Multi-Cloud Engineering Program

Parte 17 — AWS: arquitectura, automatización y operación en producción

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	17 de 23
Clases	205–216
Evaluaciones	12
Horas estimadas	52

Índice

Parte 17 — AWS: arquitectura, automatización y operación en producción

← Parte 16 · Índice completo · Parte 18 →

Descargar: Esta parte en PDF · Recorrido de AWS en PDF · Manual integral

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

Integrar los casos AWS existentes en una ruta de producción con seguridad, costo, evidencia y destrucción.

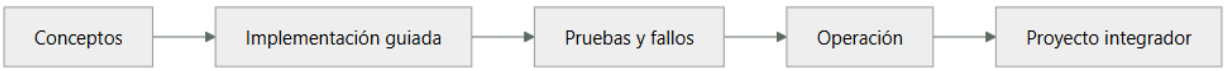
Resultados de la parte

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

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

Secuencia



Clases

ID	Clase	Laboratorio	Horas
205	Hosting progresivo con Amplify, S3 y CloudFront	delivery	4
206	OIDC de GitHub y GitLab hacia AWS sin secretos	iam	4
207	SAM, Lambda, API Gateway y despliegue serverless	serverless	4
208	DynamoDB por patrones de acceso y single-table design	data	4
209	Cognito, JWT authorizers, WAF y defensa en profundidad	security	4
210	EventBridge, SQS, DLQ, replay e idempotencia	messaging	4
211	CloudWatch, X-Ray y observabilidad como código	observability	4
212	ECR, ECS Fargate, ALB y autoscaling	container	4
213	EKS, IRSA, GitOps y operación de clúster	kubernetes	4
214	Budgets, Cost Explorer, etiquetado y FinOps automatizado	finops	4
215	Multi-región, Route 53, failover y game day	reliability	4
216	Proyecto: CloudShop productivo en AWS	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

- AWS Well-Architected Framework — AWS.
- AWS Cookbook — Culkin y Zazon.
- Serverless Architectures on AWS — Peter Sbarski.
- Cloud FinOps — Storment y Fuller.

205 — Hosting progresivo con Amplify, S3 y CloudFront

Parte: 17 — AWS: arquitectura, automatización y operación en producción Nivel: avanzado · Horas estimadas: 4
Laboratorio: delivery · Estado: EXECUTABLECORE

Propósito

Publicar una aplicación web en AWS eligiendo entre el camino gestionado y el montado a mano, y entender qué se gana y qué se pierde con cada uno. La clase compara Amplify Hosting con el montaje S3 + CloudFront pieza a pieza, fija los valores por defecto que hay que cambiar para producción —que son más de los que parece—, y trata los tres asuntos que rompen estos despliegues: el enrutado de aplicaciones de una sola página, la invalidación de caché en cada publicación y el acceso al bucket.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir entre alojamiento gestionado y montaje propio con criterios claros.
2. Configurar S3 y CloudFront con los valores correctos para producción.
3. Resolver el enrutado de una aplicación de una sola página sin romper los 404 reales.
4. Publicar sin invalidaciones masivas, usando huellas en los nombres.
5. Cerrar el acceso al origen para que solo la distribución pueda leerlo.

