Y las dos preguntas que hay que responder antes de ejecutar nada:

```
¿qué hipótesis estoy comprobando?
¿qué resultado me haría cambiar una decisión?
→ si no hay respuesta, la prueba producirá gráficos y ninguna acción
```

3. Generar carga sin mentirse

Aquí está el defecto de método que invalida la mayoría de las pruebas.

```
MODELO CERRADO   N clientes; cada uno envía, ESPERA la respuesta y repite
MODELO ABIERTO   se envían R peticiones por segundo, responda o no
```

Y el problema del primero:

```
el sistema se pone lento
→ los clientes tardan más en recibir
→ y por tanto ENVÍAN MENOS
→ la carga baja sola justo cuando el sistema está mal
→ y la latencia medida sale mucho mejor que la real
```

Eso se llama omisión coordinada, y el efecto es grande:

```
sistema que se para 1 s cada 10 s, con 100 peticiones/s previstas
  modelo cerrado, percentil 99 medido:      ~30 ms
  modelo abierto, percentil 99 real:      ~1.000 ms
```

Un factor de treinta. Y el usuario real vive el segundo caso, porque el usuario no deja de pulsar porque el sistema vaya lento; llega más gente, no menos.

La corrección es usar modelo abierto, o corregir el registro compensando el retraso acumulado.

Y los otros cuatro errores que falsean una prueba:

```
DATOS IRREALES
  base vacía o con mil filas: los planes de consulta no son los de producción
  → es la lección de las clases 104 y 109

DISTRIBUCIÓN UNIFORME
  pedir productos al azar entre dos millones
  → nunca aparece la clave caliente de la clase 110 ni el caché real
  → la proporción de aciertos de caché de la prueba no se parece a nada

SIN TIEMPO DE ESPERA ENTRE ACCIONES
  un usuario real piensa entre pantallas
  → sin eso, el perfil de carga no es el de nadie

SUBIDA DE GOLPE cuando se quiere medir estado estable
  → mejor rampa, y esperar a que se estabilice antes de medir
  → salvo que la pregunta sea justamente cómo reacciona a un pico
```

Y el más importante de los cuatro es el segundo: la distribución decide la proporción de aciertos de caché, y esa decide casi todo lo demás.

Dónde probar. Un entorno de prueba nunca es igual, y hay tres opciones honestas:

```
ENTORNO APARTE       barato, y sus resultados son orientativos
                     → sirve para comparar versiones entre sí
TRÁFICO EN ESPEJO    copia del tráfico real a un entorno paralelo,
                     sin efectos
                     → realismo alto y sin riesgo para el usuario
EN PRODUCCIÓN        la única medida verdadera
                     → con límites, en horas valle, con interruptor
                       de parada y vigilando el presupuesto de error
```

Y una forma barata y muy usada de la tercera: reducir el número de instancias en horas valle hasta acercarse al codo, medir, y devolverlas. Da el límite real sin generar nada.

4. Planificar, y lo que no escala solo

La planificación de capacidad es una cuenta con tres factores:

```
capacidad necesaria =
    demanda prevista
  × margen hasta el codo
  + lo que tarde en estar disponible
```

Y el tercero es el que se olvida:

```
añadir instancias                    segundos o minutos
subir una cuota del proveedor        horas o días
ampliar particiones de un registro   semanas (clase 114)
contratar una licencia               semanas
migrar una clave de partición        semanas (clase 110)
```

Y de ahí la regla: lo que tarda semanas en ampliarse se planifica con meses de antelación, no cuando la métrica se acerque al límite.

Y la lista de lo que no escala solo aunque todo lo demás sí, que es donde se rompen los sistemas de este programa:

```
conexiones a la base                      clase 109
cuotas de la cuenta o del proyecto        clases 049, 117
concurrencia compartida de funciones      clase 117
particiones de un registro                clase 114
límites de ritmo de terceros              clase 118
un trabajo por lotes con un solo ejecutor
la persona que aprueba algo
```

Las dos últimas se olvidan siempre, y las dos han aparecido en este programa.

Y qué vigilar para no llegar tarde:

```
uso frente al codo conocido, no frente al 100 %
proyección: a este ritmo de crecimiento, ¿cuándo se alcanza?
tiempo de aprovisionamiento de cada recurso
y una revisión antes de cada evento previsible: campañas, cierres,
  temporadas
```

La segunda es la que convierte esto en algo útil: una alerta que dice «al ritmo actual, esto se agota en 12 días» es accionable; una que dice «uso al 71 %» no.

Coste y eficiencia, que es la otra cara:

```
coste por petición, o por pedido, o por unidad de negocio
→ es la medida que permite comparar optimizaciones
```

Y el criterio para decidir si vale la pena optimizar:

```
coste de la ingeniería frente a coste del recurso
  duplicar instancias: 400 €/mes
  dos semanas de trabajo: mucho más
  → si el problema es puntual, se paga la máquina
  → si crece con el tamaño, se paga la ingeniería
```

Y la excepción: cuando lo que falta no es potencia sino un cuello de botella, añadir recursos no arregla nada. Cien instancias más contra una base saturada empeoran la situación, como demostró la clase 113.

Y la lista de comprobación de la clase:

- ■ está identificado el codo de cada servicio, medido
- ■ el pico habitual queda por debajo del 70 % del codo
- ■ se vigila saturación, no solo uso medio
- ■ el generador de carga usa modelo abierto
- ■ los datos de prueba tienen volumen y distribución realistas
- ■ hay prueba de resistencia de al menos varias horas, con tendencias
- ■ hay prueba de pico que mide la recuperación, no solo el aguante
- ■ cada prueba responde a una hipótesis escrita
- ■ está inventariado lo que no escala solo y su tiempo de ampliación
- ■ hay proyección de agotamiento, no solo uso actual
- ■ se mide coste por unidad de negocio

Y el cierre que enlaza con la clase siguiente: conocer el codo permite dimensionar, y no evita que una dependencia se caiga o se ponga lenta. Qué hace un servicio cuando aquello de lo que depende falla —y cómo se evita que ese fallo se propague— es la materia de la clase 130.

Ejemplo trabajado

CloudShop prepara una campaña que triplicará el tráfico. Antes de dimensionar nada, descubre que sus pruebas de carga llevaban dos años dando resultados que no significaban nada.

El descubrimiento: la prueba decía 8.000 y el sistema aguantaba 1.900.

```
prueba de carga habitual
  modelo                                cerrado, 500 clientes virtuales
  resultado                             8.100 peticiones/s, p99 = 41 ms
  conclusión histórica                  «aguantamos de sobra»

incidente real de la clase 109
  tráfico en el que falló               ~1.900 peticiones/s
```

Al repetir con modelo abierto:

```
peticiones/s sostenidas sin degradar       8.100           1.850
p99 a 1.900 peticiones/s                     44 ms         2.900 ms
p99 a 3.000 peticiones/s                     51 ms          fallo
```

La explicación es la omisión coordinada: al ponerse lento el sistema, los quinientos clientes virtuales enviaban menos, así que la carga bajaba sola y la latencia medida seguía siendo excelente.

Y los otros dos defectos, corregidos a continuación:

```
datos                              12.000 productos    2,1 M productos
distribución                       uniforme            real (el 0,04 % de
                                                       las claves recibe
                                                       el 68 % del tráfico)
aciertos de caché en la prueba          31 %              94 %
capacidad medida                      1.850/s           4.200/s
```

La distribución realista subió la capacidad medida, porque el caché empezó a funcionar como en producción. Con distribución uniforme, la prueba era pesimista en un factor de dos, y con modelo cerrado, optimista en un factor de cuatro. Los dos errores se compensaban parcialmente y el número final no significaba nada.

El codo, medido con esfuerzo creciente.

```
carga        uso de agrupador   p99      errores
1.000/s          22 %           38 ms      0
2.000/s          44 %           41 ms      0
3.000/s          62 %           58 ms      0
3.500/s          71 %           94 ms      0
4.000/s          81 %          310 ms      0
4.200/s          88 %          980 ms    0,1 %
4.400/s          94 %        3.900 ms    4,2 %
4.600/s          99 %          fallo     41 %
```

El codo está entre 3.500 y 4.000, con el agrupador de conexiones al 71-81 %. Doscientas peticiones por segundo separan «bien» de «mal», que es la curva del apartado primero en datos reales.

```
codo                                   ~3.800 peticiones/s
pico habitual                           1.400 peticiones/s  (37 %)
pico previsto en campaña                4.200 peticiones/s  (110 %)
→ hay que aumentar capacidad
```

La prueba de resistencia, que encontró tres cosas.

Ocho horas al 60 % de la capacidad:

```
hora 0    memoria 1,2 GB   conexiones 18   p99 62 ms
hora 2    memoria 1,9 GB   conexiones 22   p99 64 ms
hora 4    memoria 2,6 GB   conexiones 31   p99 71 ms
hora 6    memoria 3,3 GB   conexiones 44   p99 96 ms
hora 8    memoria 4,0 GB   conexiones 58   p99 210 ms
```

Tres pendientes, tres problemas:

```
memoria +350 MB/h    fuga en el cliente de un servicio interno
                     → en producción la tapaba el reinicio automático
                       de la clase 128
conexiones +5/h      conexiones no devueltas al agrupador en un camino
                     de error
                     → en 3 días habrían agotado el techo
p99 creciente        consecuencia de las dos anteriores
```

Ninguna de las tres aparece en una prueba de veinte minutos, y las tres estaban en producción.

La prueba de pico, que reprodujo la tormenta de reintentos.

```
de 1.400 a 4.200 peticiones/s en 10 segundos

t+0 s     latencia sube
t+8 s     empiezan los tiempos de espera hacia el servicio de precios
t+12 s    los reintentos multiplican la carga sobre precios por 3,4
t+25 s    el autoescalado añade instancias
t+95 s    hay capacidad suficiente
t+150 s   el sistema NO se recupera: los reintentos acumulados mantienen
          la saturación
t+240 s   se recupera al vaciarse la cola de reintentos
```

Es el incidente de la clase 113, reproducido a propósito. Y midió lo que ninguna otra prueba mide: el sistema tardó 90 segundos en tener capacidad y 240 en recuperarse.

```
espera creciente con variación            sí             sí
corte tras fallos consecutivos            no             sí (clase 130)
cola de admisión con descarte             no             sí
tiempo de recuperación tras el pico     240 s           40 s
```

El plan de capacidad para la campaña.

```
instancias del servicio                    12         36          minutos
conexiones a la base                       24         60          minutos
TECHO de conexiones de la base            400        400       ← no hay que tocar
cuota de la cuenta para instancias         50        120          2 días
concurrencia de funciones               1.000      2.400          1 día
particiones del registro                   24         24       ← 24 bastan
límite del proveedor de pago            2.000/s    5.000/s       3 SEMANAS
cuota de escrituras del almacén NoSQL  20.000/s   50.000/s          1 día
```

La fila del proveedor de pago es la que justifica la planificación: tres semanas de aviso, y se descubrió con cinco semanas de margen porque se hizo el inventario. Sin él, la campaña habría fallado en el pago con todo lo demás

sobredimensionado.

