Facsímil 06

Construir y operar

Capítulo 09

Incidencias, postmortems y mejora continua

686f6c61 · 2026

Capítulo 09: Incidencias, postmortems y mejora continua

Qué deberías poder hacer al terminar

En el capítulo 08 diseñamos cómo pausar una run y pedir revisión. Ahora subimos un nivel: **qué ocurre cuando el sistema completo entra en una situación operativa que exige coordinación**.

Una incidencia de IA no siempre es “la API está caída”. Puede ser más sutil: sube el coste por run aceptada, el contrato JSON falla, el RAG cita documentos que no corresponden, el router manda demasiado tráfico al modelo caro, una cola de revisión se atasca, un canary degrada la calidad en un segmento o una tool empieza a devolver errores tipados.

Al terminar, deberías poder hacer esto:

RESULTADO DE APRENDIZAJE	EVIDENCIA DE QUE LO SABES HACER
Declarar una incidencia de IA.	Separas síntoma, impacto, severidad, servicio afectado y owner.
Priorizar respuesta.	Clasificas por usuarios, tareas, coste, contrato, SLO y reversibilidad.
Coordinar roles.	Distingues comandante, operaciones, comunicación, especialistas y documentación.
Mitigar sin perder evidencia.	Proteges el servicio y conservas trazas, eventos y decisiones.
Calcular tiempos operativos.	Mides detección, reconocimiento, mitigación y recuperación.
Escribir un postmortem útil.	Incluyes impacto, línea temporal, causas contribuyentes y acciones verificables.
Convertir aprendizaje en trabajo.	Creas casos de regresión, runbooks, alertas, cambios de política y gates.

La idea central: **una incidencia no termina cuando vuelve la gráfica; termina cuando el sistema aprende algo verificable**.

Cuando producción te enseña lo que faltaba

Imagina que un asistente de soporte llevaba dos semanas estable. De pronto, el coste p95 sube, la latencia se dispara y soporte empieza a recibir tickets de respuestas que tardan demasiado. Nadie cambió código de aplicación, pero sí se publicó un índice RAG nuevo y se abrió un canary de prompt.

Hay varias tentaciones: mirar logs al azar, culpar al modelo, revertir todo, escribir en el chat “lo estoy mirando” y esperar que alguien encuentre la causa. Eso no es operar. Operar es declarar el evento, acotar impacto, asignar roles, mitigar, preservar evidencia y dejar un registro que permita aprender.

La pregunta no es “quién lo rompió”. La pregunta útil es:

> ¿Qué señales vimos, qué decisiones tomamos, qué mitigó el impacto y qué cambio evitará que vuelva igual?

Qué no es gestionar una incidencia

Gestionar una incidencia no es correr más rápido que el dashboard. La velocidad sin coordinación suele crear cambios simultáneos, mensajes contradictorios y pérdida de evidencia.

Tampoco es buscar una causa única al final. En sistemas de IA, una incidencia suele tener causas contribuyentes: un cambio de datos, un umbral demasiado permisivo, un fallback que no estaba probado, una alerta tardía, un runbook incompleto o un canary que no segmentaba lo suficiente.

Y un postmortem no es un documento ceremonial. Si no produce acciones verificables, casos de regresión, alertas mejores o runbooks más claros, solo archiva una historia.

CONFUSIÓN	QUÉ FALTA
“La incidencia termina cuando baja la alerta”	Falta confirmar recuperación, cerrar comunicación y capturar aprendizaje.
“Postmortem es resumen”	Falta impacto, línea temporal, causas contribuyentes y acciones con dueño.
“Revertimos y listo”	Falta saber qué señal detectó tarde y qué dataset debe aprender.
“El modelo falló”	Falta distinguir modelo, prompt, RAG, router, runtime, contrato y proveedor.
“Ya sabemos lo que pasó”	Falta evidencia reproducible: trazas, métricas, eventos y decisiones.

Qué sí es una incidencia de IA

Para este facsímil, una incidencia de IA es:

> Un evento operativo que degrada una propiedad importante del sistema: disponibilidad, latencia, coste, calidad, contrato, trazabilidad, permisos o confianza del flujo.

Una incidencia se modela con estos campos. No es una ecuación de la literatura, es una lista de ingeniería:

SÍMBOLO	SIGNIFICADO	EJEMPLO
S	Servicio o capacidad afectada.	`support_rag`, `tool_gateway`, `review_queue`.
P	Población afectada.	18% de tenants, canal chat, tarea `policy_answer`.
T	Ventana temporal.	13:40-14:35 UTC.
E	Evidencia observada.	Métricas, logs, trazas, tickets, decision records.
M	Mitigaciones aplicadas.	Rollback, fallback, bajar canary, desactivar tool.
D	Decisiones tomadas.	Quién decidió, cuándo y por qué.
A	Acciones de mejora.	Evals, alertas, runbooks, cambios de política.

Si falta D, no sabremos por qué se actuó. Si falta E, discutiremos recuerdos. Si falta A, la incidencia puede repetirse con otro nombre.

Fecha de corte del estado del arte

Fecha de corte: 28 de mayo de 2026.
Fuentes consultadas: capítulos de Google SRE sobre monitorización, respuesta de emergencia, gestión de incidencias, postmortems, alerting on SLOs y sobrecarga; OpenTelemetry sobre trazas, logs, métricas y convenciones GenAI; NIST AI RMF; y trabajos de ingeniería de ML.

El libro de SRE de Google recomienda que las alertas que interrumpen a personas sean simples, accionables y orientadas a síntomas relevantes para usuarios.[1] El SRE Workbook explica cómo alertar sobre SLOs usando consumo de presupuesto de error, no solo umbrales aislados.[2]

Google SRE también insiste en preparar la respuesta a emergencias antes de necesitarlas, practicarla y pedir ayuda cuando la situación supera a una persona.[3] En gestión de incidencias, separa roles como comando, trabajo operativo, comunicación y planificación para evitar cambios descoordinados.[4]

Sobre postmortems, Google SRE los define como registros escritos de impacto, acciones tomadas, causas contribuyentes y seguimiento para evitar recurrencia, con una cultura centrada en aprender y mejorar sistemas.[5] Lo estable es el método: declarar, coordinar, mitigar, documentar, aprender y verificar. Lo cambiante son herramientas concretas de guardia, chat, tickets, dashboards y proveedores.