Conceptos centrales

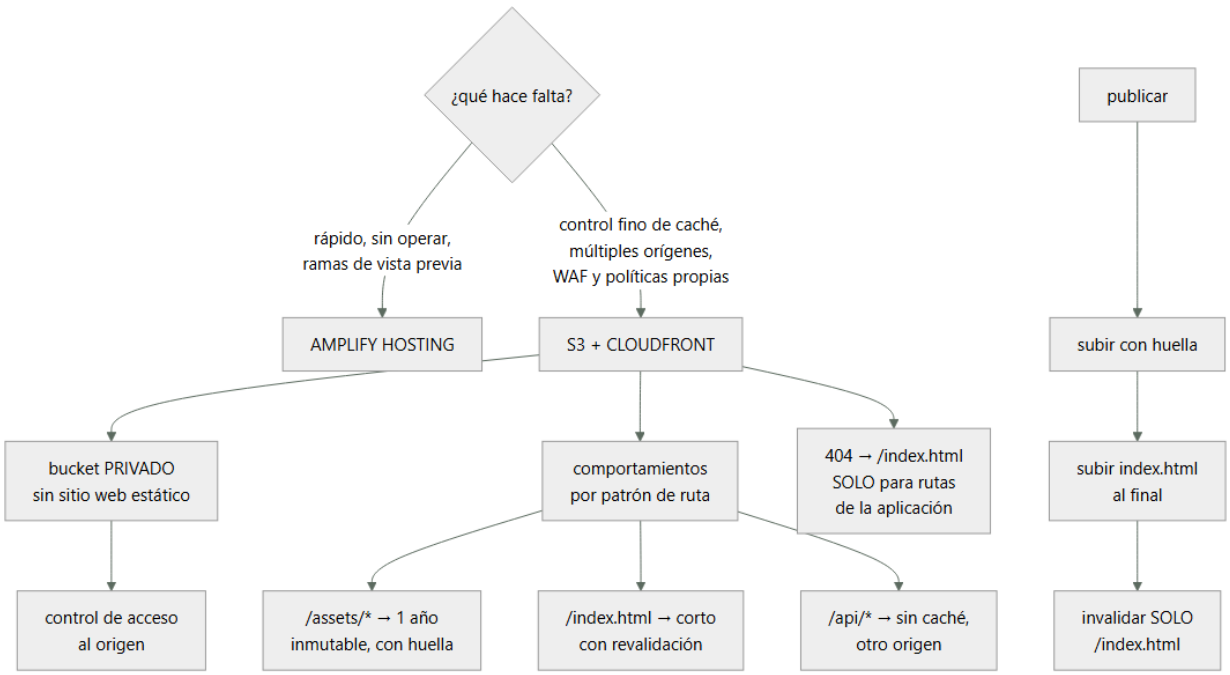
Concepto	Comprensión verificable
Amplify Hosting	Servicio gestionado que construye y publica una aplicación web desde un repositorio, con distribución incluida.
distribución	Recurso de CloudFront que sirve el contenido desde el borde y define caché, orígenes y comportamiento.
control de acceso al origen	Mecanismo por el que solo la distribución puede leer el bucket, que queda privado.
comportamiento de caché	Regla por patrón de ruta que fija clave de caché, validez y política de origen.
función de borde	Código ligero que se ejecuta en el borde para reescribir peticiones o añadir cabeceras.
huella en el nombre	Sufijo derivado del contenido que hace innecesaria la invalidación al publicar.

Modelo mental

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

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. Gestionado o montado a mano

Las dos opciones sirven para lo mismo y se diferencian en control y en operación.

```
AMPLIFY HOSTING
  construye desde el repositorio y publica
  da   ramas de vista previa por cada propuesta de cambio
       certificado y dominio gestionados
       reversión a una publicación anterior
       reescrituras y redirecciones por configuración
       distribución por debajo, sin gestionarla
  no da
       control fino de la clave de caché por ruta
       comportamientos con orígenes distintos
       políticas de origen propias
       WAF con reglas propias en muchos casos
       acceso a los registros de la distribución

S3 + CLOUDFRONT MONTADO
  da   control total: comportamientos, claves, políticas,
       WAF, registros, varios orígenes
  cuesta
       montarlo y mantenerlo
       y aprender los valores por defecto que hay que cambiar
```

Y el criterio:

```
¿el sitio es estático o casi, y no hay requisitos de caché
 ni de seguridad especiales?
  → Amplify; se monta en una tarde y se opera solo

¿hay API en el mismo dominio, varias rutas con caché
 distinta, WAF propio o necesidad de registros?
  → S3 + CloudFront

¿el equipo ya opera CloudFront para otras cosas?
  → montarlo, por coherencia
```

Y una vía intermedia que funciona bien:

```
Amplify para las ramas de vista previa (donde su valor es
mayor y los requisitos son bajos)
S3 + CloudFront para producción
→ y así se tiene rapidez donde importa la velocidad y
```

```
  control donde importa el control
```

Y la advertencia de coste que aparece con Amplify:

```
factura por minuto de construcción y por gigabyte servido
→ con muchas ramas de vista previa y construcciones
  frecuentes, la partida de construcción sube deprisa
→ y las vistas previa no caducan solas                ley 25
```

