# El vocabulario del canario: testing, infraestructura y por qué se confunden

## Requisitos

- Contexto de lectura: nociones básicas de tests automáticos (para las secciones de código) y de infraestructura (para las de despliegue). El documento introduce su propio vocabulario desde cero.
- Para aplicar el criterio del cuerpo: un suite de tests o una verificación de infraestructura propia donde distinguir canario de test negativo.

Documento completo a continuación:

## Introducción

Nota de alcance: este documento no es sobre pájaros. Es sobre un patrón de detección de fallo silencioso que aparece, con matices distintos, en código y en infraestructura — y sobre por qué llamar "canario" a cosas distintas genera confusión operativa real.

---

## 1. El origen: por qué la mina importa

Los mineros llevaban un canario porque el gas grisú (metano, monóxido de carbono) es indetectable para el ser humano hasta que ya es tarde. El canario, con metabolismo más sensible, dejaba de cantar — o moría — **antes** de que la concentración fuera letal para las personas. Su valor no estaba en sobrevivir; estaba en fallar de forma temprana, específica y observable ante

una amenaza concreta que de otro modo se acumula sin ruido.

Esa es la propiedad que toda la terminología posterior intenta capturar: **detectar la ausencia silenciosa de una protección, no el fallo ruidoso de una función.**

Guárdate esa frase. Es el criterio que separa un canario de verdad de algo que solo se le parece.

---

# 2. Canario de test (código)

## Contrato

Un canario de test no verifica que el sistema *funcione*. Verifica que una **protección específica sigue activa**. Estructura:

1. Construyes deliberadamente el caso prohibido (usuario sin permiso, input inválido, estado no autorizado).
2. Afirmas que un guard concreto lo rechaza.
3. La aserción identifica **cuál** guard actuó, no solo que "algo" rechazó.

## Canario ≠ test negativo

Se solapan pero no son lo mismo. La diferencia está en la intención y, por tanto, en qué necesita afirmar el test:

| | Test negativo (genérico) | Canario |
|---|---|---|
| Qué protege | El contrato de comportamiento: "ante input inválido, se rechaza" | La existencia de un mecanismo concreto: "el guard G sigue siendo quien rechaza" |
| Aserción típica | "La petición fue rechazada" | "La petición fue rechazada **por G**, identificable por una firma propia" |
| Qué pasa si `G` desaparece pero otro mecanismo cubre el mismo caso | Sigue en verde — y con razón, el contrato de comportamiento se cumple | Debería fallar, porque `G` ya no existe. Si sigue en verde, el canario está roto |

Todo canario es un test negativo. No todo test negativo es un canario. La confusión de categorías es la fuente de la mayoría de canarios que fallan en su propósito sin que nadie se entere.

---

# 3. El fallo silencioso: co-extensividad y tautología

**Co-extensivo** = dos guards cubren exactamente el mismo conjunto de casos. Todo lo que rechaza A también lo rechaza B.

## El problema

Si escribes un canario para vigilar al guard A, pero B es co-extensivo con A en la ruta que pruebas, tu aserción ("la petición fue rechazada") **no puede distinguir cuál de los dos actuó**. Puedes eliminar A por completo — un refactor, una regresión, un merge descuidado — y el canario sigue en verde porque B sigue atrapando el mismo caso.

El test se ha vuelto **tautológico**: ya no puede fallar por la causa que dice vigilar. En términos de mutation testing: un test solo vale lo que valen los mutantes que mata. Si "matas" (eliminas) el guard A y el test no muere, el test no estaba probando A — estaba probando la propiedad más débil de "algo en esta ruta rechaza esto".

## Ejemplo genérico

```
Ruta protegida por:
  Guard A (autorización de dominio) → lanza ForbiddenException → 403
  Guard B (validación de framework, más genérica)   → lanza ValidationException → 403 también
```

Ambos devuelven 403. El canario que solo afirma `response.status === 403` es tautológico: prueba la ruta, no prueba A.

## Cómo se rompe la tautología: el discriminador

Un canario válido necesita un **discriminador observable** entre A y B. Puede ser:

- Tipo de excepción concreto (afirmar la clase, no solo el status code).
- Código de estado *distinto* entre los dos guards.
- Mensaje o payload de error con firma propia.
- Entrada de log/auditoría específica del mecanismo.