Severidad: no todo merece el mismo ruido

La severidad debe estar definida antes de la incidencia. Si se decide en caliente, cada persona trae su propio umbral.

SEVERIDAD	IMPACTO	EJEMPLO EN IA	RESPUESTA
SEV1	Servicio principal no cumple función crítica o hay impacto amplio.	El asistente no responde o rompe contrato en producción para muchas tareas.	Comando formal, mitigación inmediata, comunicación periódica.
SEV2	Degradación importante, acotada o con workaround.	p95 fuera de SLO en una tarea clave; coste p95 se duplica.	Equipo operativo, owner, update regular, postmortem si supera umbral.
SEV3	Problema limitado, sin impacto amplio.	Una ruta RAG falla en un tenant o canal.	Ticket prioritario, runbook, revisión posterior.
SEV4	Anomalía o mejora preventiva.	Alerta de tendencia, flake rate subiendo, cola creciendo.	Trabajo planificado antes de que escale.

Para IA conviene clasificar por más dimensiones que “está caído”:

DIMENSIÓN	PREGUNTA
Disponibilidad	¿El usuario puede completar la tarea?
Latencia	¿La experiencia sigue dentro del SLO?
Contrato	¿La salida sigue siendo parseable y compatible?
Calidad	¿Aumentan rechazos, revisiones o quejas verificables?
Coste	¿Se consume presupuesto de forma anómala?
Trazabilidad	¿Podemos reconstruir lo ocurrido?
Reversibilidad	¿Podemos volver a estado conocido sin pérdida?
Alcance	¿Cuántos tenants, tareas, idiomas o canales afecta?

Fórmulas operativas que sí usaría

El primer grupo mide tiempos:

$$MTTA = \frac{1}{n}\sum_{i=1}^{n}(t_i^{ack} - t_i^{detect})$$

$$MTTR = \frac{1}{n}\sum_{i=1}^{n}(t_i^{recover} - t_i^{start})$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$MTTA$	Tiempo medio hasta reconocer la incidencia.	7 minutos.
$MTTR$	Tiempo medio hasta recuperar.	42 minutos.
t_i^{detect}	Momento en que el sistema detecta el evento.	13:43 UTC.
t_i^{ack}	Momento en que una persona o proceso lo reconoce.	13:48 UTC.
t_i^{start}	Inicio de la incidencia.	13:40 UTC.
$t_i^{recover}$	Recuperación confirmada.	14:22 UTC.
n	Número de incidencias medidas.	12.

La prioridad de una incidencia es una puntuación ponderada de varios factores, una decisión multicriterio.[6]

$$P = 4U + 3C + 2L + 2K + R$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
P	Puntuación de prioridad.	23.
U	Alcance de usuarios o tenants afectados, de 0 a 5.	3.
C	Criticidad de la tarea, de 0 a 5.	4.
L	Degradación de latencia o disponibilidad, de 0 a 5.	2.
K	Degradación de contrato/calidad, de 0 a 5.	3.
R	Penalización por baja reversibilidad, de 0 a 5.	1.

Ejemplo:

$$P = 4 \cdot 3 + 3 \cdot 4 + 2 \cdot 2 + 2 \cdot 3 + 1 = 35$$

Con $P = 35$, probablemente no estamos ante un ticket normal. Necesitamos coordinación, mitigación y seguimiento.

También conviene recordar que las colas obedecen matemáticas. La ley de Little relaciona trabajos en sistema, tasa de llegada y tiempo medio.[7] Si la cola de revisión o de serving crece, no basta con "esperar un poco": o baja llegada, o sube capacidad, o aumenta tiempo de espera.

Cuándo esperar el próximo fallo: fiabilidad exponencial. Si los fallos ocurren al azar a tasa constante, el tiempo hasta el siguiente sigue una distribución exponencial, y la probabilidad de aguantar sin incidencias decae con el tiempo de forma conocida.[8]

$$R(t) = e^{-\lambda t}, \qquad \text{MTBF} = \frac{1}{\lambda}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
λ	Tasa de incidencias por unidad de tiempo.	0,5 por semana.
t	Horizonte de tiempo considerado.	2 semanas.
$R(t)$	Probabilidad de no tener incidencias hasta t.	$e^{-1} \approx 0{,}37$.
MTBF	Tiempo medio entre incidencias.	2 semanas.

En palabras: con media incidencia por semana, la probabilidad de pasar dos semanas limpias es solo del 37%. Esto desmonta el «llevamos una semana sin incidencias, vamos bien»: una racha corta es ruido esperable, no evidencia de mejora. Para afirmar que mejoras necesitas ver bajar λ sostenidamente, no una buena semana suelta.

Qué causas atacar primero: el principio de Pareto. No todas las causas de incidencia pesan igual. La ley de Pareto, observada una y otra vez en sistemas operativos, dice que una minoría de causas explica la mayoría de los fallos, lo que da una regla de priorización para los postmortems.[9]

$$P(X > x) = \left(\frac{x_m}{x}\right)^{\alpha}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
x_m	Valor mínimo del fenómeno.	1 incidencia.
α	Parámetro de forma (concentración).	$\approx 1{,}16$ para el 80/20.
$P(X > x)$	Probabilidad de superar x.	Cola pesada: pocas causas, mucho efecto.

En palabras: si el 20% de las causas provoca el 80% de las incidencias, ordenar los postmortems por frecuencia y atacar primero esa minoría rinde mucho más que repartir esfuerzo por igual. La mejora continua no es arreglar todo, es arreglar lo que más pesa, y Pareto dice cuánto pesa cada cosa.