2. Los valores por defecto que hay que cambiar

Montar S3 + CloudFront con los valores por defecto produce algo que funciona y no es apto para producción. Esta es la lista.

```
EL BUCKET
  ✗ por defecto   se tiende a activar «alojamiento de sitio
                  web estático» y hacerlo público
  ✓ correcto      bucket PRIVADO, sin alojamiento estático,
                  accesible solo por control de acceso al
                  origen desde la distribución
  por qué         un bucket público es alcanzable saltándose
                  la distribución: sin WAF, sin registros,
                  sin caché y pagando salida  clases 189, 200

  ✓ además        bloqueo de acceso público activado
  ✓               cifrado en reposo
  ✓               versionado, para poder revertir una
                  publicación

LA DISTRIBUCIÓN
  ✗ por defecto   política de caché heredada que reenvía
                  cabeceras y cookies innecesarias
  ✓ correcto      política que NO incluye cookies ni
                  cabeceras salvo las que varían   clase 197

  ✗ por defecto   un solo comportamiento para todo
  ✓ correcto      comportamientos por patrón de ruta

  ✗ por defecto   HTTP permitido
  ✓ correcto      redirección a HTTPS y versión mínima de TLS

  ✗ por defecto   sin cabeceras de seguridad
  ✓ correcto      política de cabeceras de respuesta

  ✗ por defecto   sin compresión negociada en algunos casos
  ✓ correcto      compresión activada

  ✗ por defecto   registros desactivados
  ✓ correcto      registros a un bucket propio, con caducidad
```

Y los comportamientos que hay que definir, con sus valores:

```
/assets/*   o /_next/static/*  (recursos con huella)
  validez           1 año, inmutable
  clave de caché    solo la ruta
  → nunca se invalidan: cambian de nombre       clase 197

/index.html  y las rutas de la aplicación
  validez           0 o muy corta, con revalidación
  → es el único fichero que hay que invalidar al publicar

/api/*
  otro origen (API Gateway o balanceador)
  sin caché, reenviando las cabeceras necesarias
  → tener la API en el mismo dominio evita CORS y cookies
    de terceros

/media/*  (imágenes subidas)
  validez media, con nombre versionado si se pueden
  reemplazar
```

Y una decisión de arquitectura que ahorra problemas:

```
servir la aplicación y la API bajo el MISMO dominio, con
comportamientos distintos
```

```
→ sin CORS, sin cookies de terceros y con una sola
  distribución que registrar y proteger
```

3. Las tres cosas que rompen

1 · El enrutado de una aplicación de una sola página.

```
EL PROBLEMA
  la aplicación tiene rutas como /pedidos/4471
  ese objeto NO existe en el bucket
  → S3 devuelve 404 y el usuario ve un error al recargar o
    al entrar por un enlace directo

LA SOLUCIÓN HABITUAL, Y SU DEFECTO
  configurar «404 → /index.html con código 200»
  funciona… y convierte TODOS los 404 en 200
  → incluidos los de recursos que faltan de verdad
  → los buscadores indexan páginas inexistentes
  → y los errores de despliegue se vuelven invisibles
                                                  ley 13

LA SOLUCIÓN CORRECTA
  una función de borde que reescribe la petición a
  /index.html SOLO si la ruta no tiene extensión de fichero
  → /pedidos/4471   → index.html, 200
  → /assets/x.js    → 404 real si no existe
  → y para rutas conocidas de la aplicación, mejor aún:
    lista explícita
```

2 · La invalidación en cada publicación.

```
EL ANTIPATRÓN
  publicar y ejecutar «invalidar /*»
  → tira toda la caché en cada despliegue
  → avalancha contra el origen                  clase 197
  → y por encima de cierto número, las invalidaciones se
    facturan

EL PATRÓN CORRECTO
  1  construir con huella en el nombre de cada recurso
     app.7f3a91.js, estilos.2b91cd.css
  2  subir los recursos nuevos PRIMERO
  3  subir index.html AL FINAL
     → así nunca hay un index que apunte a algo que aún no
       está
  4  invalidar SOLO /index.html (y las rutas HTML)
  5  no borrar los recursos antiguos de inmediato
     → los usuarios con la página abierta siguen pidiéndolos
     → borrarlos con una regla de caducidad de 30 días
```

Y el fallo clásico que evita el paso 5:

```
publicar y borrar lo antiguo a la vez
→ quien tenía la aplicación abierta pide un recurso que ya
  no existe y la página se rompe hasta recargar
```

3 · El acceso al bucket.

```
CON CONTROL DE ACCESO AL ORIGEN
  la distribución firma sus peticiones al bucket
  la política del bucket permite SOLO a esa distribución
  el bucket no tiene acceso público
  → y el contenido solo se puede obtener por la
    distribución, con su WAF, sus registros y su caché

LO QUE HAY QUE COMPROBAR, NO SUPONER
  intentar leer un objeto directamente por la URL del bucket
  → debe fallar                                    ley 22
```