# Caso real, generalizado: el defecto que sin querer arregló el canario

En un caso concreto, una excepción de un paquete de terceros (`AuthorizationException` del framework) no renderizaba el 403 canónico, sino un 500 — un defecto real, no un diseño intencional. El guard propio del dominio sí devolvía 403.

Efecto lateral: ese defecto **rompió la co-extensividad observable** entre ambos guards. El canario pudo afirmar el código de estado exacto y así saber con certeza cuál guard había actuado. El bug ajeno funcionó, sin buscarlo, como discriminador.

**Advertencia importante si vas a documentar un caso así:** esta validez es frágil, no robusta. Depende de un comportamiento *no contractual* de una dependencia externa. El día que el paquete corrija su excepción para devolver el 403 correcto, los dos guards vuelven a ser observablemente co-extensivos — y el canario vuelve a ser tautológico **en silencio**, sin que ningún test rompa para avisarte. La lección correcta no es "qué suerte que el bug nos ayudó", es "hay que sustituir ese discriminador accidental por uno propio antes de que alguien arregle el bug ajeno".

---

# 4. Canario en infraestructura: tres animales bajo el mismo nombre

Aquí es donde la terminología SysOp/DevOps se vuelve resbaladiza, porque "canario" se usa para tres cosas distintas y solo una de ellas hereda de verdad la semántica minera.

## 4.1 Sandbox desechable — *esto normalmente NO es un canario minero*

Levantas una VM/contenedor efímero, aplicas ahí un cambio de infraestructura (playbook, regla de firewall, imagen, migración) antes de tocar el entorno real, y compruebas que "funciona" o que "no rompe nada evidente".

Es un **ensayo con doble de riesgo**: una copia sacrificable donde fallar es barato. Tiene valor real, pero su lógica es la de un test negativo genérico como mucho ("¿el cambio rompe algo detectable?"), no la del canario. No vigila la desaparición silenciosa de un mecanismo concreto; solo da una oportunidad de fallar en un sitio que no duele.

Llamarlo "canario" no es grave, pero no aporta el rigor que el nombre promete.

# 4.2 Canary deployment / canary release — el uso correcto de "deployment canary"

Una instancia real de producción recibe una fracción del tráfico real antes que el resto de la flota. Si sus métricas se degradan, se frena el rollout.

Aquí sí hay vigilancia activa con consecuencia automática, pero el objeto vigilado es la **salud general del sistema** ante un cambio, no "¿sigue vivo el guard X?". Es un patrón legítimo y bien nombrado — pero resuelve un problema distinto al canario de test.

# 4.3 El canario minero real en infraestructura

Para que una prueba de infra tenga la misma estructura que el canario de código, necesita:

1. Provisionarse deliberadamente para **intentar la acción prohibida** contra un mecanismo de defensa concreto (AppArmor/SELinux profile, política de firewall, IAM policy, seccomp, permisos POSIX).
2. Afirmar que **ese mecanismo específico** — identificable por su firma propia — es quien la bloquea. No basta con "la operación falló".

**Ejemplo:** una política de AppArmor impide que un proceso escriba fuera de `/var/lib/miapp`. El canario correcto no levanta la VM y comprueba que "todo funciona bien" — intenta deliberadamente escribir fuera de esa ruta y afirma que **AppArmor**, y no otro mecanismo, generó el rechazo, verificando la entrada de auditoría específica de AppArmor (`AVC denied` en syslog, por ejemplo), no un `Permission denied` genérico que podría venir de cualquier sitio.

# La misma trampa reaparece en infra

Si además del AppArmor profile hay un contenedor con seccomp restrictivo o un usuario sin permisos POSIX suficientes que *también* bloquearía esa escritura, un canario que solo comprueba "la escritura falló" es tautológico exactamente por la misma razón que en código: no distingues si AppArmor sigue activo o si alguien lo desactivó y el rechazo viene de otra capa. El grisú se acumula sin que el pájaro cante, porque otro mecanismo tapa el hueco por casualidad.