¿Es buena tu operación? Las métricas DORA. Para saber si operas bien necesitas un marco comparable, no una sensación. La investigación DORA identificó cuatro métricas que distinguen a los equipos de alto rendimiento; dos son directamente de incidencias: cada cuánto fallan los cambios y cuánto tardas en recuperarte.[10]

$$\mathrm{CFR} = \frac{\text{despliegues con incidencia}}{\text{despliegues totales}}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
CFR	*Change Failure Rate*: fracción de cambios que provocan incidencia.	0,10
MTTR	Tiempo medio de recuperación de un cambio fallido.	menos de 1 hora en equipos de élite.
despliegues totales	Cambios desplegados en la ventana.	200

En palabras: un CFR del 10% y un MTTR de minutos caracterizan a un equipo que entrega rápido y se recupera antes; un CFR alto con MTTR de días, lo contrario. Lo valioso es que son comparables entre equipos y a lo largo del tiempo: te dicen si tu mejora continua se nota en los números o solo en las reuniones.

¿Estás mejorando de verdad? Crecimiento de fiabilidad de Crow-AMSAA. Si cada postmortem corrige causas, el sistema debería fallar cada vez menos. El modelo de Crow-AMSAA cuantifica esa mejora: el número acumulado de fallos sigue una ley de potencia cuyo exponente dice si estás mejorando, estancado o empeorando.[11]

$$N(t) = \lambda\, t^{\beta}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$N(t)$	Número acumulado de incidencias hasta t.	Curva que se aplana si mejoras.
β	Exponente de crecimiento.	<1 mejora; $=1$ estable; >1 empeora.
λ	Escala inicial.	Ajustada a los datos.

En palabras: un exponente menor que 1 significa que el tiempo entre incidencias crece, es decir, que los postmortems funcionan; mayor que 1 significa que el sistema se degrada más rápido de lo que lo arreglas. Es la forma honesta de responder «¿la mejora continua está funcionando?» con la curva real de incidencias, no con la intención.

Tipos de incidencia propios de IA

TIPO	SÍNTOMA	PRIMERAS COMPROBACIONES
Contrato	JSON inválido, campos extra, enum incorrecto.	`contract_version`, salida raw, validador, cambio de prompt/modelo.
Calidad	Más rechazos, revisiones o casos corregidos.	Muestras, judge calibrado, feedback, canary, dataset afectado.
RAG	Fuentes pobres, citas ausentes, documentos obsoletos.	`index_version`, retrieval trace, `top_k`, reranker, fechas de documento.
Routing	Ruta cara o lenta sube de golpe.	`route_catalog`, distribución de tareas, fallback, proveedor.
Coste	Coste p95 o total diario sube.	Tokens, reintentos, contexto, modelo, cache hit rate.
Serving	Saturación de workers, cola, KV cache, timeouts.	Goodput, queue depth, GPU/CPU, batch size, p95/p99.
Handoff	Cola de revisión crece o caduca.	Backlog, `needs_more_info_rate`, owners, SLO de cola.
Trazabilidad	No se puede reconstruir la run.	`trace_id`, spans, sampling, logs, atributos obligatorios.

Dean y Barroso mostraron que la cola de la latencia importa especialmente en sistemas a gran escala: una parte pequeña de operaciones lentas puede dominar la experiencia.[1][2] En IA esto se agrava por contexto largo, herramientas, colas de serving, reintentos y validación.

Taxonomía por capa: dónde mirar primero

Un ingeniero de IA necesita separar capas. Si todo se llama “fallo del modelo”, el postmortem no produce buenas acciones.

CAPA	QUÉ PUEDE DEGRADARSE	SEÑALES	MITIGACIÓN TÍPICA
Producto	Usuario no completa tarea.	Tickets, abandono, baja aceptación.	Respuesta de estado, handoff, rollback de experiencia.
Contrato	Consumidor no parsea salida.	`contract_fail_rate`, errores de schema.	Schema anterior, validador estricto, prompt rollback.
Prompt	Cambia formato, tono, abstención o longitud.	Tokens, longitud, fallos de JSON, quejas.	Prompt anterior, límite de salida, eval de regresión.
Modelo	Latencia, coste, calidad o tool calling cambian.	`model_id`, p95, coste, flake rate.	Cambiar ruta, bajar esfuerzo, fallback de modelo.
RAG	Recuperación trae ruido o documentos obsoletos.	`index_version`, chunks, citas, recall manual.	Índice anterior, bajar `top_k`, desactivar reranker.
Router	Demasiado tráfico va a ruta cara/lenta.	`route_mix`, fallback, coste por tarea.	Catálogo anterior, override por tarea.
Tool gateway	Args inválidos, permisos o errores externos.	`tool_error_rate`, args, scope, retries.	Modo lectura, bloquear tool, handoff.
Serving	Saturación, colas, timeouts, memoria.	queue depth, p95/p99, goodput, GPU/CPU.	Rate limit, autoscaling, reducir contexto.
Observabilidad	Faltan trazas o atributos.	`trace_completeness`, sampling, logs.	Subir sampling temporal, añadir atributos obligatorios.

Checklist de primera hora:

1. ¿Qué cambió en las últimas 24 horas: prompt, modelo, índice, router, proveedor, política, código o tráfico?
2. ¿El síntoma aparece por `task`, `tenant`, `model_id`, `prompt_version`, `index_version` o `release_id`?
3. ¿La mitigación más reversible está clara?
4. ¿Tenemos suficiente evidencia antes de borrar o sobrescribir estado?
5. ¿Qué caso concreto debería entrar en regresión?

Queries reales durante la incidencia

Las incidencias se operan mejor con preguntas concretas. Ejemplos con PromQL:

```
histogram_quantile(
  0.95,
  sum(rate(ai_run_latency_seconds_bucket{task="support_reply"}[10m])) by (le, release_id,
variant)
)
```

```
sum(rate(ai_contract_fail_total{task="support_reply"}[10m])) by (release_id, variant)
/
sum(rate(ai_run_total{task="support_reply"}[10m])) by (release_id, variant)
```

```
histogram_quantile(
  0.95,
  sum(rate(ai_run_cost_eur_bucket{task="support_reply"}[30m])) by (le, model_id, route_id)
)
```

```
sum(rate(ai_tool_error_total{tool_name="crm_lookup"}[10m])) by (error_type, release_id)
```

Y con SQL para eventos de producto:

```
select
  task,
  release_id,
  variant,
  count(*) as runs,
  avg(case when contract_valid then 0 else 1 end) as contract_fail_rate,
  avg(case when accepted_by_user then 1 else 0 end) as acceptance_rate,
  percentile_cont(0.95) within group (order by latency_ms) as latency_p95_ms
from ai_run_events
where created_at >= now() - interval '2 hours'
group by task, release_id, variant
order by contract_fail_rate desc, latency_p95_ms desc;
```

Estas consultas deberían vivir en el runbook. Si las inventas durante la incidencia, perderás minutos y quizá harás preguntas distintas cada vez.