Y el fallo de configuración que deja el hueco:

```
dejar activado el alojamiento de sitio web estático del
bucket «por si acaso»
→ expone un punto de entrada alternativo, público y sin
  control                                      clase 189
```

4. Publicar y operar

La canalización de publicación, que debe hacer lo mismo siempre:

```
1  construir, con huella en los nombres
2  autenticarse sin secretos                        clase 206
3  subir recursos nuevos
4  subir HTML
5  invalidar solo HTML
6  comprobar que el sitio responde y sirve la versión nueva
7  y si no, revertir
```

Y la reversión, que hay que tener resuelta antes de necesitarla:

```
con versionado del bucket y huellas
  revertir = volver a subir el index.html anterior e
  invalidarlo
  → los recursos antiguos siguen ahí (por eso no se borran)
  → tiempo de reversión: segundos

sin huellas ni versionado
  → reconstruir y volver a publicar: minutos u horas
```

Lo que hay que vigilar:

```
tasa de aciertos por comportamiento, no global  clase 197
errores 4xx y 5xx desde el origen
bytes servidos desde el borde y desde el origen
latencia en el borde, por región
certificado: días hasta caducar y días desde la renovación
                                                clase 196
y el coste: peticiones, transferencia e invalidaciones
```

Y dos alertas que evitan sorpresas:

```
«la tasa de aciertos ha caído más de 10 puntos»
  → suele significar que alguien cambió la clave de caché
    o que se está invalidando de más

«hay peticiones directas al bucket»
  → si el control de acceso al origen está bien, deben ser
    cero; cualquier cosa distinta es un hueco
```

Y una nota sobre el dominio y el certificado:

```
el certificado de una distribución de CloudFront debe estar
en la región us-east-1, independientemente de dónde esté
todo lo demás
→ es una de esas particularidades que cuestan una tarde la
  primera vez
```

Y la lista de comprobación de la clase:

- ■ la elección entre gestionado y montado está justificada
- ■ el bucket es privado y sin alojamiento estático activado
- ■ el acceso solo es posible por la distribución, comprobado
- ■ hay bloqueo de acceso público y versionado
- ■ hay comportamientos por patrón de ruta
- ■ la clave de caché no incluye cookies ni cabeceras innecesarias
- ■ los recursos llevan huella y validez de un año
- ■ el HTML tiene validez corta
- ■ el enrutado de la aplicación no convierte todos los 404 en 200
- ■ la publicación sube HTML al final e invalida solo HTML
- ■ los recursos antiguos caducan a 30 días, no se borran
- ■ hay redirección a HTTPS y cabeceras de seguridad
- ■ los registros están activados con caducidad
- ■ la reversión está probada y tarda segundos
- ■ hay alerta si aparecen peticiones directas al bucket

Y el cierre que enlaza con la clase siguiente: la canalización que publica necesita permisos en AWS, y el modo habitual de dárselos —una clave de acceso guardada como secreto— es exactamente lo que este programa lleva desaconsejando desde la parte 11. Federación desde el repositorio, sin secretos, es la materia de la clase 206.

Ejemplo trabajado

CloudShop publica su tienda web. Lo que sigue es la comparación de las dos opciones con cifras, el montaje que eligieron, y los tres problemas que aparecieron —dos de ellos por valores por defecto.

La comparación, hecha con datos:

```
requisitos
  la tienda y la API deben estar bajo el mismo dominio
  WAF con reglas propias                          clase 209
  registros de la distribución para análisis de coste
  caché distinta por ruta: 4 comportamientos
  ramas de vista previa para cada propuesta de cambio

Amplify
  cubre    ramas de vista previa, certificado, reversión
  no cubre WAF propio, registros, comportamientos por ruta
           y otro origen para /api/*

decisión
  producción            S3 + CloudFront montado
  vistas previa         Amplify, con caducidad automática a
                        los 14 días
  motivo de la caducidad   las ramas de vista previa no se
                        borran solas y facturan       ley 25
```