Y el resultado de la campaña:

```
pico real                                  4.080 peticiones/s
uso frente al codo ampliado                    48 %
p99 durante el pico                            88 ms
errores                                       0,02 %
presupuesto de error consumido en el mes       14 %
incidentes                                      0
```

Lo que se dejó sin optimizar, a propósito.

```
consulta del catálogo, 140 ms         optimizable a ~40 ms
coste estimado del trabajo            2 semanas
alternativa                           duplicar el caché: 90 €/mes
decisión                              pagar los 90 €
motivo                                el coste no crece con el tamaño;
                                      es un caché, no una ineficiencia
                                      que se multiplique
```

A los tres meses.

```
capacidad medida (modelo cerrado)          8.100/s        no se usa
capacidad real (modelo abierto)              1.850/s       4.200/s
codo conocido                              no se sabía     3.800/s
pico habitual respecto al codo             desconocido       37 %
pruebas de resistencia                          0          mensual
fugas encontradas por resistencia               —             3
tiempo de recuperación tras un pico           240 s           40 s
recursos que no escalan solos, inventariados    0              8
tiempo de ampliación conocido                  no              sí
incidentes de capacidad                    3 / 6 meses         0
```

La lección que esta clase traslada a la parte 10: durante dos años las pruebas de carga dijeron que el sistema aguantaba 8.100 peticiones por segundo y la realidad eran 1.850. No fallaba la herramienta ni el esfuerzo: fallaba el modelo de generación, que dejaba de enviar carga justo cuando el sistema empezaba a ir mal. Y de los tres problemas más caros encontrados, ninguno se veía en una prueba corta: los tres eran pendientes que solo aparecen tras varias horas, y los tres estaban ya en producción tapados por reinicios automáticos.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-10-observability-sre-reliability/129-capacidad-rendimiento-y-pruebas-de-carga/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 informe-carga 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 informe-carga 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 prueba de carga dice que se aguanta cuatro veces más de lo real	Modelo cerrado: al ralentizarse el sistema, el generador envía menos y oculta la degradación	Usa modelo abierto a ritmo fijo, o corrige el registro compensando el retraso acumulado.
Los resultados de la prueba no se parecen a producción	Datos escasos y distribución uniforme, que falsean planes de consulta y aciertos de caché	Volumen realista y distribución real, incluida la concentración en las claves calientes.
El sistema pasa de ir bien a caerse sin estados intermedios	La espera crece de forma no proporcional cerca de la saturación	Mide el codo, mantén el pico habitual por debajo del 70 % de él y vigila saturación en vez de uso medio.
Aparecen fugas y agotamientos en producción que ninguna prueba detectó	Solo se hacen pruebas cortas	Prueba de resistencia de varias horas mirando pendientes de memoria, conexiones y colas, no valores puntuales.
El sistema tiene capacidad suficiente y tarda minutos en recuperarse de un pico	Los reintentos acumulados mantienen la saturación después de que pase el pico	Prueba de pico midiendo la recuperación; añade corte tras fallos consecutivos y descarte en la admisión.
Se amplía todo y algo bloquea igualmente	Hay recursos que no escalan solos y tardan semanas en ampliarse	Inventaría cuotas, límites de terceros, particiones y conexiones con su tiempo de ampliación, y planifica con esa antelación.

■ 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 la latencia se dispara cerca de la saturación y qué margen implica?
2. ¿Qué es la omisión coordinada y en qué factor puede falsear la medida?
3. ¿Qué encuentra una prueba de resistencia que no encuentra ninguna otra?
4. ¿Por qué una distribución uniforme falsea los resultados?
5. ¿Qué recursos no escalan solos y por qué cambian la planificación?

Referencias

- Gunther, N. (2007). Guerrilla Capacity Planning — curva de espera, ley de escalabilidad y márgenes. <https://link.springer.com/book/10.1007/978-3-540-31010-5>
- Tene, G. (2015). How NOT to measure latency — omisión coordinada y modelos abierto y cerrado. <https://www.infoq.com/presentations/latency-response-time/>

- Google SRE (2025). Managing load and handling overload — comportamiento cerca de la saturación y descarte. <https://sre.google/sre-book/handling-overload/>
- Gregg, B. (2020). Systems Performance, cap. 2 — metodología de medición y cuellos de botella. <https://www.brendangregg.com/systems-performance-2nd-edition-book.html>
- k6 (2025). Test types: smoke, load, stress, soak, spike — qué pregunta responde cada una. <https://grafana.com/docs/k6/latest/testing-guides/test-types/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 10 en PDF · Manual integral

Anterior	Índice	Siguiente
← 128 · Runbooks, playbooks y automatización operativa	Parte 10 · Programa	130 · Timeouts, retries, backoff, circuit breaker y bulkhead →

Evaluación — 129 Capacidad, rendimiento y pruebas de carga

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 informe-carga 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.

130 — Timeouts, retries, backoff, circuit breaker y bulkhead

Parte: 10 — Observabilidad, SRE y confiabilidad Nivel: avanzado · Horas estimadas: 4 Laboratorio: reliability · Estado: EXECUTABLECORE

Propósito

Decidir qué hace un servicio cuando aquello de lo que depende falla o se pone lento, sabiendo que el comportamiento por defecto es el peor posible: esperar indefinidamente, reintentar de inmediato y compartir los mismos recursos entre todas las dependencias. La clase da los cinco mecanismos que lo corrigen, insiste en el que más incidentes evita y menos se aplica —el plazo, y que ese plazo se herede—, hace la aritmética de la amplificación de reintentos, y ordena los cinco, porque aplicados en el orden equivocado no sirven.

Resultados de aprendizaje

Al finalizar podrás:

1. Fijar plazos a partir de datos y propagarlos entre servicios.
2. Calcular la amplificación de reintentos y limitarla a una capa.
3. Configurar un cortacircuitos que se abra y que además se cierre.
4. Aislar dependencias para que una lenta no consuma todos los recursos.
5. Rechazar pronto cuando hay sobrecarga, en vez de encolar sin límite.

Conceptos centrales

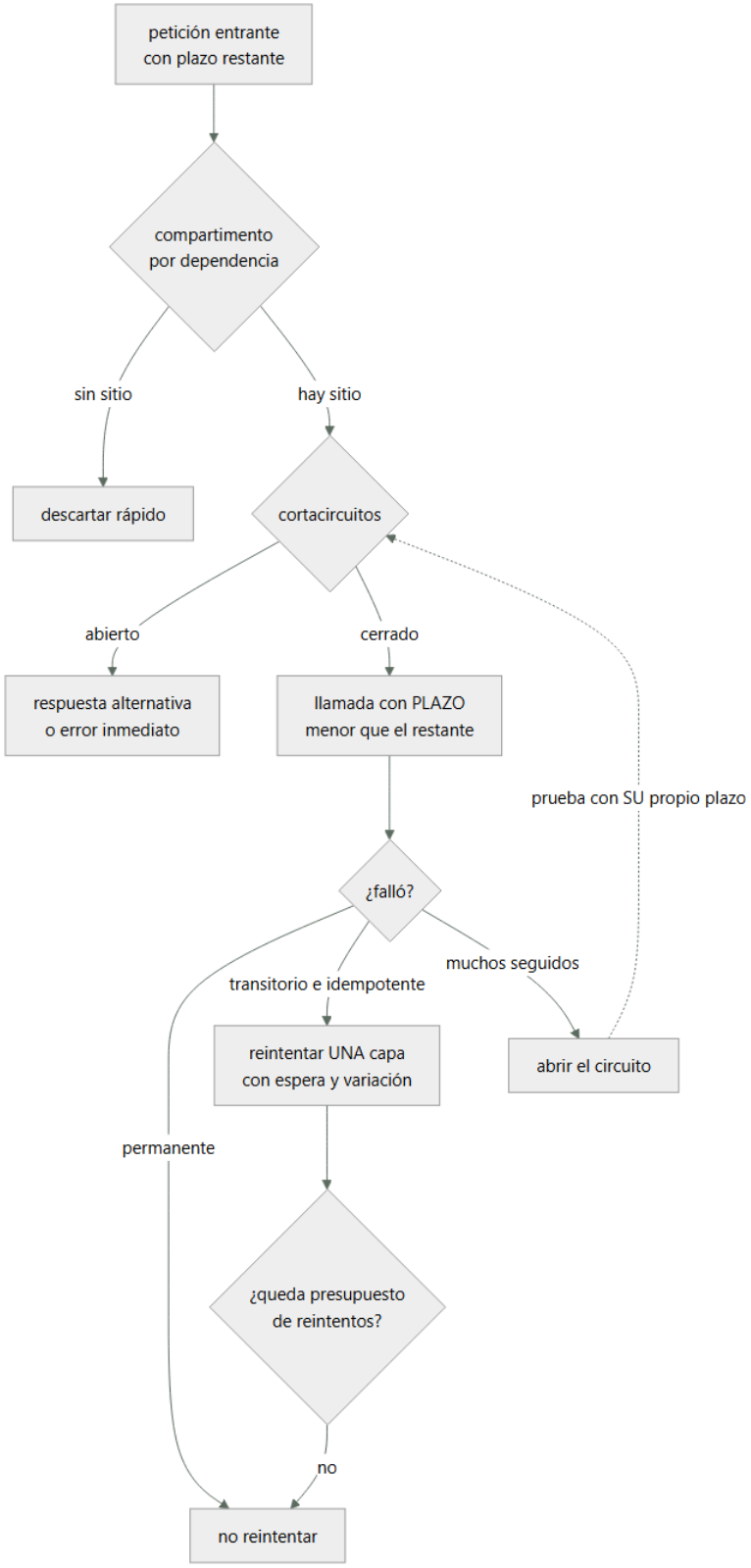
Concepto	Comprensión verificable
plazo	Tiempo máximo que se espera una respuesta. Sin él, los recursos propios quedan retenidos por el problema de otro.
presupuesto de plazo	Lo que queda del plazo de quien llamó. Trabajar más allá de él es trabajo que nadie recibirá.
amplificación	Multiplicación de la carga cuando varias capas reintentan a la vez. Es exponencial en el número de capas.
cortacircuitos	Mecanismo que deja de llamar a una dependencia que falla, falla rápido y prueba de vez en cuando si ya se recuperó.
compartimento estanco	Recursos separados por dependencia, para que la saturación de una no consuma los de todas.
descarte por sobrecarga	Rechazar peticiones deprisa cuando no hay capacidad, en vez de aceptarlas y no poder atenderlas.
cola acotada	Cola con tamaño máximo. Una cola sin límite convierte un problema de capacidad en uno de latencia y luego en uno de memoria.

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

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 plazo, y que se herede

Es el mecanismo más importante de los cinco y el que más veces falta.

```
sin plazo
  la dependencia se queda pensando 90 s
  → el hilo, la conexión y la memoria de la petición siguen ocupados
```

```
  → con suficientes peticiones, el servicio se llena
  → y deja de responder a CUALQUIER COSA, no solo a lo que dependía de ella
```

Y eso es el fallo en cascada: el problema de otro consume tus recursos.

Cómo se elige el valor, que no es un número redondo:

```
se parte del percentil 99 de la dependencia, medido      p99 = 180 ms
se añade margen                                          ×1,5
plazo                                                    ~270 ms
```

Y las dos comprobaciones que evitan los errores típicos:

```
si el plazo es mucho mayor que el p99, no protege de nada
  → 30 s sobre un p99 de 180 ms es como no tener plazo
si es menor que el p99, se cortan peticiones que iban bien
  → y se convierte en un generador de errores propio
```

Y hay más de un plazo que fijar, y se olvidan tres:

```
conexión           establecer la conexión: corto, 1-2 s
lectura            esperar la respuesta: el del cálculo anterior
TOTAL de la llamada  incluidos los reintentos
de la petición completa  el que ve el usuario
```

El tercero es el que más sorprende: con tres reintentos y un plazo de 5 s cada uno, la llamada puede durar veinte segundos aunque cada intento respete su plazo.