# Tabla comparativa

| | Sandbox desechable | Canary deployment | Canario minero (infra) |
|---|---|---|---|
| Qué es | VM/contenedor de usar y tirar | Instancia real con tráfico parcial | Prueba deliberada de la acción prohibida |
| Qué vigila | Nada en particular — solo abarata el fallo | Salud general del sistema | Un guard concreto y su firma de rechazo |
| ¿Hereda la semántica minera? | No | Parcialmente | Sí, si tiene discriminador |
| Riesgo típico | Ninguno grave, solo falsa sensación de rigor | Detecta degradación, no ausencia de guard concreto | Tautológico si hay guards co-extensivos sin discriminador |

---

# 5. El test decisivo

Ante cualquier cosa etiquetada "canario" — código o infra — una sola pregunta la diagnostica:

> **¿Este test falla específicamente porque el guard G desapareció, o fallaría igual con G ausente por cualquier otra razón coincidente?**

- Si la respuesta es "falla igual, no lo sé" → es un test negativo o un sandbox con nombre bonito. Útil, pero no un canario.
- Si la respuesta es "falla *solo* si G desaparece, porque afirmo su firma propia" → es un canario de verdad.

---

# 6. Por qué esto pesa más cuando eres SysOp + DevOps a la vez

Cuando una sola persona diseña guards en varias capas — autorización de aplicación, políticas del SO, reglas de red, IAM de infraestructura — la probabilidad de introducir **co-extensividad accidental entre capas** sube, no baja. Es fácil que un guard de aplicación y una política de sistema operativo terminen cubriendo el mismo caso sin que nadie lo diseñara así a propósito; cada capa se construyó pensando solo en la suya.

La disciplina que compensa esto no es "confiar menos en los guards", es **exigir discriminador propio en cada canario que cruce capas**: si un canario de infra y uno de aplicación podrían, en teoría, disparar por el mismo caso prohibido, cada uno debe afirmar su firma específica — no un

resultado genérico que ambos podrían producir.

---

# 7. Resumen operativo

- **Test negativo**: prueba que un input/estado inválido se rechaza. Contrato de comportamiento.
- **Canario**: prueba que un guard *concreto* sigue vivo. Contrato de existencia de mecanismo. Necesita discriminador observable o es tautológico.
- **Sandbox desechable**: ensayo barato antes de tocar producción. No es un canario, aunque se llame así.
- **Canary deployment**: fracción de tráfico real como detector de salud general. Legítimo, pero resuelve otro problema.
- **Canario minero en infra**: intento deliberado de la acción prohibida contra un mecanismo específico, con aserción de su firma propia. El único de los cuatro que hereda sin trampa la metáfora original.

Regla mnemotécnica: *el canario no muere de cualquier cosa — muere de una cosa concreta, y si no puedes decir de cuál, no es un canario, es ruido de fondo con buena reputación.*

## Aviso

Esta documentación y su contenido, no implica que funcione en tu caso o determinados casos. También implica que tienes conocimientos sobre lo que trata, y que en cualquier caso tienes copias de seguridad. El contenido el contenido se entrega, tal y como está, sin que ello implique ningún obligación ni responsabilidad por parte de Castris

Si necesitas soporte profesional puedes contratar con Castris soporte profesional.

# Verificación

- La verificación de un canario es la de su capacidad de fallar (la que el cuerpo llama la prueba de mutación): eliminar o desactivar deliberadamente el mecanismo vigilado (el guard, el profile, la política) debe poner el canario en rojo. Si sigue en verde, el canario es tautológico y hay que darle un discriminador propio.
- Control cruzado de la advertencia del cuerpo: si el discriminador actual depende de un comportamiento no contractual de una dependencia (el caso del bug ajeno), programar la sustitución por un discriminador propio y verificar la alarma al sustituirlo.
- Para las variantes de infraestructura: la prueba 4.3 del cuerpo (intentar la acción prohibida y afirmar la firma específica del mecanismo, p. ej. el `AVC denied` de AppArmor) es el patrón a reproducir; "la operación falló" sin firma no verifica nada.

---

Revision #2
Created 2026-07-09 03:26:26 UTC by Abkrim
Updated 2026-11-13 17:16:20 UTC by Abkrim