Replay: reproducir antes de prometer causa

Una incidencia de IA no queda entendida hasta que podemos reproducir al menos una parte. No siempre se puede reproducir todo, pero sí guardar un paquete mínimo.

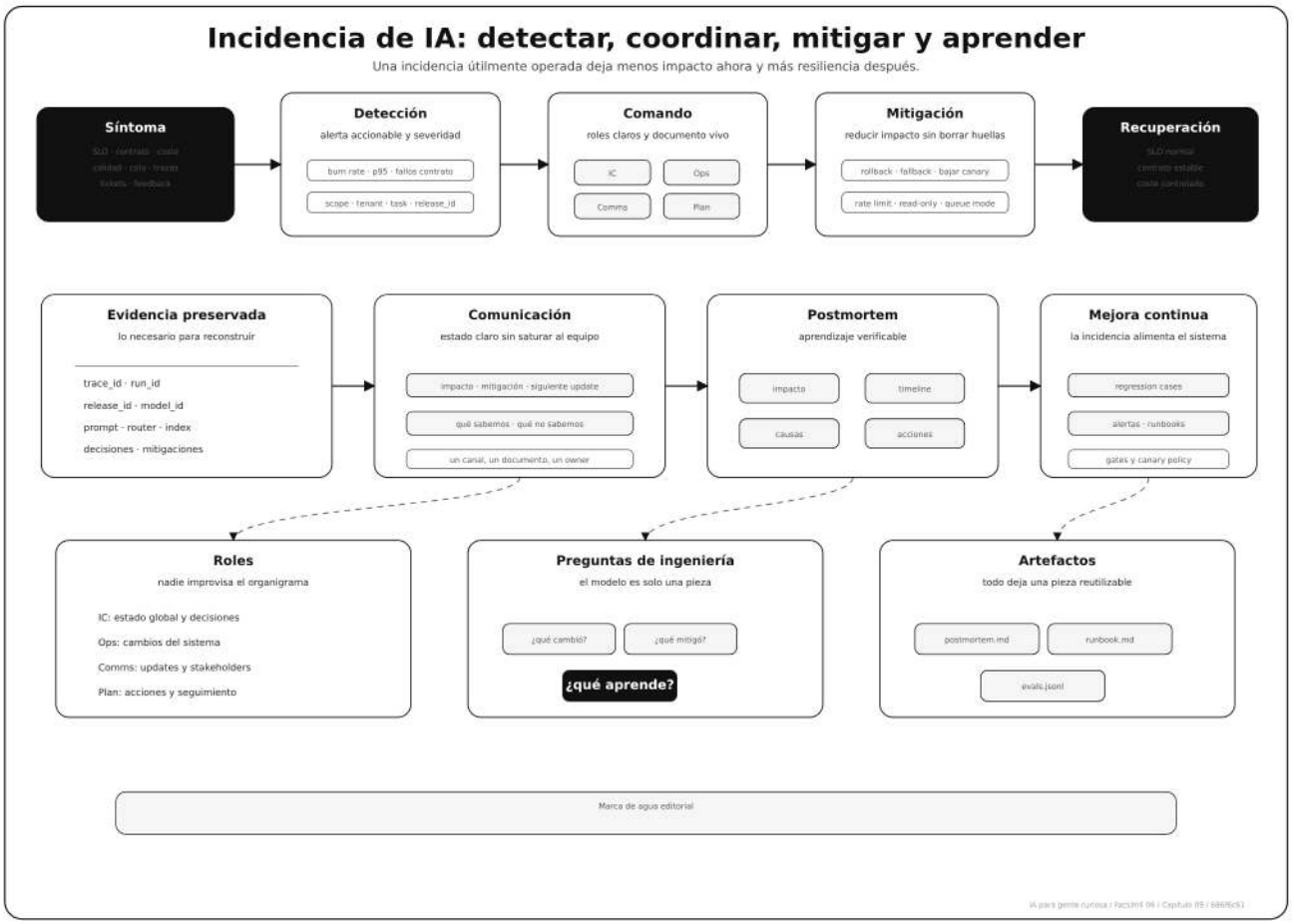
PIEZA	QUÉ GUARDAR	POR QUÉ
Entrada	Prompt de usuario o payload redactado.	Permite repetir el caso.
Contexto	IDs de chunks, documentos, versiones y hashes.	RAG cambia con el tiempo.
Configuración	Modelo, parámetros, prompt, router, tools.	Sin versiones no hay comparación.
Salida	Respuesta raw, salida validada y errores.	Permite medir contrato y calidad.
Traza	`trace_id`, spans y eventos.	Reconstruye latencia, retries y tools.
Decisión	Mitigación, rollback o handoff aplicado.	Explica por qué terminó así.

Plantilla `replay_case.json`:

```
{
  "case_id": "inc_support_rag_2026_05_28_case_001",
  "source": "incident",
  "task": "support_reply",
  "release_id": "support-rag@1.9.0-rc1",
  "input_hash": "sha256:...",
  "prompt_version": "prompt_v13",
  "model_id": "model_b",
  "route_catalog": "route_catalog@32",
  "rag_index": "rag_index_2026_06",
  "expected": {
    "contract_valid": true,
    "must_cite_source": true,
    "max_latency_ms": 4200
  },
  "observed": {
    "contract_valid": false,
    "latency_ms": 6200,
    "missing_source": true
  }
}
```

Este archivo es el puente entre postmortem y EvalOps. Si no hay replay, el aprendizaje se queda en memoria oral.