El presupuesto de plazo, que es la versión correcta y casi nadie aplica:

```
el usuario espera como mucho 2 s
  la puerta de entrada llama con 1,9 s de presupuesto
    pedidos llama a precios con 1,2 s restantes
      precios llama a la base con 400 ms restantes
```

Y la regla que lo hace útil:

```
si el presupuesto restante es menor que lo que suele tardar la llamada,
NO se llama: se falla ya
→ porque el resultado llegaría cuando nadie lo espera
```

Eso evita el trabajo inútil, que en una sobrecarga es una parte enorme de la carga: el sistema procesando peticiones que el cliente ya abandonó.

Y se implementa propagando el plazo restante en una cabecera y respetándolo en cada salto.

2. Reintentar sin multiplicar

La clase 113 fijó cómo se reintenta —espera creciente, variación, límite— y aquí falta lo que ocurre cuando varias capas reintentan a la vez:

```
cliente móvil          reintenta 3 veces
puerta de entrada      reintenta 3 veces
servicio de pedidos    reintenta 3 veces
cliente de la base     reintenta 3 veces
                    ■■■■■■■■■■■■■■■■■■■■■■
                    3■ = 81 llamadas por cada petición original
```

Ochenta y una llamadas a un sistema que ya está mal. La amplificación es exponencial en el número de capas, no lineal.

Y la regla que lo corta:

```
REINTENTAR EN UNA SOLA CAPA
  la más cercana al fallo que tenga información suficiente
  y las demás propagan el error sin reintentar
```

Y un mecanismo mejor que el límite por llamada, porque acota el total:

```
PRESUPUESTO DE REINTENTOS
  «los reintentos no pueden superar el 10 % de las peticiones normales»
  → con todo bien, hay margen de sobra
  → cuando la dependencia falla al 50 %, los reintentos se cortan solos
  → y el sistema deja de amplificar exactamente cuando importa
```

Es superior al límite por llamada porque el límite por llamada no sabe nada del estado global: tres reintentos por petición son inofensivos con una petición y devastadores con diez mil.

Y el recordatorio de las clases 113 y 116, que aquí es condición previa:

```
solo se reintenta lo TRANSITORIO
y solo si repetir no añade efecto
→ reintentar un cobro no idempotente es cobrar dos veces
```

Y una alternativa que a veces vale más que reintentar: pedir a dos sitios y quedarse con el primero. Cuesta el doble de llamadas y elimina la cola de latencia, y es razonable en lecturas baratas del camino crítico.

3. Cortacircuitos y compartimentos

El cortacircuitos deja de llamar a lo que está roto:

```
CERRADO   pasa todo; se cuentan fallos y lentitud
          si superan el umbral en una ventana → se abre
ABIERTO   no se llama: se falla al instante o se sirve alternativa
          → deja de gastarse tiempo y deja de castigarse a la dependencia
SEMIABIERTO  pasado un tiempo, se dejan pasar unas pocas pruebas
          si van bien → cerrado; si no → abierto otra vez
```

Lo que aporta es doble y conviene tenerlo claro:

```
protege al LLAMANTE   deja de esperar por algo que va a fallar
protege al LLAMADO    deja de recibir carga mientras se recupera
```

Y los tres errores de configuración habituales:

```
umbral por número absoluto de fallos
  → con poco tráfico se abre por tres errores casuales
  → mejor por PROPORCIÓN, con un mínimo de peticiones para evaluar

ventana demasiado corta
  → se abre y se cierra continuamente

la prueba del estado semiabierto sin plazo propio
  → si la dependencia está lenta, la prueba también se queda esperando
  → el circuito NUNCA se cierra, aunque la dependencia ya funcione
```

El tercero es el más traicionero: el circuito abierto se queda abierto para siempre y hace falta un reinicio para arreglarlo.

Y dos decisiones más:

```
un circuito POR DEPENDENCIA, no uno global
estado por instancia o compartido
  por instancia: simple, y cada una aprende por su cuenta
  compartido: reacciona antes, y añade una dependencia más
```

Los compartimentos estancos son el mecanismo que de verdad evita el colapso total:

```
sin compartimentos
  un grupo de 200 hilos para todo
  una dependencia lenta ocupa los 200
  → el servicio deja de atender también lo que no la usa

con compartimentos
  precios: 40    catálogo: 40    pago: 30    resto: 90
  → la dependencia lenta agota SUS 40 y nada más
  → el resto del servicio sigue funcionando
```

Y el efecto en la disponibilidad es exactamente lo que la clase 126 necesitaba: una dependencia caída deja de ser una caída propia.

Y conviene medir por compartimento:

```
ocupación de cada uno, y rechazos por compartimento lleno
→ un compartimento siempre lleno está mal dimensionado
→ o su dependencia está permanentemente degradada
```

4. Rechazar pronto, y en qué orden se aplica todo

Cuando llega más trabajo del que se puede hacer, hay dos opciones y solo una es buena:

```
ENCOLAR SIN LÍMITE
  se aceptan todas y se atienden despacio
  → la latencia crece hasta que nadie espera ya la respuesta
  → y se sigue trabajando para clientes que se fueron
  → y la memoria crece hasta que el proceso muere
```

```
RECHAZAR DEPRISA
  se acepta lo que se puede atender y el resto se rechaza al instante
  → el cliente recibe un error rápido y puede reaccionar
  → y el sistema sigue sirviendo a quienes sí acepta
```

Y la regla que resume esto: toda cola tiene que estar acotada, incluida la que no se ve —la de conexiones aceptadas y no atendidas—.

Y el descarte se hace con criterio, no al azar:

```
por prioridad     las comprobaciones de salud y el camino crítico primero
                  lo secundario se descarta antes
por antigüedad    descartar lo más viejo, no lo más nuevo:
                  lo viejo probablemente ya no lo espera nadie
por cliente       proteger a los demás de uno que abusa   clase 118
```

La segunda es contraintuitiva y correcta.

Las respuestas alternativas, que es lo que se sirve con el circuito abierto:

```
valor cacheado, aunque esté caducado          clase 111
valor por defecto: precio base, sin descuento
respuesta parcial: la página sin esa sección  clase 124
colar el trabajo para hacerlo después         clase 113
error claro, si ninguna de las anteriores sirve
```

Y la advertencia imprescindible: una alternativa que no se prueba no funciona. Se ensaya, y de eso trata la clase 131.

El orden en que se aplican los cinco, que importa:

```
1. COMPARTIMENTO   antes de nada: ¿hay recursos para esta dependencia?
2. CORTACIRCUITOS  ¿tiene sentido intentarlo siquiera?
3. PLAZO           con el presupuesto restante
4. REINTENTO       solo si es transitorio, idempotente y hay presupuesto
5. ALTERNATIVA     si nada de lo anterior dio resultado
```

Y el error frecuente es ponerlos al revés: reintentar dentro del compartimento consume sus recursos tres veces, y un cortacircuitos por detrás del reintento cuenta los tres intentos como tres fallos y se abre demasiado pronto.

Y la lista de comprobación de la clase:

- toda llamada remota tiene plazo, y sale de datos medidos
- hay plazo de conexión, de lectura y total con reintentos
- el plazo restante se propaga y se respeta en cada salto
- no se llama si el presupuesto restante no da para la llamada
- solo reintenta una capa, y está escrito cuál
- hay presupuesto global de reintentos, no solo límite por llamada
- solo se reintenta lo transitorio e idempotente
- hay un cortacircuitos por dependencia, con umbral por proporción
- la prueba del estado semiabierto tiene su propio plazo
- hay compartimentos por dependencia, y se mide su ocupación
- todas las colas están acotadas, incluidas las de aceptación
- el descarte prioriza y descarta lo más antiguo
- las respuestas alternativas se ensayan

Y el cierre que enlaza con la clase siguiente: los cinco mecanismos están configurados y nadie ha comprobado que funcionen. Provocar el fallo a propósito, en un entorno controlado y luego en producción, es la materia de la clase 131.

Ejemplo trabajado

CloudShop sufre una caída total cuando el servicio de recomendaciones —que es opcional— se pone lento. El ejercicio consiste en reconstruir por qué algo opcional tumbó el sistema, y aplicar los cinco mecanismos midiendo cada uno.

El incidente que lo motiva.

```
10:14  el servicio de recomendaciones empieza a tardar 30 s por llamada
       (no falla: TARDA)
10:15  los hilos del servicio de portada se van ocupando
10:17  los 200 hilos están esperando a recomendaciones
10:17  la portada deja de responder a CUALQUIER COSA, incluso a lo que no la usa
10:18  la comprobación de salud tampoco responde: se reinician instancias
10:19  las instancias nuevas se llenan igual en 40 s
10:23  las comprobaciones fallidas sacan instancias del reparto
10:31  caída total del sitio
10:52  se identifica y se apaga recomendaciones con un interruptor
```

```
10:54  recuperado

duración                                              40 min
causa                        una dependencia OPCIONAL, que no fallaba
```

Y el diagnóstico con el vocabulario de esta clase:

```
sin plazo                    la llamada esperaba 30 s
sin compartimento            los 200 hilos eran compartidos
sin cortacircuitos           se seguía llamando a algo que no respondía
sin alternativa              no había forma de servir sin recomendaciones
la salud compartía hilos     así que el reinicio empeoró todo
```

Corrección 1: plazos, medidos.

```
dependencia        p99 medido    plazo fijado
recomendaciones       210 ms        300 ms
catálogo               96 ms        150 ms
precios                41 ms         80 ms
inventario             88 ms        150 ms
pago                  680 ms      1.200 ms
base de datos          14 ms         50 ms
```

Y tres plazos que faltaban por completo:

```
plazo de conexión                    no había  →  1 s
plazo total con reintentos           no había  →  2× el de lectura
plazo de la petición completa        no había  →  2 s
```

Y el efecto inmediato al repetir el incidente en un ensayo:

```
hilos ocupados por recomendaciones          200            varía
tiempo hasta que la portada se llena         3 min         no se llena
latencia de la portada durante el fallo    sin respuesta   +300 ms
```

Corrección 2: la amplificación, medida.

Al instrumentar el conteo de llamadas por petición original durante un fallo:

```
peticiones originales de usuario                          1.000
llamadas recibidas por el servicio de precios             7.400
```

Siete coma cuatro llamadas por petición. Y las capas que reintentaban:

```
cliente móvil                          2 reintentos
puerta de entrada                      2
servicio de portada                    2
cliente HTTP de la biblioteca          1   ← nadie sabía que reintentaba
```

La última es la que suele sorprender: el cliente HTTP reintentaba por su cuenta, con la configuración por defecto de la librería.

```
capas que reintentan                       4               1 (la portada)
llamadas por petición durante un fallo     7,4             1,3
presupuesto de reintentos                  no había        10 %
qué pasa con el 50 % de fallos       reintentos ×3     reintentos cortados
                                                       al llegar al 10 %
```

Corrección 3: el cortacircuitos que no se cerraba.