El montaje.

```
bucket
  privado, sin alojamiento estático
  bloqueo de acceso público activado
  versionado activado
  cifrado con clave gestionada
  regla de caducidad: versiones antiguas a 30 días

distribución
  origen 1   el bucket, con control de acceso al origen
  origen 2   API Gateway, para /api/*             clase 207
  TLS mínimo 1.2, redirección a HTTPS
  compresión activada
  registros a un bucket propio, caducidad de 90 días
  WAF asociado                                    clase 209

comportamientos
  /assets/*      caché 1 año, inmutable; clave = ruta
  /media/*       caché 7 días; clave = ruta
  /api/*         sin caché; origen 2; reenvía
                 Authorization y Content-Type
  por defecto    caché 0 con revalidación; función de borde
                 de enrutado
```

Problema 1 · Todos los 404 devolvían 200.

```
montaje inicial
  página de error personalizada: 404 → /index.html con 200
  funcionaba: las rutas de la aplicación cargaban bien

lo que se descubrió tres semanas después
  · un despliegue subió el HTML sin uno de los recursos
    el navegador pidió /assets/app.9c2f1a.js
    la distribución devolvió index.html con código 200
    el navegador intentó ejecutar HTML como JavaScript
    → pantalla en blanco, sin ningún error en los paneles
    → la tasa de errores era del 0 %               ley 13

  · el buscador había indexado 1.900 rutas inexistentes
    generadas por enlaces rotos de una campaña

corrección
  función de borde con esta lógica
    si la ruta contiene un punto (tiene extensión) → pasa
      tal cual; si no existe, 404 real
    si no → reescribe a /index.html, código 200
```

```
  y una alerta nueva
    «peticiones a /assets/* que devuelven 404»
    → habría detectado el despliegue incompleto en 30 s
```

Problema 2 · La invalidación masiva en cada publicación.

```
la canalización hacía
  aws s3 sync ... --delete
  aws cloudfront create-invalidation --paths "/*"

efectos medidos
  tasa de aciertos tras cada publicación   97 % → 12 %
  tiempo en recuperarla                    ~40 min
  peticiones al origen en ese periodo      ×31
  y dos veces, con despliegues en hora punta, el origen se
    saturó                                       clase 197
  invalidaciones facturadas al mes         1.100 rutas

y el efecto del --delete
  los recursos antiguos se borraban al publicar
  → quien tenía la tienda abierta veía la página romperse
  → 3 incidentes reportados por atención al cliente,
    clasificados como «error del navegador»

corrección
  construcción con huella en todos los recursos
  publicación en tres pasos
    1  subir /assets/* nuevos (sin borrar)
    2  subir HTML
    3  invalidar solo /index.html y /
  recursos antiguos: regla de caducidad a 30 días

resultado
  tasa de aciertos tras publicar    97 % → 96,4 %
  invalidaciones al mes             1.100 → 62
  incidentes por publicación             3 → 0
  tiempo de publicación             4 min → 1 min
```

Problema 3 · El bucket seguía siendo alcanzable.

```
la prueba negativa, ejecutada en la revisión de seguridad
  pedir un objeto por la URL directa del bucket
  → FUNCIONÓ

causa
  el alojamiento de sitio web estático se había activado
  durante el montaje inicial y no se desactivó
  la política del bucket tenía una regla antigua que
  permitía lectura pública, añadida «para probar»  ley 25

qué implicaba
  se podía descargar todo el contenido saltándose
    el WAF
    los registros
    la caché → pagando salida a precio de S3   clase 200
  y en los registros del bucket aparecían 41.000 peticiones
    al mes por esa vía, de rastreadores

corrección
  alojamiento estático desactivado
  política del bucket reducida al control de acceso al
    origen
  bloqueo de acceso público activado
  alerta: «peticiones directas al bucket > 0»
  y la prueba negativa añadida al calendario trimestral
```

La reversión, probada:

```
ensayo
  publicar una versión con un fallo evidente
  revertir subiendo el index.html anterior desde el
  versionado e invalidándolo

  tiempo medido                        38 s
  recursos antiguos disponibles        sí (no se borran)
```

```
  usuarios afectados durante la reversión  los que cargaron
                                        en esos 38 s
```

El resultado, tres meses después:

```
tasa de aciertos en régimen                97 %       97,4 %
tasa de aciertos tras publicar             12 %       96,4 %
invalidaciones al mes                     1.100          62
peticiones directas al bucket            41.000/mes       0
404 reales que se veían como 200            todos         0
tiempo de reversión                      reconstruir     38 s
incidentes por publicación                   3/trim        0
coste mensual de la distribución          1.240 €       890 €
```