Anatomía visual de una incidencia de IA



Roles: separar trabajo para pensar mejor

Durante una incidencia, mezclar todos los roles crea ruido. Google SRE describe roles como comando, trabajo operativo, comunicación y planificación.[13] En IA los adaptaría así:

ROL	RESPONSABILIDAD	NO DEBERÍA HACER
Comando de incidencia	Mantener estado global, severidad, decisiones y prioridades.	Tocar todos los mandos técnicos a la vez.
Operaciones	Ejecutar mitigaciones: rollback, fallback, rate limit, desactivar ruta.	Comunicar estimaciones no validadas.
Especialista IA	Analizar prompt, modelo, RAG, evals, router o tool.	Cambiar producción sin coordinar.
Comunicación	Explicar impacto, mitigación y próximo update.	Especular causas antes de evidencia.
Planificación	Registrar acciones, owners, follow-ups y postmortem.	Perderse en debugging de bajo nivel.

La regla de oro: **una persona coordina, pocas personas cambian producción, todas las decisiones quedan escritas**.

Mitigación: proteger primero, explicar después

La mitigación reduce impacto. No siempre corrige la causa profunda. Está bien. Durante una incidencia, primero evitamos que el daño operativo crezca.

SÍNTOMA	MITIGACIÓN INMEDIATA	EVIDENCIA A PRESERVAR
Contrato JSON falla.	Volver a prompt/schema anterior o activar validador estricto.	Salidas raw, `schema_version`, modelo, prompt.
Coste sube.	Bajar `max_tokens`, cambiar ruta, activar cache, pausar canary.	Tokens, route mix, provider, retries.
Latencia p95 sube.	Fallback a ruta ligera, limitar contexto, bajar batch, rate limit.	Queue depth, spans, p95 por fase.
RAG trae fuentes pobres.	Volver a índice anterior o bajar `top_k`.	Queries, chunks, index_version, reranker.
Cola de revisión caduca.	`queue_only_critical`, reasignar, respuesta de estado.	Backlog, owners, `needs_more_info_rate`.
Tool externa falla.	Modo solo lectura, fallback sin tool, abrir handoff.	Tool args, error tipado, trace_id.

El libro de SRE sobre sobrecarga recuerda que un sistema debe decidir qué trabajo aceptar, retrasar o rechazar para protegerse.[14] En IA esto incluye rechazar generación cara, pausar rutas, pedir revisión o responder con estado claro.

Rollback, roll forward o contención

No toda incidencia pide la misma respuesta. Para un ingeniero de IA, la decisión importante es elegir la acción menos destructiva que reduzca impacto.

SITUACIÓN	ACCIÓN PREFERENTE	MOTIVO
Prompt nuevo rompe contrato.	Rollback de prompt.	Cambio reversible y localizado.
Índice RAG nuevo trae fuentes malas.	Rollback de índice o shadow de retrieval.	No hace falta tocar todo el servicio.
Modelo nuevo degrada p95.	Cambiar ruta o bajar porcentaje.	Mantienes producto mientras analizas.
Tool externa falla.	Contención: modo lectura o fallback sin tool.	Evitas efectos parciales.
Coste sube por contexto largo.	Límite de contexto, cache o ruta barata.	Reduce sangrado mientras investigas.
Causa no entendida y alcance crece.	Contención amplia y preservar evidencia.	Prioriza impacto y trazabilidad.
Fix pequeño probado.	Roll forward con canary nuevo.	Evita volver a estado antiguo si el parche es más seguro.

Regla práctica:

```
si el cambio causante es conocido y reversible -> rollback específico
si el causante es desconocido y el impacto crece -> contención
si el fix es pequeño, probado y observable -> roll forward controlado
```

La peor respuesta suele ser cambiar tres cosas a la vez y perder la capacidad de saber cuál ayudó.

Guardia operativa y handoff de turno

Una incidencia puede durar más que una persona. El cambio de turno también necesita contrato.

`oncall_handoff.md` debería incluir:

```
# Handoff de guardia

## Estado actual

- Incidencia:
- Severidad:
- Servicio:
- Comando:
- Último update:
- Próximo update:

## Qué sabemos

- Síntomas confirmados:
- Cambios recientes:
- Mitigaciones aplicadas:
- Estado de SLO:

## Qué no sabemos todavía

- Hipótesis abiertas:
- Evidencia pendiente:

## No tocar sin coordinar

- Rutas:
- Índices:
- Prompts:
- Proveedores:

## Siguiente acción recomendada

Paso concreto para los próximos 15-30 minutos.
```

Esto parece pequeño, pero evita uno de los fallos más caros: que el turno nuevo repita investigación o revierta una mitigación que estaba funcionando.

Qué te llevas para poner en práctica

El alumno debería salir con un kit mínimo de incidencia para IA:

ARTEFACTO	PARA QUÉ SIRVE	QUÉ CONTIENE
`incident_events.jsonl`	Entrada reproducible.	Alertas, mitigaciones, cambios, recuperación y evidencias.
`incident_review.py`	Análisis ejecutable.	Severidad, timeline, MTTA, MTTR, mitigaciones y acciones.
`severity_matrix.yaml`	Criterio previo.	SEV1-SEV4 por disponibilidad, contrato, coste, calidad y alcance.
`incident_state.md`	Documento vivo.	Estado actual, owner, impacto, acciones y próximo update.
`postmortem.md`	Cierre de aprendizaje.	Impacto, línea temporal, causas contribuyentes y acciones.
`action_items.csv`	Seguimiento.	Acción, owner, fecha, verificación y estado.
`regression_cases.jsonl`	Vuelta a EvalOps.	Casos derivados de la incidencia.
`replay_case.json`	Reproducción mínima.	Entrada redactada, versiones, esperado y observado.
`oncall_handoff.md`	Cambio de turno.	Estado, hipótesis, mitigaciones y siguiente paso.
`incident_queries.promql`	Consultas listas.	Latencia, contrato, coste y errores por release.

Qué capítulos necesitas tener frescos

La práctica es usable al terminar este capítulo, pero no aparece de la nada. Cada pieza viene de algo trabajado antes:

PARTE DE LA PRÁCTICA	CAPÍTULO CONECTADO	QUÉ RECUPERA
`incident_events.jsonl` con eventos trazables.	F6 · Capítulo 04	Logs, métricas, trazas, `run_id`, `trace_id`, SLI y SLO.
Mitigaciones como bajar canary o volver índice.	F6 · Capítulo 07	Rollback, canary, kill switch y release progresiva.
Casos que vuelven a `regression_cases.jsonl`.	F6 · Capítulo 06	EvalOps, regresiones, gates y scorecard.
Handoff de guardia y revisión humana.	F6 · Capítulo 08	Colas, decisiones, evidence bundle y reanudación.
Routing, fallback y contención.	F6 · Capítulo 05	Rutas, presupuestos, fallback y control de coste.