La primera versión se configuró así:

```
umbral: 5 fallos consecutivos
ventana: 10 s
prueba semiabierta: 1 llamada cada 30 s, SIN plazo propio
```

Y en el primer uso real:

```
14:20  recomendaciones se degrada; el circuito se abre. Correcto.
14:26  recomendaciones se recupera
14:26  la prueba semiabierta se lanza… y usa el plazo por defecto,
       que era el heredado de la llamada normal
14:26  la prueba se queda esperando y se cuenta como fallo
17:40  el circuito sigue abierto 3 h 14 después de la recuperación
```

Tres horas y catorce minutos sin recomendaciones por un fallo de configuración de la prueba.

```
umbral                             5 fallos absolutos   50 % en 30 s,
                                                        mínimo 20 peticiones
```

```
prueba semiabierta                  1 sin plazo          3 con plazo de 500 ms
aperturas por ruido con poco tráfico   11 / mes             0
tiempo medio hasta cerrar tras
recuperarse                          no cerraba           38 s
```

Corrección 4: los compartimentos.

```
hilos                                 200 compartidos    por dependencia:
                                                         precios 40
                                                         catálogo 40
                                                         inventario 30
                                                         recomendaciones 20
                                                         pago 30
                                                         resto 40
salud y camino crítico              mismos hilos       hilos propios: 10
```

Y el ensayo del mismo fallo, con todo aplicado:

```
duración                                  40 min             0
peticiones fallidas                       100 %              0 %
latencia de la portada                 sin respuesta       +40 ms
funcionalidad perdida                     todo          solo recomendaciones
comprobaciones de salud                   fallaban        correctas
instancias reiniciadas por error             34              0
```

Corrección 5: descartar y responder algo.

```
cola de aceptación                      sin límite      500, con descarte
qué se descarta                            nada        lo más antiguo primero
respuesta con el circuito abierto       error 500      portada sin la sección
valor caducado servible                    no              sí, hasta 1 h
```

Y el ensayo de sobrecarga, comparado con la clase 129:

```
carga aplicada                           6.000/s           6.000/s
peticiones atendidas correctamente       410/s            3.700/s
p99 de las atendidas                     41 s             210 ms
peticiones rechazadas al instante          0              2.300/s
memoria del proceso                      creciente        estable
```

Atendiendo menos peticiones se atienden nueve veces más, porque dejan de gastarse recursos en peticiones que ya nadie espera.

El orden, comprobado.

La primera implantación puso el reintento por dentro del compartimento:

```
reintento dentro del compartimento
  → cada petición ocupaba su hueco 3 veces
  → el compartimento de 40 se comportaba como uno de 13
  → y el cortacircuitos contaba 3 fallos por cada fallo real
     y se abría tres veces más rápido de lo previsto
```

```
capacidad efectiva del compartimento         13 de 40           40 de 40
aperturas del circuito por mes                 14                  3
```

A los cuatro meses.

```
llamadas remotas con plazo                    31 %           100 %
plazo propagado entre servicios                 no             sí
capas que reintentan                             4              1
llamadas por petición durante un fallo         7,4            1,3
cortacircuitos por dependencia                   0             11
compartimentos                                   0              6
colas acotadas                                   0         todas (7)
caídas totales por una dependencia lenta    2 / 6 meses          0
disponibilidad del flujo de compra           99,21 %         99,74 %
peticiones atendidas en sobrecarga             410/s          3.700/s
```

La lección que esta clase traslada a la parte 10: la caída de cuarenta minutos la provocó una dependencia opcional que ni siquiera fallaba: solo tardaba. Y de los cinco mecanismos, los dos que la habrían evitado por sí solos son los más aburridos —poner un plazo y separar los hilos—. Los otros tres mejoran el sistema, y el más vistoso, el cortacircuitos, estuvo tres horas dejando el servicio degradado por un fallo propio de configuración: un mecanismo de resiliencia mal ajustado es una fuente de incidentes más.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-10-observability-sre-reliability/130-timeouts-retries-backoff-circuit-breaker-y-bulkhead/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 cliente-resiliente 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 cliente-resiliente para el caso CloudShop. Incluye una alternativa descartada, un supuesto que pueda falsarse, una prueba de fallo y una decisión de rollback.

■ Criterio de aceptación

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

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
Una dependencia lenta deja sin respuesta al servicio entero	No hay plazo y los recursos quedan retenidos esperando	Plazo en toda llamada remota, calculado desde el percentil 99 medido, y compartimentos separados por dependencia.
Durante un fallo, la dependencia recibe muchas más llamadas de las que se le hacen	Varias capas reintentan a la vez y la amplificación es exponencial	Reintenta en una sola capa, escríbelo, revisa lo que hacen las librerías por su cuenta y añade presupuesto global de reintentos.
El circuito se abre y no vuelve a cerrarse aunque la dependencia funcione	La prueba del estado semiabierto no tiene plazo propio y se cuenta como fallo	Plazo corto y propio para las pruebas, varias pruebas en vez de una, y umbral por proporción con mínimo de peticiones.
El sistema acepta todo y no atiende casi nada	Colas sin límite: se sigue trabajando para clientes que ya se fueron	Acota todas las colas, rechaza deprisa cuando no hay capacidad y descarta primero lo más antiguo.
El compartimento se comporta como si tuviera un tercio de su tamaño	El reintento está por dentro del compartimento y ocupa su hueco varias veces	Aplica el orden: compartimento, cortacircuitos, plazo, reintento, alternativa.
La respuesta alternativa falla justo cuando se necesita	Nunca se ha ejecutado	Ensaya las alternativas periódicamente provocando el fallo de la dependencia.

■ Seguridad, ética y costo

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

■ Preguntas de comprobación

1. ¿Por qué una llamada sin plazo puede tumbar un servicio entero?
2. ¿Qué es el presupuesto de plazo y qué evita?
3. ¿Cuánto amplifica la carga que cuatro capas reintenten tres veces?
4. ¿Por qué un presupuesto global de reintentos es mejor que un límite por llamada?
5. ¿En qué orden se aplican los cinco mecanismos y qué pasa si se invierten dos?

Referencias

• Nygard, M. (2018). Release It!, caps. 4 y 5 — plazos, cortacircuitos, compartimentos y fallos en cascada. <https://pragprog.com/titles/mnee2/release-it-second-edition/>

• AWS (2025). Timeouts, retries and backoff with jitter — cálculo de plazos y amplificación. <https://aws.amazon.com/builders-library/timeouts-retries-and-backoff-with-jitter/>

• Google SRE (2025). Addressing cascading failures — descarte por sobrecarga, colas acotadas y presupuesto de reintentos. <https://sre.google/sre-book/addressing-cascading-failures/>

• gRPC (2025). Deadlines and deadline propagation — presupuesto de plazo heredado entre servicios. <https://grpc.io/docs/guides/deadlines/>

• Dean, J. y Barroso, L. (2013). The tail at scale — peticiones duplicadas y respuesta parcial como alternativas al reintento. <https://research.google/pubs/pub40801/>

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

Descargar: Parte 10 en PDF · Manual integral

Anterior	Índice	Siguiente
← 129 · Capacidad, rendimiento y pruebas de carga	Parte 10 · Programa	131 · Chaos engineering y game days →

Evaluación — 130 Timeouts, retries, backoff, circuit breaker y bulkhead

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 cliente-resiliente 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.

131 — Chaos engineering y game days

Parte: 10 — Observabilidad, SRE y confiabilidad Nivel: avanzado · Horas estimadas: 4 Laboratorio: chaos · Estado: EXECUTABLECORE

Propósito

Comprobar que todo lo de las clases 121 a 130 funciona de verdad, y la única forma de comprobarlo es provocarlo. La clase distingue un experimento —con estado normal definido, hipótesis escrita, radio acotado y condición de parada— de romper cosas al azar; establece los requisitos previos sin los cuales esto es solo causar incidentes; y añade la parte que la automatización no cubre: ensayar con personas, porque los procedimientos, los permisos y los relevos solo fallan cuando hay alguien ejecutándolos.

Resultados de aprendizaje

Al finalizar podrás:

1. Formular un experimento con estado normal, hipótesis, radio y parada.
2. Comprobar los requisitos previos antes de provocar nada.
3. Elegir qué fallo inyectar según lo que se quiere verificar.
4. Organizar un ensayo con personas que ponga a prueba el proceso, no solo el sistema.
5. Convertir los experimentos en comprobaciones continuas que no se pudran.

Conceptos centrales

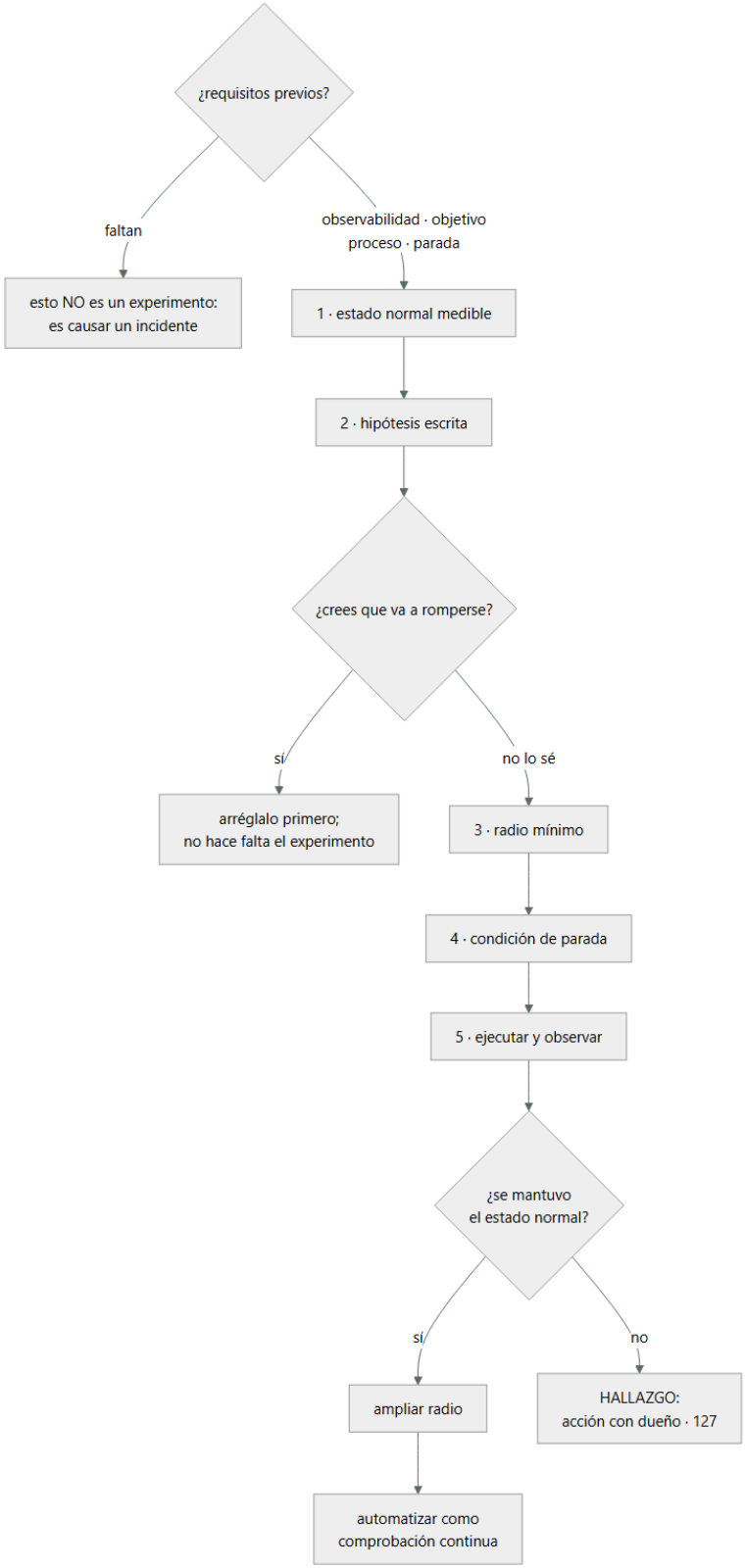
Concepto	Comprensión verificable
estado normal	Descripción medible de que el sistema funciona. Se toma del objetivo de la clase 126, no se inventa.
hipótesis	Afirmación previa y falsable: «si ocurre X, el estado normal se mantiene». Sin ella no hay experimento.
radio de afectación	Cuánto del sistema y cuántos usuarios pueden verse afectados. Se empieza por el mínimo y se amplía.
condición de parada	Umbral escrito de antemano que detiene el experimento automáticamente. Es lo que lo hace responsable.
inyección de latencia	Hacer que una dependencia tarde en lugar de fallar. Es el fallo más dañino y el menos ensayado.
ensayo con personas	Ejercicio anunciado en el que un equipo responde a un fallo real siguiendo el proceso. Prueba lo que ninguna automatización prueba.