Si alguien termina el capítulo 09 sin dominar todo lo anterior, aún puede ejecutar el script. Pero para defender la decisión técnica (por qué SEV2, por qué rollback de índice, por qué crear regresión) necesita volver a esas piezas.

Reto de capítulo: analizar una incidencia de IA

Escenario: un canary de prompt y un índice RAG nuevo coinciden con subida de latencia, coste y fallos de contrato. Tu tarea es reconstruir la incidencia desde eventos, calcular tiempos, proponer severidad y generar acciones.

Archivos:

```
mi-proyecto/
  ops/
    ai/
      incident_review.py
  data/
    incident_events.jsonl
  output/
    incident_report.json
```

Datos de entrada en `data/incident_events.jsonl`:

```
{"ts":"2026-05-28T13:40:00+00:00","type":"change","message":"canary prompt_v13 sube al
25%","release_id":"support-rag@1.9.0-rc1","task":"support_reply","impact":0}
{"ts":"2026-05-28T13:43:00+00:00","type":"alert","message":"latency_p95_ms supera
SLO","metric":"latency_p95_ms","value":6200,"threshold":4200,"task":"support_reply","impact"
:3}
{"ts":"2026-05-28T13:47:00+00:00","type":"alert","message":"contract_fail_rate supera
umbral","metric":"contract_fail_rate","value":0.031,"threshold":0.006,"task":"support_reply"
,"impact":4}
{"ts":"2026-05-28T13:49:00+00:00","type":"ack","message":"incidencia reconocida por ai-
platform","actor":"ai-platform-oncall","impact":0}
{"ts":"2026-05-28T13:54:00+00:00","type":"mitigation","message":"candidate_weight baja de 25
a 5","action":"reduce_canary","impact":0}
{"ts":"2026-05-28T14:02:00+00:00","type":"mitigation","message":"rag_index vuelve a
rag_index_2026_05","action":"rollback_index","impact":0}
{"ts":"2026-05-28T14:18:00+00:00","type":"recovery","message":"latency y contrato vuelven a
SLO","metric":"all","value":1,"threshold":1,"impact":0}
{"ts":"2026-05-28T14:25:00+00:00","type":"followup","message":"crear regresión con salida
JSON rota y chunk obsoleto","owner":"evalops","impact":0}
```

Comandos:

```
mkdir -p ops/ai data output
python ops/ai/incident_review.py --write
cat output/incident_report.json
```

Salida esperada:

CAMPO	VALOR ESPERADO
`severity`	`SEV2`
`mtta_minutes`	6
`mttr_minutes`	38
`primary_symptoms`	Latencia p95 y fallos de contrato.
`mitigations`	Bajar canary y volver índice RAG.
`postmortem_required`	`true`

Entrega mínima:

1. `incident_report.json`.

2. Una propuesta de `postmortem.md`.
3. Dos acciones correctivas con owner y verificación.
4. Un caso nuevo para `regression_cases.jsonl`.
5. Una frase que explique qué señal debería haber alertado antes.
6. Un `replay_case.json` con versiones de prompt, modelo, router e índice.
7. Una decisión escrita: rollback, roll forward o contención, con motivo.

Comprobación de que la práctica está bien hecha:

COMPROBACIÓN	RESULTADO ESPERADO
Ejecuta sin dependencias externas.	Solo necesita Python estándar.
Lee `data/incident_events.jsonl`.	Si existe, usa esos eventos; si no existe, usa los eventos de ejemplo.
Escribe salida con `--write`.	Crea `output/incident_report.json`.
Calcula tiempos.	`mtta_minutes = 6` y `mttr_minutes = 38`.
Propone severidad.	`severity = SEV2`.
Conecta con EvalOps.	Genera acciones recomendadas y pide caso de regresión.
Es defendible en clase/equipo.	El alumno puede explicar síntoma, mitigación, replay y acción correctiva.

En el día a día

Las incidencias no son un «si pasa», son un «cuándo pasa», y la diferencia entre un equipo que aprende y uno que tropieza dos veces con la misma piedra está en lo que hace después. En el momento caliente, lo operable es tener una clasificación de severidad (no es lo mismo un SEV1 que tira el servicio que un SEV3 cosmético), un runbook que diga los primeros pasos sin tener que pensarlos a las tres de la mañana, y una decisión clara sobre si se mitiga, se hace rollback o se escala. Improvisar bajo presión es como casi todo el mundo gestiona su primera incidencia grave, y casi nadie lo recuerda con orgullo.

Después viene la parte que de verdad mejora el sistema: el postmortem. La escena sana no busca un culpable, busca la cadena de causas (qué falló, qué lo permitió, qué lo detectó tarde) y sale con acciones concretas y con dueño: un test de regresión nuevo, una alerta que faltaba, un límite que no estaba. La escena tóxica busca a quién señalar, y garantiza que el mismo fallo vuelva, porque nadie se atreve a contar lo que de verdad pasó.

Por qué debería importarte

Porque en operación no se trata de no fallar nunca (eso no existe en sistemas no deterministas conectados a proveedores que también fallan), sino de fallar cada vez de una forma nueva, porque las viejas ya se cerraron con una acción concreta. La gestión de incidencias y el postmortem sin culpa son la maquinaria que convierte cada fallo en una mejora permanente del sistema, en vez de en una herida que se reabre.

Y hay una razón muy personal para tomárselo en serio. La forma en que un equipo trata sus incidencias determina si la gente cuenta la verdad cuando algo sale mal o la esconde. Un postmortem sin culpa, con severidades claras y acciones con dueño, construye lo único que de verdad hace fiable a un sistema operado por personas: la confianza para decir «esto lo rompí yo, y así evitamos que vuelva a pasar».

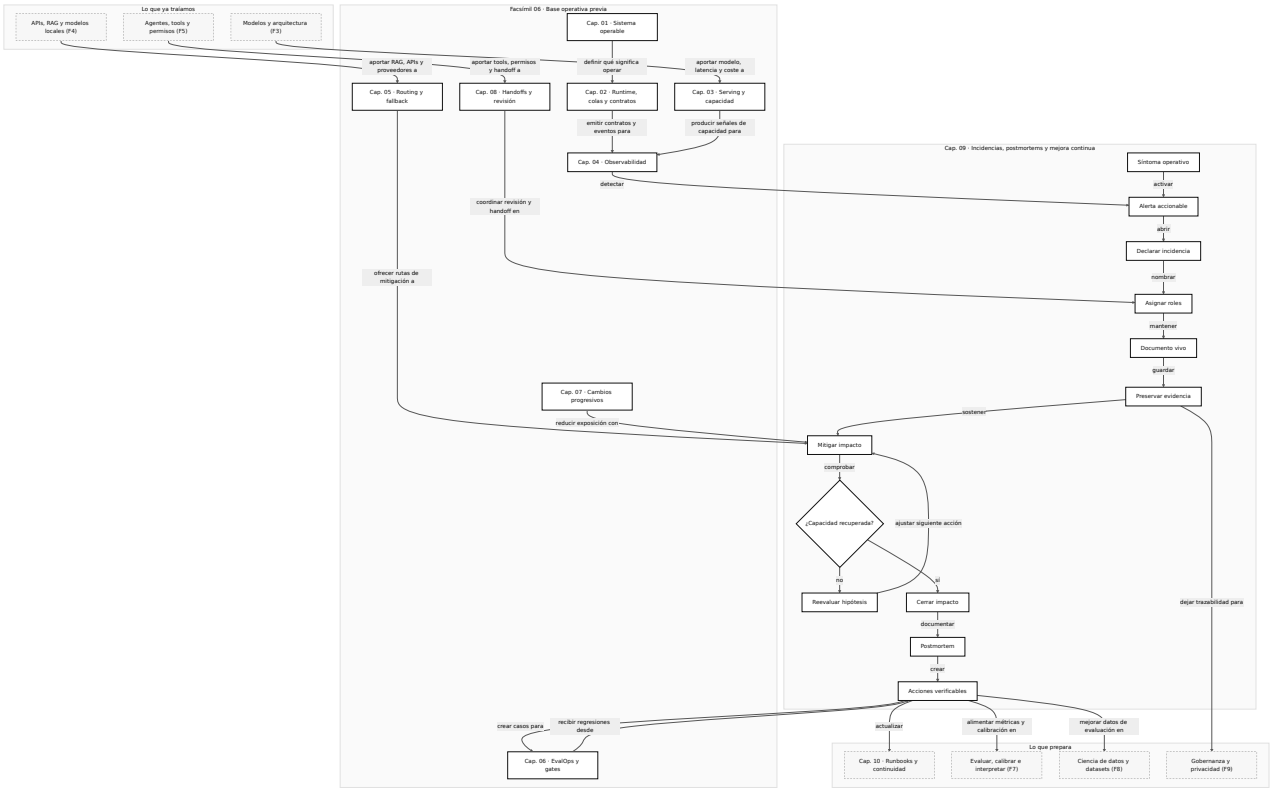
Dónde solía tropezar yo

Me costó aceptar que una incidencia bien operada puede parecer lenta al principio: parar, nombrar roles y escribir estado parece burocracia. Luego descubres que evita cambios cruzados, duplicidad de mensajes y pérdida de evidencia.

TROPIEZO	ANTÍDOTO
Investigar sin declarar.	Si afecta SLO o requiere otro equipo, declara pronto.
Cambiar varias cosas a la vez.	Una mitigación, un owner, un registro.
Cerrar al recuperar.	Cerrar solo cuando hay postmortem o decisión de no hacerlo.
Acciones vagas.	Cada acción con owner, fecha y verificación.
No alimentar EvalOps.	Cada incidencia debe crear o revisar casos de regresión.

La frase útil: **la recuperación devuelve el servicio; el postmortem mejora el sistema**.

Cómo encaja todo



Este mapa no intenta repetir toda la incidencia paso a paso. Su función es enseñar dónde vive el capítulo 09 dentro del facsímil: nace de observabilidad, routing, EvalOps, cambios progresivos y handoffs; convierte una degradación en evidencia y acciones; y deja preparado el capítulo 10, donde esas acciones se vuelven runbooks, continuidad y laboratorio de operación.

Relación con otros capítulos

CAPÍTULO	QUÉ APORTA AQUÍ
F6 · Capítulo 04	Señales, SLI, SLO, alertas y runbooks.
F6 · Capítulo 05	Fallback, presupuestos, rate limits y degradación controlada.
F6 · Capítulo 06	Casos de incidencia que vuelven a datasets y gates.
F6 · Capítulo 07	Rollback, kill switch y post-release review.
F6 · Capítulo 08	Handoffs, colas, SLO de revisión y decisiones humanas.

Amershi y colaboradores mostraron que los sistemas de ML en producción requieren procesos de ingeniería alrededor de datos, evaluación, monitorización y operación.[15] El NIST AI RMF organiza el trabajo alrededor de gobernar, mapear, medir y gestionar sistemas de IA.[16] Una incidencia bien cerrada toca los cuatro verbos: alguien gobierna, el equipo mapea impacto, mide señales y gestiona cambios.

Para entenderlo

Tres escenas:

SITUACIÓN	RESPUESTA DÉBIL	RESPUESTA OPERATIVA
Sube latencia p95 tras canary.	“Estamos mirando logs”.	SEV2, bajar canary, guardar trazas y abrir postmortem.
RAG cita documentos obsoletos.	Cambiar el índice sin dejar rastro.	Rollback de índice, conservar queries y crear regresión.
Cola de revisión caduca.	Pedir paciencia a soporte.	Activar backpressure y revisar política de handoff.

La madurez no está en no tener incidencias. Está en que cada una reduzca la próxima.

Vocabulario aprendido

TÉRMINO	DEFINICIÓN BREVE
Incidencia	Evento que degrada una propiedad operativa relevante.
Severidad	Clasificación por impacto, alcance, duración y urgencia.
MTTA	Tiempo hasta reconocer una incidencia.
MTTR	Tiempo hasta recuperar el servicio o capacidad.
Comando de incidencia	Rol que mantiene estado global y coordina respuesta.
Mitigación	Acción para reducir impacto antes de corregir causa profunda.
Postmortem	Documento de aprendizaje y acciones verificables.
Acción correctiva	Trabajo con owner y verificación que reduce recurrencia o impacto.