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

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

■ Flujo de razonamiento



Desarrollo

1. Un experimento, no una travesura

La diferencia entre esto y romper cosas al azar es el método, y son cinco pasos:

1. `ESTADO NORMAL`
   `qué significa que el sistema funciona, con números`
   `→ sale del objetivo de la clase 126, no se inventa uno nuevo`

```
   → «proporción de pedidos completados ≥ 99,5 %, p99 < 500 ms»

2. HIPÓTESIS
   «si el servicio de precios deja de responder, el estado normal
    se mantiene sirviendo el precio base»
   → escrita ANTES, y falsable

3. RADIO
   el mínimo que permita observar algo: una instancia, un 1 % del tráfico,
   un entorno, una zona

4. CONDICIÓN DE PARADA
   «si los errores superan el 2 % o el presupuesto consumido supera
    el 5 %, se detiene automáticamente»
   → escrita antes, y automática, no «alguien lo estará mirando»

5. EJECUTAR, OBSERVAR Y CONCLUIR
   y publicar el resultado, se confirme o no la hipótesis
```

Y la regla que más malentendidos evita:

```
si estás bastante seguro de que se va a romper, NO hagas el experimento
→ arréglalo, que ya sabes lo que va a pasar
→ el experimento sirve para comprobar creencias, no para descubrir
  lo evidente
```

Y su contraria: un experimento que nunca falla tampoco aporta. Si diez experimentos seguidos confirman la hipótesis, hay que subir la dificultad.

Los requisitos previos, sin los cuales esto no es un experimento sino un incidente provocado:

```
observabilidad suficiente para ver el efecto        clases 121-124
un objetivo que defina el estado normal             clase 126
proceso de incidentes por si se va de las manos     clase 127
un botón de parada que funcione, probado antes
y aviso previo a quien esté de guardia
```

El cuarto merece énfasis: lo primero que se prueba es la parada, antes que ninguna otra cosa.

Y la postura organizativa, que decide si esto sobrevive:

```
si el experimento rompe algo, eso es un HALLAZGO, no un fallo de nadie
→ y se trata como una revisión de incidente: acción, dueño y fecha
→ si se castiga, no se vuelve a hacer ningún experimento
```

2. Qué inyectar

Ordenado por lo que este programa ha demostrado que hace daño:

```
LATENCIA en una dependencia
  la más valiosa y la menos ensayada
  → la caída de la clase 130 la provocó algo que NO fallaba: tardaba
  → verifica plazos, compartimentos y cortacircuitos a la vez

ERROR en una dependencia
  devolver 500 o rechazar conexiones
  → verifica alternativas y presupuesto de reintentos

AGOTAMIENTO DE RECURSOS
  llenar el agrupador de conexiones, los hilos, el disco, la memoria
  → verifica saturación, descarte y colas acotadas

MUERTE DE INSTANCIAS
  la más conocida y la menos informativa en sistemas modernos:
  casi siempre funciona

PÉRDIDA DE UNA ZONA
  verifica reparto, cuotas y capacidad restante

PARTICIÓN DE RED
  el caso duro: los dos lados vivos y sin verse
  → verifica plazos y comportamiento ante ambigüedad

DESFASE DE RELOJ
  rompe expiraciones, firmas y ordenaciones por tiempo
```

```
CADUCIDAD de certificado o credencial
  → ensaya lo que la clase 125 alertaba como indicador adelantado
```

Y los específicos de la parte 09, que casi nadie ensaya y que causaron la mayoría de sus incidentes:

```
entregar el mismo mensaje dos veces              clase 116
entregar mensajes desordenados                   clase 114
conmutar la base de datos                        clase 109
vaciar el caché en hora punta                    clase 111
parar un consumidor o un publicador              clases 113, 116
reanudar instancias del motor con código nuevo   clase 119
```

Y dos que se ensayan poco y valen mucho:

```
LA DEPENDENCIA QUE RESPONDE MAL
  no falla ni tarda: devuelve datos incorrectos o vacíos
  → verifica validación y si el sistema propaga basura

LA PROPIA TELEMETRÍA CAÍDA
  → verifica que la aplicación no se cae con ella (clases 121, 122)
  → y que alguien se entera de que no hay datos (clase 123)
```

Y dónde se ejecuta, con una postura honesta:

```
ENTORNO APARTE   primero siempre; verifica caminos de código
                 y no dice nada sobre la escala real
PRODUCCIÓN       la única que dice la verdad
                 con radio del 1 %, en horas valle, con parada automática
                 atada al presupuesto de error
```

Y el argumento para llegar a producción, que conviene tener preparado: el fallo va a ocurrir en producción de todos modos; la elección es si ocurre un martes a las once con el equipo mirando, o un domingo a las cuatro de la madrugada.

3. Ensayar con personas

La automatización comprueba el sistema. Un ensayo con personas comprueba lo que la automatización nunca toca:

```
¿el procedimiento sirve a quien no lo escribió?          clase 128
¿quien está de guardia tiene los permisos?
¿el proceso de incidentes se sigue de verdad?            clase 127
¿alguien sabe a quién escalar, y esa persona responde?
¿las herramientas están accesibles desde fuera de la oficina?
¿el canal, el documento y los avisos funcionan?
```

Y el formato que funciona:

```
ANUNCIADO      fecha conocida; no se trata de pillar a nadie
ACOTADO        un escenario, 60-90 minutos
CON OBSERVADOR quien conoce la respuesta correcta apunta y NO ayuda
CON PARADA     el observador corta si se va de las manos
CON REVISIÓN   media hora después, con la misma disciplina de la clase 127
```

Y dos variantes que enseñan mucho:

```
SIN QUIEN LO CONSTRUYÓ
  quien construyó el servicio no participa: solo observa
  → revela cuánto conocimiento vive en una sola cabeza

CON ESCENARIO OCULTO
  el equipo sabe que hay ejercicio, no cuál es el fallo
  → prueba el diagnóstico, no la memoria
```

Y los hallazgos típicos, que rara vez son técnicos:

```
el procedimiento existe y quien está de guardia no lo encuentra
faltan permisos, y pedirlos tarda más que el incidente
el escalado apunta a alguien que cambió de equipo
nadie sabe quién comunica a los clientes
el panel que hace falta no existe
y la herramienta de emergencia requiere una credencial que caducó
```

Y una regla de higiene: el ensayo termina con acciones, dueño y fecha, exactamente igual que un incidente real. Si no, se convierte en una actividad simpática que no cambia nada.

Y la frecuencia razonable:

```
ensayo con personas          trimestral, por equipo
experimento automático       continuo
escenario nuevo              al menos uno por trimestre
repetir un escenario viejo   una vez al año, para comprobar que sigue bien
```

La última es la que detecta las regresiones: lo que se arregló hace un año puede haberse desarreglado.

4. Que no se pudra

Un experimento ejecutado una vez comprueba el sistema de ese día. La ley 13 se aplica también aquí: un experimento que dejó de ejecutarse no da ningún error.

Lo que lo convierte en algo permanente:

```
el experimento se escribe como código y vive en el repositorio
se ejecuta de forma programada, no cuando alguien se acuerda
su resultado es una comprobación que pasa o falla
y si falla, abre una tarea con dueño
```

Y conviene distinguir dos ritmos:

```
EN LA CANALIZACIÓN   experimentos baratos y acotados, en preproducción
                     → matar una instancia, inyectar latencia en una
                       dependencia, devolver errores
                     → tardan minutos y detectan regresiones

PROGRAMADOS EN PRODUCCIÓN  los caros y los de radio mayor
                     → pérdida de zona, conmutación de base,
                       vaciado de caché
                     → mensual o trimestral, en ventana anunciada
```

Y las cifras con las que se mide el programa:

```
escenarios en el catálogo
proporción ejecutada en el último trimestre
hallazgos por trimestre, y cuántos siguen abiertos
escenarios que fallan hoy y antes pasaban   ← regresiones
tiempo medio de detección durante los ensayos frente al de incidentes reales
```

La última es especialmente útil: si en los ensayos se detecta en dos minutos y en los incidentes reales en veinte, el ensayo está siendo demasiado fácil —normalmente porque todo el mundo sabe que hay ejercicio y está mirando—.

Y una advertencia sobre el alcance: esto no sustituye al diseño. Un sistema sin plazos ni compartimentos no necesita experimentos para saber que va a caerse; necesita la clase 130. Los experimentos son para comprobar lo que se cree que está resuelto.

Y la lista de comprobación de la clase:

- ■ hay observabilidad, objetivo y proceso de incidentes antes de empezar
- ■ la parada automática está probada antes que ningún experimento
- ■ cada experimento tiene estado normal, hipótesis, radio y parada escritos
- ■ no se ejecutan experimentos cuyo resultado ya se conoce
- ■ se inyecta latencia, no solo fallo
- ■ el catálogo incluye los fallos propios de la parte 09
- ■ se ensaya también la caída de la propia telemetría
- ■ se empieza en un entorno aparte y se progresa a producción con radio mínimo
- ■ hay ensayo con personas trimestral, con observador que no ayuda
- ■ los hallazgos generan acciones con dueño y fecha
- ■ los experimentos viven como código y se ejecutan programados
- ■ se repite un escenario antiguo cada año para detectar regresiones
- ■ romper algo en un experimento no se penaliza

Y el cierre que enlaza con la clase siguiente: con esto está completo el material de la parte 10. La clase 132 monta la operación entera y califica las cinco predicciones de la clase 120, empezando por la cifra que abrió la parte: tres de veintiún problemas detectados por una alerta.

Ejemplo trabajado

CloudShop tiene configurados los cinco mecanismos de la clase 130 y ninguno verificado. El programa de experimentos empieza por lo más barato y termina descubriendo que dos de las tres cosas que fallan no son técnicas.

Experimento 0: la parada.

Antes de nada se probó el botón de parada, y no funcionaba:

```
el mecanismo de parada requería una credencial administrativa
que la persona designada no tenía
tiempo hasta poder parar un experimento, medido            11 min
corregido antes de ejecutar ningún experimento
```

Experimento 1: latencia en una dependencia opcional.

```
estado normal   pedidos completados ≥ 99,5 % · p99 < 500 ms
hipótesis       «si recomendaciones tarda 5 s, el estado normal se mantiene
                 sirviendo la portada sin esa sección»
radio           1 % del tráfico, martes 11:00
parada          errores > 2 % o presupuesto consumido > 3 %

resultado    HIPÓTESIS CONFIRMADA
  p99 de la portada durante el experimento          +38 ms
  peticiones servidas sin recomendaciones            0,9 %
  errores                                            0 %
  circuito abierto a los                             14 s
  circuito cerrado tras retirar la latencia          41 s
```

Y se amplió el radio al 25 % y luego al 100 %, con el mismo resultado. Lo que en la clase 130 fue una caída de cuarenta minutos ahora es un experimento de rutina.