Antes de pasar página

Comprueba que puedes responder:

1. ¿Qué diferencia hay entre alerta, incidencia y postmortem?
2. ¿Qué dimensiones usarías para severidad en un sistema de IA?
3. ¿Qué roles separarías durante una incidencia?
4. ¿Por qué mitigar no siempre corrige la causa profunda?
5. ¿Qué datos necesitas para calcular MTTA y MTTR?
6. ¿Qué acción correctiva considerarías verificable?
7. ¿Cómo convertirías una incidencia en un caso de EvalOps?

En resumen

IDEA	PARA LLEVARTE
Una incidencia de IA puede ser calidad, contrato, coste, latencia o trazabilidad.	No mires solo caídas de servicio.
La respuesta necesita roles.	Comando, operaciones, comunicación y planificación reducen caos.
Mitigar protege, postmortem aprende.	Las dos cosas son necesarias y distintas.
Una acción sin verificación no cambia el sistema.	Owner, fecha, prueba y enlace al caso de regresión.

Para saber más

- Amershi, S. y otros *Software Engineering for Machine Learning: A Case Study*. https://doi.org/10.1109/ICSE-SEIP.2019.00042
- Baye, C. A. *Emergency Response*. https://sre.google/sre-book/emergency-response/
- Beyer, B. y otros *Handling Overload*. https://sre.google/sre-book/handling-overload/
- Dean, J. y Barroso, L. A. *The Tail at Scale*. https://doi.org/10.1145/2408776.2408794
- Ewaschuk, R. *Monitoring Distributed Systems*. https://sre.google/sre-book/monitoring-distributed-systems/
- Little, J. D. C. *A Proof for the Queuing Formula: $L = \lambda W$*. https://doi.org/10.1287/opre.9.3.383
- Lunney, J. y Lueder, S. *Postmortem Culture: Learning from Failure*. https://sre.google/sre-book/postmortem-culture/
- OpenTelemetry. *Logs*. https://opentelemetry.io/docs/concepts/signals/logs/
- OpenTelemetry. *Metrics*. https://opentelemetry.io/docs/concepts/signals/metrics/
- OpenTelemetry. *Traces*. https://opentelemetry.io/docs/concepts/signals/traces/
- Stribblehill, A. *Managing Incidents*. https://sre.google/sre-book/managing-incidents/
- Wilkinson, J. *Alerting on SLOs*. https://sre.google/workbook/alerting-on-slos/

Notas

1. Ewaschuk, R. (2016). *Monitoring Distributed Systems*. En B. Beyer, C. Jones, J. Petoff y N. R. Murphy (eds.), *Site Reliability Engineering*. https://sre.google/sre-book/monitoring-distributed-systems/. Consultado el 27 de mayo de 2026. ↩

2. Wilkinson, J. (2018). *Alerting on SLOs*. En B. Beyer, N. R. Murphy, D. Rensin, K. Kawahara y S. Thorne (eds.), *The Site Reliability Workbook*. https://sre.google/workbook/alerting-on-slos/. Consultado el 27 de mayo de 2026. ↩

3. Baye, C. A. (2016). *Emergency Response*. En *Site Reliability Engineering*. https://sre.google/sre-book/emergency-response/. Consultado el 28 de mayo de 2026. ↩

4. Stribblehill, A. (2016). *Managing Incidents*. En *Site Reliability Engineering*. https://sre.google/sre-book/managing-incidents/. Consultado el 28 de mayo de 2026. ↩

5. Lunney, J. y Lueder, S. (2016). *Postmortem Culture: Learning from Failure*. En *Site Reliability Engineering*. https://sre.google/sre-book/postmortem-culture/. Consultado el 28 de mayo de 2026. ↩

6. Keeney, R. L. y Raiffa, H. (1976). *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley. Formaliza la función de valor aditiva ponderada para decidir entre alternativas. ↩

7. Little, J. D. C. (1961). A proof for the queuing formula: L = λW. *Operations Research*, 9(3), 383-387. https://doi.org/10.1287/opre.9.3.383 ↩

8. Trivedi, K. S. (2016). *Probability and Statistics with Reliability, Queuing, and Computer Science Applications* (2.ª ed.). Wiley. Para una tasa de fallo constante, el tiempo hasta el fallo es exponencial y la fiabilidad decae como $e^{-\lambda t}$. ↩

9. Pareto, V. (1896). *Cours d'économie politique*. La distribución de Pareto, $P(X > x) = (x_m/x)^{\alpha}$, modela fenómenos donde una minoría concentra la mayor parte del efecto; su aplicación a la calidad la popularizó J. M. Juran. ↩

10. Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate: The Science of Lean Software and DevOps*. IT Revolution. Define y valida empíricamente las cuatro métricas clave de rendimiento de entrega, entre ellas la tasa de fallo de cambios y el tiempo de recuperación. ↩

11. Crow, L. H. (1975). *Reliability Analysis for Complex, Repairable Systems* (AMSAA Technical Report 138). U.S. Army Materiel Systems Analysis Activity. Modelo de crecimiento de fiabilidad de Crow-AMSAA basado en un proceso de potencia. ↩

12. Dean, J. y Barroso, L. A. (2013). The Tail at Scale. *Communications of the ACM*, 56(2), 74-80. https://doi.org/10.1145/2408776.2408794 ↩

13. Stribblehill, 2016. ↩

4. Beyer, B., Jones, C., Petoff, J. y Murphy, N. R. (2016). *Handling Overload*. En *Site Reliability Engineering*. https://sre.google/sre-book/handling-overload/. Consultado el 27 de mayo de 2026. ↩

5. Amershi, S. y otros (2019). *Software Engineering for Machine Learning: A Case Study*. Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, 291-300. https://doi.org/10.1109/ICSE-SEIP.2019.00042 ↩

6. Tabassi, E. (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. NIST AI 100-1. https://doi.org/10.6028/NIST.AI.100-1 ↩