Experimento 2: la dependencia que responde mal.

```
hipótesis    «si el servicio de precios devuelve importes vacíos,
              el sistema los rechaza y sirve el precio base»
radio        entorno aparte, primero

resultado    HIPÓTESIS REFUTADA
  el importe vacío se interpretaba como 0
  pedidos creados con importe 0 en el experimento           41
  el sistema NO validaba: propagaba
  llegó hasta la factura, en el entorno de pruebas
```

Cuarenta y un pedidos de cero euros. Y este experimento no se ejecutó en producción precisamente porque el resultado en el entorno aparte fue concluyente. Se añadió validación y se repitió:

```
segunda ejecución    hipótesis confirmada; los importes vacíos se rechazan
                     y se sirve precio base
```

Experimento 3: parar el publicador de la tabla de salida.

```
hipótesis   «si el publicador se para 20 min, la alerta de antigüedad avisa
             en menos de 5 y no se pierde ningún evento»

resultado   PARCIALMENTE REFUTADA
  alerta de antigüedad                      avisó a los 4 min ✓
  eventos perdidos                          0 ✓
  al reanudar, 38.000 filas de golpe        los consumidores absorbieron ✓
  PERO: la alerta fue al buzón de un equipo que ya no existía
  tiempo real hasta que alguien la vio                      26 min
```

La alerta funcionaba y su destinatario no. Se revisaron los dueños de las setenta y tres alertas: once apuntaban a equipos o personas que habían cambiado.

Experimento 4: pérdida de una zona.

```
hipótesis   «si se pierde una zona de tres, el servicio se mantiene
             con degradación menor del 10 % en latencia»
radio       producción, domingo 07:00, con parada automática

resultado   REFUTADA por un motivo que nadie había previsto
  el reparto funcionó ✓
  las réplicas de la base conmutaron en 68 s ✓
  PERO la cuota de instancias de la cuenta impedía crear
  las instancias de reemplazo en las otras dos zonas
  capacidad disponible tras la pérdida              64 % de la necesaria
  latencia p99                                      1.900 ms
  parada automática activada a los                  3 min 20
```

Es exactamente el inventario de «lo que no escala solo» de la clase 129, y estaba mal:

```
cuota de instancias                          50          140
reserva para pérdida de zona                 no        sí, calculada
repetición del experimento                    —     hipótesis confirmada
```

El primer ensayo con personas.

```
escenario oculto   inyección de latencia en la base de datos
equipo             cuatro personas de guardia rotatoria
observador         quien conocía el escenario, sin ayudar
duración prevista  90 min
```

```
línea de tiempo del ensayo
11:00  se inyecta latencia
11:02  alerta de ritmo de consumo del presupuesto ✓
11:03  declarado; papeles asignados ✓
11:04  se consulta la línea de cambios: no hay cambios recientes ✓
11:09  se identifica la base como origen, por saturación del agrupador ✓
11:11  se busca el procedimiento: existe y está enlazado ✓
11:13  el procedimiento pide ejecutar una consulta de diagnóstico
       → quien está de guardia NO tiene permiso de lectura sobre
         las vistas del sistema
11:22  se localiza a alguien con permiso, que no estaba de guardia
11:31  diagnóstico completado
11:34  mitigado
```

```
hallazgos del ensayo
  técnicos                                        1
  de permisos                                     3
  de procedimiento                                4
  de proceso (quién comunica, a quién se escala)  2
```

Nueve de diez hallazgos no eran técnicos. Y el de permisos costó nueve de los treinta y cuatro minutos.

Y la variante sin quien lo construyó, dos meses después:

```
escenario   consumidor de eventos detenido
resultado   el equipo tardó 41 min en algo que su autor resuelve en 6
causa       tres pasos del procedimiento suponían conocimiento no escrito
acción      reescritos y verificados con la prueba de la clase 128
repetición  a los 3 meses: 9 min
```

El catálogo continuo.

```
escenarios en el catálogo                      4            23
en la canalización, por despliegue             0             6
programados en producción                      0             9
ensayos con personas                           0       2 por trimestre
hallazgos totales                              —            41
  técnicos                                     —            17
  de permisos, procedimiento o proceso         —            24
hallazgos cerrados en plazo                    —       37 de 41
regresiones detectadas (pasaba antes, falla hoy)  —            3
```

Las tres regresiones son el argumento del apartado cuarto: un plazo eliminado en una refactorización, un compartimento mal dimensionado tras un cambio de configuración y una alternativa que dejó de funcionar al cambiar un contrato. Ninguna la habría detectado nada más.

A los seis meses.

```
mecanismos de la clase 130 verificados        0 de 5        5 de 5
escenarios ejecutados                            0            23
hallazgos                                        —            41
de ellos, no técnicos                            —            24
alertas con destinatario incorrecto             11             0
cuotas mal dimensionadas                         3             0
regresiones de resiliencia detectadas            —             3
incidentes reales en el semestre                 6             2
tiempo medio de mitigación en incidentes reales  14 min        7 min
```

La lección que esta clase traslada a la parte 10: de cuarenta y un hallazgos, veinticuatro no eran del sistema: eran permisos que la persona de guardia no tenía, procedimientos que suponían conocimiento no escrito, alertas que iban a equipos disueltos y una cuota mal dimensionada. Ninguno de los cinco mecanismos de la clase 130 falló en el primer experimento; lo que falló fue todo lo que los rodea. Y el hallazgo que mejor resume la clase ocurrió antes del primer experimento: el botón de parada no funcionaba.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-10-observability-sre-reliability/131-chaos-engineering-y-game-days/lab.py
```

El laboratorio selecciona el motor de práctica chaos 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 experimento-caos 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 hipótesis de resiliencia y criterio de abortar. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye experimento-caos 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 experimento se convierte en un incidente real	No había condición de parada automática, observabilidad suficiente o proceso de incidentes	Comprueba los requisitos previos, prueba la parada antes que nada y ata la condición de corte al presupuesto de error.
Los experimentos no descubren nada nuevo	Se ejecutan solo escenarios cuyo resultado ya se conoce	Si estás seguro de que se romperá, arréglalo; si diez seguidos confirman la hipótesis, sube la dificultad.
Se ensaya la caída de dependencias y los incidentes reales los causa la lentitud	Solo se inyecta fallo, no latencia	Inyecta latencia como escenario principal: es lo que verifica plazos, compartimentos y cortacircuitos a la vez.
El sistema responde bien al experimento y el equipo no	Solo se ensaya de forma automática, sin personas	Ensayo trimestral con observador que no ayuda, escenario oculto y variante sin quien construyó el servicio.
Lo que se arregló hace un año vuelve a fallar	Los experimentos se ejecutaron una vez y no se repiten	Escríbelos como código, ejecútalos programados y repite un escenario antiguo cada año.
Nadie quiere volver a hacer experimentos	Cuando algo se rompió se trató como un fallo de alguien	Un hallazgo es el resultado esperado; se gestiona como una revisión de incidente, con acción, dueño y fecha.

■ 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 cinco pasos de un experimento y qué lo distingue de romper cosas?
2. ¿Qué requisitos previos hay que tener y cuál se prueba primero?
3. ¿Por qué la inyección de latencia es más valiosa que la de fallo?
4. ¿Qué comprueba un ensayo con personas que ninguna automatización comprueba?
5. ¿Por qué hay que repetir escenarios antiguos?

Referencias

- Basiri, A. y otros (2016). Chaos engineering — método, hipótesis y radio de afectación. <https://ieeexplore.ieee.org/document/7503833>
- Principles of Chaos (2025). Principles of chaos engineering — estado normal, experimentos en producción y automatización. <https://principlesofchaos.org/>
- Rosenthal, C. y Jones, N. (2020). Chaos Engineering, caps. 3-5 — madurez, adopción y ensayos con personas. <https://www.oreilly.com/library/view/chaos-engineering/9781492043850/>
- Google SRE (2025). Disaster role playing and DiRT — ensayos con personas y hallazgos organizativos. <https://sre.google/sre-book/accelerating-sre-on-call/>
- AWS (2025). Fault Injection Service: experiment templates and stop conditions — radio y condiciones de parada. <https://docs.aws.amazon.com/fis/latest/userguide/what-is.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 10 en PDF · Manual integral

Anterior	Índice	Siguiente
← 130 · Timeouts, retries, backoff, circuit breaker y bulkhead	Parte 10 · Programa	132 · Proyecto: operación SRE de CloudShop →

Evaluación — 131 Chaos engineering y game days

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 experimento-caos 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.

132 — Proyecto: operación SRE de CloudShop

Parte: 10 — Observabilidad, SRE y confiabilidad Nivel: avanzado · Horas estimadas: 8 Laboratorio: capstone · Estado: EXECUTABLECORE

Propósito

Montar la operación completa —señales, objetivos, alertas, incidentes, procedimientos, capacidad, resiliencia y experimentos— como un solo bucle, y cerrar la parte con las tres piezas de siempre: calificar las cinco predicciones de la clase 120, dos de las cuales fallaron y una se quedó muy corta; incorporar la ley que ha aparecido cuatro veces en estas doce clases y que es hermana de la ley 13; y escribir la predicción que la parte 11 tendrá que corregir.

Resultados de aprendizaje

Al finalizar podrás:

1. Describir la operación como un bucle cerrado y no como una lista de herramientas.
2. Montar el conjunto sobre un servicio real y comprobarlo.
3. Calificar las cinco predicciones de la clase 120 con evidencia.
4. Incorporar la ley 19 al cuestionario, con sus cuatro apariciones.
5. Escribir la predicción de la parte 11 en términos que se puedan desmentir.

Conceptos centrales

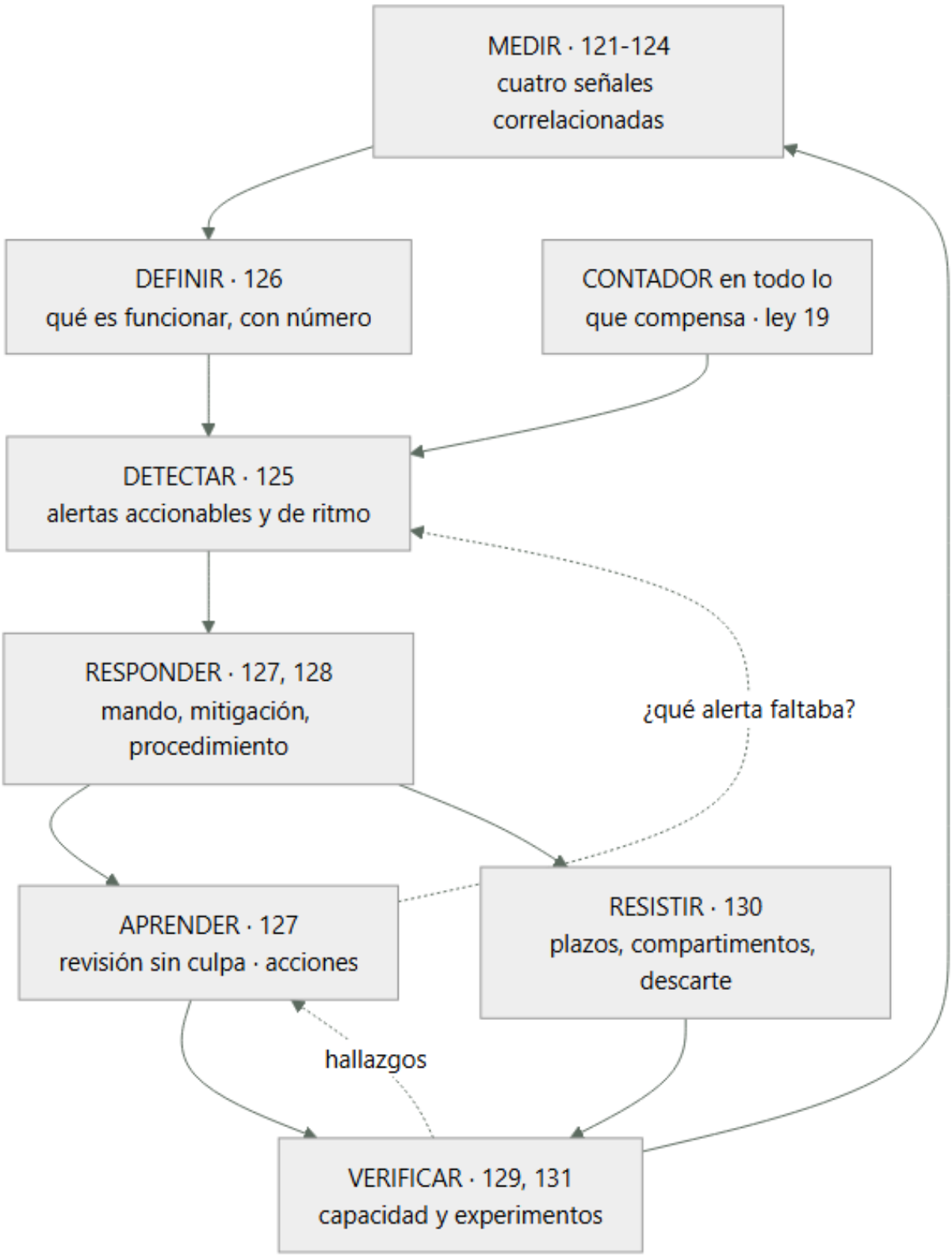
Concepto	Comprensión verificable
bucle de operación	Ciclo cerrado: medir, definir lo aceptable, detectar, responder, aprender y verificar. Cada pieza alimenta a la siguiente.
ley 19	Todo mecanismo que compensa un fallo automáticamente lo vuelve invisible. Sin contador, la compensación sustituye a la detección.
medio de detección	Cómo se supo de un problema. Es la medida de la operación, más que la duración del incidente.
calificación de hipótesis	Comparar lo predicho con lo ocurrido, publicando lo que se predijo mal y en qué medida.
coste de saber	Lo que cuesta la telemetría. Es un coste de ingeniería más y se gobierna como cualquier otro.
hipótesis de la parte 11	Predicción escrita ahora sobre lo que ocurrirá cuando el sujeto sea el control, el gobierno y el dinero.

Modelo mental

La confiabilidad es una característica del servicio percibido por usuarios; SRE la administra con objetivos explícitos y presupuestos de error.

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 operación es un bucle

Las once clases anteriores no son once herramientas: son un ciclo en el que cada pieza da de comer a la siguiente.

```
MEDIR       cuatro señales, correlacionadas, con coste gobernado   121-124
   ↓        y sin las cuales lo demás no es posible
DEFINIR     qué significa funcionar, con un número                 126
```

```
   ↓        y de ahí sale el resto del presupuesto
DETECTAR    pocas alertas, accionables, por ritmo de consumo       125
   ↓        y con dueño
RESPONDER   declarar pronto, mitigar antes de diagnosticar,
   ↓        un cambio cada vez                                     127
            con procedimientos que sirvan a quien no los escribió  128
APRENDER    revisión sin culpa, y la pregunta obligatoria:
   ↓        ¿qué alerta faltaba?                                   127
VERIFICAR   que la capacidad da y que los mecanismos funcionan     129, 131
   ↓
(vuelta a MEDIR)
```

Y las dos flechas que cierran el bucle son las que más se olvidan:

```
de APRENDER a DETECTAR    cada incidente produce la alerta que faltaba
                          → 48 de las 94 acciones de la clase 127
de VERIFICAR a APRENDER   cada experimento produce hallazgos
                          → 41 en seis meses en la clase 131
```

Sin ellas, el conjunto es estático: detecta lo que alguien anticipó el primer día y nada más.

Y una pieza transversal que atraviesa todo el bucle y que esta parte ha descubierto por el camino:

```
CUANTO COMPENSA UN FALLO NECESITA UN CONTADOR
  reinicios automáticos, reintentos, autoescalado, reversión automática,
  reconciliación, cortacircuitos, alternativas
→ es el apartado cuarto de esta clase
```

Y el coste, que también forma parte del bucle y no de un anexo:

```
saber lo que pasa cuesta dinero
y el coste se decide al EMITIR, no al consultar         clase 121
→ se gobierna como cualquier otro coste de ingeniería,
  con la misma pregunta: ¿quién consulta esto?
```

2. El proyecto

Montar la operación completa sobre un servicio real. Lo que hay que entregar:

```
1. SEÑALES
   cuatro señales con identificadores comunes
   línea de cambios con los cinco orígenes
   línea ancha por unidad de trabajo, con lista de permitidos
   métricas sin etiquetas de alta cardinalidad, con límite de series
   trazas con propagación en colas y trabajos programados
   y la cuenta de lo que cuesta todo eso

2. OBJETIVOS
   de dos a cuatro indicadores, medidos en el borde
   objetivo fijado tras medir, con el techo de dependencias calculado
   regla de parada aceptada por escrito antes de necesitarla

3. DETECCIÓN
   alertas de ritmo de consumo en cuatro ventanas
   alertas de antigüedad para todo lo que puede dejar de ejecutarse
   alertas de ausencia de datos
   toda alerta con dueño y procedimiento enlazado

4. RESPUESTA
   criterios de declaración y gravedad escritos
   papeles, canal automático y catálogo de mitigaciones
   comunicación externa con cadencia

5. VERIFICACIÓN
   codo medido con modelo abierto y datos realistas
   prueba de resistencia de varias horas
   inventario de lo que no escala solo, con tiempos
   los cinco mecanismos de resiliencia, en el orden correcto
   catálogo de experimentos, ejecutándose solo
   un ensayo con personas al trimestre
```

Y las preguntas cuya respuesta hay que escribir:

```
¿cómo se detectó el último incidente? ¿y el anterior?
¿cuántas alertas por turno de guardia?
¿qué proporción de lo emitido no consulta nadie?
```

```
¿cuánto cuesta la telemetría frente al cómputo?
¿cuál es el codo, y a qué distancia está el pico habitual?
¿qué mecanismos compensan fallos y cuántas veces actuaron ayer?
¿qué pasa si se cae el sistema de observabilidad?
```

La última se comprueba, no se razona: es un experimento de la clase 131.

Y las pruebas negativas de esta parte:

- parar un servicio y comprobar que salta una alerta de ausencia
- desplegar una versión con un 2 % de errores y medir cuánto tarda en avisar el ritmo de consumo
- inyectar latencia en una dependencia opcional
- parar un consumidor y esperar la alerta de antigüedad
- ejecutar un procedimiento con alguien que no lo escribió
- tirar el recolector de telemetría y comprobar que nada más se cae
- vaciar el caché en hora punta
- comprobar que el botón de parada de experimentos funciona

3. Calificación de las cinco predicciones

Predicción 1: «la parte 10 no introducirá fallos nuevos; hará visibles los de las partes anteriores. Al menos la mitad de sus ejemplos serán problemas ya documentados, vistos desde la detección».

```
veredicto: EQUIVOCADA, y el error es interesante
```

El reparto real de los problemas tratados en las clases 121 a 131:

```
problemas de las partes anteriores, vistos desde la detección        11
problemas DE LA PROPIA MAQUINARIA de observabilidad y operación      17
  cardinalidad que tumba el sistema de métricas          123
  percentiles promediados que mentían por un factor 11   123
  intervalos de histograma que hacían inútil el p99      123
  99,8 % de los errores que nadie miraba                 122
  registro que amplificó un fallo parcial a total        122
  340 alertas con 11 % accionables                       125
  46 silencios sin fecha                                 125
  modelo cerrado que mintió por un factor 4              129
  procedimientos que nadie sin permisos podía ejecutar   128
  cortacircuitos abierto 3 h por su propia configuración 130
  botón de parada que no funcionaba                      131
```

La mitad no: algo menos de cuatro de cada diez. Y lo que la predicción no vio es lo que hace útil el error: el instrumental para saber si el sistema funciona tiene sus propios modos de fallo, y son tan numerosos como los del sistema. Un panel puede mentir, una prueba de carga puede mentir y una alerta puede estar dirigida a un equipo que ya no existe.

Predicción 2: «de 3 de 21 detectados por alerta, subir por encima de 12 de 21, y más por síntomas que por causas».

```
veredicto: ACERTADA en la cifra, EQUIVOCADA en el motivo

incidentes detectados por alerta        antes  5 de 14   después  10 de 11
```

Y el reparto de esas diez detecciones por tipo de alerta:

```
alertas de ANTIGÜEDAD (algo dejó de ejecutarse)          5
alertas de síntoma (errores, latencia, ritmo de consumo) 4
indicador adelantado (saturación, cuota)                 1
```

Se predijo que ganarían los síntomas. Ganaron las alertas de antigüedad, que técnicamente son alertas de causa, y que existen por una razón concreta de este programa: la ley 13 lleva quince apariciones y produce problemas que no tienen síntoma hasta mucho después.

Predicción 3: «dominará la ley 15, y el trabajo REDUCIRÁ el número de alertas».

```
veredicto: ACERTADA en las dos partes

ley 15 en la parte 10        5 apariciones
  78 % de las series sin consultar             121
  99,8 % de los errores sin mirar              122
  8,4 millones de series                       123
  340 alertas, 29 por turno                    125
  118 paneles, 29 abiertos                     125

ley 13 en la parte 10        4 apariciones
  ausencia de datos que no dispara nada        123
  procedimientos que se pudren                 128
```

```
  experimentos que dejan de ejecutarse         131
  alertas de antigüedad como categoría propia  125
```

Y la reducción, con cifras:

```
alertas configuradas        340  →   73
disparos por semana         410  →   19
paneles                     118  →   34
series de métricas      8,4 M    →  310.000
```

Predicción 4: «lo más difícil será definir qué significa funcionar, y la mayoría de las alertas existentes medirán causas técnicas».

```
veredicto: ACERTADA
```

```
las 5 alertas más frecuentes, todas causas técnicas    85 % del volumen
alertas que no exigían ninguna acción                  184 de 340
objetivo que se quería prometer                        99,99 %
objetivo posible según las dependencias                99,05 %
trabajo necesario para poder prometer 99,5 %     rediseñar 3 dependencias
```

La última fila es la confirmación más clara: definir el número obligó a cambiar la arquitectura, no a esforzarse más.

Predicción 5: «la factura de telemetría será un problema, del orden de un porcentaje de dos cifras del coste de cómputo, y su causa será guardar todo por si acaso».

```
veredicto: ACERTADA en la causa, MUY CORTA en la magnitud
```

```
coste de cómputo                                    9.200 €/mes
registros                                           3.100 €
métricas                                            2.900 €
trazas y otros                                        410 €
                                                    ████████
telemetría                                          6.410 € = 70 %
```

Setenta por ciento, no «dos cifras». Y la causa fue exactamente la predicha:

```
78 % de las series no se consultaba nunca
14 líneas de registro por petición en vez de una
69 % de los registros sin estructurar
todo caliente durante 90 días

tras la parte 10:  1.110 €/mes = 12 % del cómputo
```

4. La ley 19, el recuento y la hipótesis de la parte 11

Una regularidad ha aparecido cuatro veces en esta parte y es lo bastante distinta de la ley 13 como para tener número propio:

```
LEY 19
  Todo mecanismo que compensa un fallo automáticamente lo vuelve invisible.
  Sin un contador, la compensación sustituye a la detección.
```

Sus cuatro apariciones:

```
clase 128   4.180 reinicios automáticos al día durante 4 meses
            disponibilidad medida: 99,7 %; el panel, verde
clase 124   11 % de las peticiones pagaban un reintento oculto
            ninguna métrica de error lo mostraba: terminaban bien
clase 130   cortacircuitos abierto 3 h 14 tras recuperarse la dependencia
            el servicio «funcionaba», degradado, y nadie lo sabía
clase 128   3 de 4 reparaciones automáticas tapaban un problema distinto
```

Y su diferencia con la ley 13, que conviene tener clara:

```
ley 13   algo DEJA DE FUNCIONAR y no da error
ley 19   algo SIGUE FUNCIONANDO demasiado bien y tapa que hay un problema
```

Y lo que añade al cuestionario:

```
¿qué mecanismos de este sistema compensan fallos automáticamente?
¿cuántas veces actuaron ayer?
¿esa cifra sube?
¿hay un límite a partir del cual dejan de compensar y avisan?
```

Recuento tras la parte 10:

```
ley 13  el bucle que no corre no da error                      19
ley 15  una señal con demasiados elementos deja de ser señal   17
```

```
ley 16  un control que estorba acaba desactivado o rodeado       10
ley 14  las decisiones de creación son irreversibles              9
ley 11  lo que entra en un sistema de solo-añadir se queda        7
ley 18  lo asíncrono traslada la garantía, no la elimina          5
ley 17  la medida que se vuelve objetivo se alcanza sin mejorar   5
ley 19  lo que compensa un fallo lo vuelve invisible              4
        NUEVA en esta parte
```

La hipótesis de la parte 11. La parte siguiente cambia el sujeto a control, gobierno y dinero. La predicción, escrita para poder desmentirla:

1. La ley dominante será la 16 —un control que estorba acaba desactivado o rodeado—, porque la parte 11 trata precisamente de controles.
   → predigo que MÁS DE LA MITAD de sus ejemplos serán controles implantados y luego rodeados, no controles ausentes

2. En la parte de coste, el gasto mayor NO será el que nadie estaba optimizando.
   → predigo que la partida más grande será capacidad comprada para un pico que ya no ocurre, o datos que nadie lee
   → y que el mayor ahorro individual superará a la suma de los tres siguientes

3. La ley 19, recién estrenada, reaparecerá en forma financiera: un mecanismo automático —autoescalado, ciclo de vida, aprovisionamiento— ocultando un problema de coste durante meses.

4. El problema más difícil será la ATRIBUCIÓN: saber de quién es cada coste y cada riesgo.
   → y predigo que la respuesta recurrente será el catálogo de la clase 095, igual que lo fue en las partes 08 y 10

5. Y la predicción que puede salir del revés: seguridad y coste resultarán ser EL MISMO PROBLEMA, porque los dos consisten en recursos que existen sin dueño y sin que nadie sepa por qué.

Y lo que se anota para calificar sin trampa:

```
lo que ya sabemos    que la ley 16 lleva 10 apariciones
lo que creemos       que el problema es de atribución, no de herramienta
lo que no sabemos    si la quinta predicción es una observación útil
                     o una frase bonita
```

Ejemplo trabajado

Se monta la operación completa sobre el sistema de las partes anteriores y se somete a las ocho pruebas negativas. Después, el recuento de la parte y la tabla de detección rehecha.

Las ocho pruebas negativas.

1. parar un servicio → ¿alerta de ausencia?
   sí, 70 s. CORRECTO

2. desplegar una versión con 2 % de errores → ¿ritmo de consumo?
   sí; ventana de 1 h con confirmación de 5 min: aviso a los 6 min
   CORRECTO

3. inyectar latencia en una dependencia opcional
   circuito abierto en 14 s, portada servida sin la sección, +38 ms
   CORRECTO

4. parar un consumidor → ¿alerta de antigüedad?
   sí, 4 min. Y la alerta fue a un equipo disuelto: HALLAZGO
   → revisados los 73 dueños; 11 estaban mal

5. ejecutar un procedimiento con alguien que no lo escribió
   68 de 73 sin preguntas; los 5 restantes exigían juicio y se marcaron como «escalar siempre». CORRECTO

6. tirar el recolector de telemetría
   la aplicación siguió sirviendo; se perdieron 4 min de trazas
   PERO ninguna alerta avisó de que no había datos durante 12 min
   → añadida alerta sobre el propio recolector. HALLAZGO

7. vaciar el caché en hora punta
   90 s de degradación, sin caída. CORRECTO

8. botón de parada de experimentos
   no funcionaba: faltaba un permiso. HALLAZGO, corregido antes
   de ejecutar nada

Cinco correctas y tres hallazgos. Los tres hallazgos son de la maquinaria de operación, no del sistema: un destinatario equivocado, una alerta que faltaba sobre la propia telemetría y un permiso.

La tabla de detección, rehecha.

La parte 09 cerró con veintiún problemas y esta cuenta:

por una alerta	3 de 21

Y los once incidentes del semestre posterior a la parte 10:

por una alerta de antigüedad	5
por una alerta de síntoma o de ritmo de consumo	4
por un indicador adelantado	1
por una reclamación de un cliente	1
por una caída total	0
por una auditoría	0
por la factura	0

Diez de once, y el único detectado por un cliente fue una funcionalidad rota que no afectaba a ningún indicador: el buscador devolvía resultados en orden incorrecto. Ninguna señal de esta parte lo habría visto, y eso también hay que decirlo.

El recuento de la parte 10.

señales instrumentadas	3 de 4	4 de 4
eventos de cambio al día	0	103
series de métricas	8,4 millones	310.000
volumen de registro diario	510 GB	98 GB
coste mensual de telemetría	6.410 €	1.110 €
telemetría frente a cómputo	70 %	12 %
trazas completas de extremo a extremo	31 %	96 %
indicadores definidos	0	11
servicios con objetivo	0 de 5	5 de 5
alertas configuradas	340	73
disparos por semana	410	19
alertas por turno de guardia	29	1,4
proporción accionable	11 %	78 %
paneles	118	34
incidentes detectados por alerta	5 de 14	10 de 11
tiempo medio hasta declarar	41 min	2 min
tiempo medio hasta mitigar	1 h 52	7 min
acciones de revisión completadas en plazo	13 %	84 %
trabajo repetitivo	34 %	11 %
capacidad real conocida	no	codo medido
caídas totales por una dependencia lenta	2 / 6 meses	0
disponibilidad del flujo de compra	99,21 %	99,74 %
experimentos en catálogo	0	23
hallazgos de experimentos	—	41
mecanismos de compensación con contador	0 de 5	5 de 5
incidentes en el semestre	6	2

Las tres cifras que mejor resumen la parte.

1. alertas: de 340 a 73, y de 5 de 14 detecciones a 10 de 11
   → menos señal y más detección son la misma cosa

2. telemetría: de 6.410 € a 1.110 € con más información disponible
   → el 78 % de lo emitido no lo consultaba nadie

3. reinicios automáticos: 4.180 al día durante 4 meses, panel en verde
   → la ley 19, y el hallazgo más caro de la parte

Y lo que la parte 10 no resolvió, dicho con claridad.

el buscador con resultados mal ordenados	ningún indicador lo cubre
la fuga entre usuarios del caché (clase 111)	la detectó una prueba, no la observabilidad

```
saber si lo entregado sirve para algo          clase 107, sigue abierto
```

Las tres tienen la misma forma: el sistema funciona correctamente según todas sus señales y aun así está mal. Ninguna de las doce clases de esta parte puede hacer nada al respecto, y conviene no fingir lo contrario.

La conclusión que cierra la parte 10: la operación pasó de detectar tres de veintiún problemas con una alerta a detectar diez de once, y lo hizo con menos alertas, menos paneles y una quinta parte del coste de telemetría. Lo que cambió no fue la cantidad de datos: fue haber decidido qué significa funcionar, haber contado lo que compensaba fallos en silencio y haber preguntado, después de cada incidente, qué alerta debería haber sonado.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-10-observability-sre-reliability/132-proyecto-operacion-sre-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 plataforma-sre en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

El artefacto principal es un incremento integrado, demostrable y documentado. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye plataforma-sre 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 compran herramientas de observabilidad y la detección no mejora	Se trató como una lista de herramientas y no como un bucle: falta definir qué significa funcionar y la vuelta de aprender a detectar	Cierra el bucle: cada incidente produce la alerta que faltaba y cada experimento produce hallazgos con dueño.
Los paneles están verdes y el sistema se deteriora	Ley 19: algo compensa el fallo automáticamente y nadie cuenta cuántas veces	Contador, límite y tendencia en todo lo que compense: reinicios, reintentos, autoescalado, reversión, reconciliación y cortacircuitos.
El coste de telemetría se acerca al de cómputo	Se emite todo por si acaso y el coste se decide al emitir	Mide qué proporción no consulta nadie, estructura, usa línea ancha, reduce cardinalidad y aplica retención por capas.
Las pruebas negativas se dan por hechas sin ejecutarlas	Se razona sobre lo que debería pasar en vez de provocarlo	Ejecuta las ocho: los hallazgos suelen estar en la maquinaria de operación, no en el sistema.
Se declara la operación terminada porque hay alertas y paneles	No se mide el medio de detección de los incidentes reales	Cuenta cómo te enteraste de cada uno; es la única medida honesta de la operación.
Se cree que la observabilidad detectará cualquier problema	Hay fallos con todas las señales correctas: resultados mal ordenados, fugas entre usuarios, funcionalidad inútil	Escribe explícitamente qué tipos de problema no cubre este instrumental y cúbrelos con pruebas y con medidas de resultado.

Seguridad, ética y costo

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

Preguntas de comprobación

1. ¿Cuáles son las dos flechas que cierran el bucle de operación y qué pasa sin ellas?
2. ¿En qué se equivocó la predicción de que la parte 10 solo haría visibles problemas antiguos?
3. ¿Qué tipo de alerta resultó ser la que más detecciones aportó, y por qué no fue la predicha?
4. ¿Qué dice la ley 19 y en qué se diferencia de la ley 13?
5. ¿Qué predice la hipótesis de la parte 11 sobre la ley dominante y sobre la atribución?

Referencias

- Beyer, B. y otros (2016). Site Reliability Engineering — el conjunto de prácticas como sistema, no como herramientas. <https://sre.google/sre-book/table-of-contents/>
- Beyer, B. y otros (2018). The Site Reliability Workbook — implantación práctica y medición del progreso. <https://sre.google/workbook/table-of-contents/>
- Majors, C. y otros (2022). Observability Engineering — coste, cardinalidad y preguntas nuevas. <https://www.oreilly.com/library/view/observability-engineering/9781492076438/>
- Woods, D. y Hollnagel, E. (2006). Resilience Engineering — por qué la compensación automática oculta el deterioro. <https://www.taylorfrancis.com/books/edit/10.1201/9781315605685/resilience-engineering>
- Allspaw, J. (2015). Trade-offs under pressure — decisiones durante incidentes y aprendizaje posterior. <https://www.adaptivecapacitylabs.com/blog/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 10 en PDF · Manual integral

Evaluación — 132 Proyecto: operación SRE 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 plataforma-sre 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.