La lección que esta clase deja: dos de los tres problemas venían de valores por defecto que funcionaban: convertir los 404 en 200 hacía que las rutas de la aplicación cargasen, y el alojamiento estático del bucket hacía que el contenido se sirviera. Los dos escondían algo: un despliegue incompleto que no generaba ningún error y cuarenta y un mil peticiones al mes por un camino sin WAF ni registros. Y el tercero —invalidar todo en cada publicación— costaba cuarenta minutos de caché fría por despliegue y se resolvió cambiando el orden de tres comandos.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-17-aws-production-architecture/205-hosting-progresivo-con-amplify-s3-y-cloudfront/
lab.py
```

El laboratorio selecciona el motor de práctica delivery 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 aws-static-platform 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 pipeline con gates, promoción y rollback. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye aws-static-platform 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 errores 404 reales se ven como respuestas correctas	La página de error personalizada convierte todos los 404 en 200 con index.html	Reescribe a index.html solo las rutas sin extensión, con una función de borde, y alerta sobre 404 en recursos.
La tasa de aciertos se hunde tras cada publicación	Se invalida con comodín en cada despliegue	Construye con huella en los nombres, sube el HTML al final e invalida solo el HTML.
La página se rompe para quien la tenía abierta al publicar	Se borran los recursos antiguos al sincronizar	No borres al publicar; retira los recursos viejos con una regla de caducidad a 30 días.
Se puede descargar el contenido saltándose la distribución	El bucket tiene alojamiento estático o una política pública heredada de las pruebas	Bucket privado con control de acceso al origen, bloqueo de acceso público, y prueba negativa que compruebe que la URL directa falla.
La factura de construcción sube sin control	Las ramas de vista previa no caducan y siguen construyendo	Pon caducidad automática a las vistas previa; lo provisional sin fecha se queda para siempre.
El certificado no se puede asociar a la distribución	Se emitió en una región distinta de us-east-1	Emite el certificado de una distribución de CloudFront en us-east-1, aunque el resto del sistema esté en otra región.

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. ¿En qué casos compensa el alojamiento gestionado y en cuáles el montaje propio?
2. ¿Qué valores por defecto del bucket y de la distribución hay que cambiar para producción?
3. ¿Cuál es el defecto de resolver el enrutado con la página de error 404?
4. ¿Qué cinco pasos evitan invalidar la caché entera en cada publicación?
5. ¿Qué prueba negativa comprueba que el bucket está realmente cerrado?

Referencias

- AWS (2025). Amplify Hosting user guide. <https://docs.aws.amazon.com/amplify/latest/userguide/welcome.html>
- AWS (2025). Restricting access to an Amazon S3 origin (origin access control). <https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/private-content-restricting-access-to-s3.html>
- AWS (2025). CloudFront cache policies and cache key. <https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/controlling-the-cache-key.html>
- AWS (2025). CloudFront Functions. <https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cloudfront-functions.html>
- AWS (2025). Blocking public access to your Amazon S3 storage. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-control-block-public-access.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 17 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 204 · Proyecto: red multi-región y multi-cloud	Parte 17 · Programa	206 · OIDC de GitHub y GitLab hacia AWS sin secretos →

Evaluación — 205 Hosting progresivo con Amplify, S3 y CloudFront

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 aws-static-platform 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.

206 — OIDC de GitHub y GitLab hacia AWS sin secretos

Parte: 17 — AWS: arquitectura, automatización y operación en producción Nivel: avanzado · Horas estimadas: 4
Laboratorio: iam · Estado: EXECUTABLECORE

Propósito

Eliminar las claves de acceso de larga duración de las canalizaciones, sustituyéndolas por federación con testigos de corta vida. La clase explica el mecanismo pieza a pieza —proveedor de identidad, condiciones de confianza, testigo y rol—, insiste en la parte donde se cometen los errores graves (la condición de sujeto mal escrita permite que cualquier repositorio del mundo asuma tu rol), y aborda la migración de decenas de canalizaciones sin romperlas.

Resultados de aprendizaje

Al finalizar podrás:

1. Explicar el intercambio de testigo por credenciales temporales.
2. Configurar el proveedor de identidad y la política de confianza correctamente.
3. Escribir condiciones de sujeto que no dejen huecos.
4. Migrar canalizaciones existentes sin interrumpirlas.
5. Verificar con pruebas negativas que nadie más puede asumir el rol.

Conceptos centrales

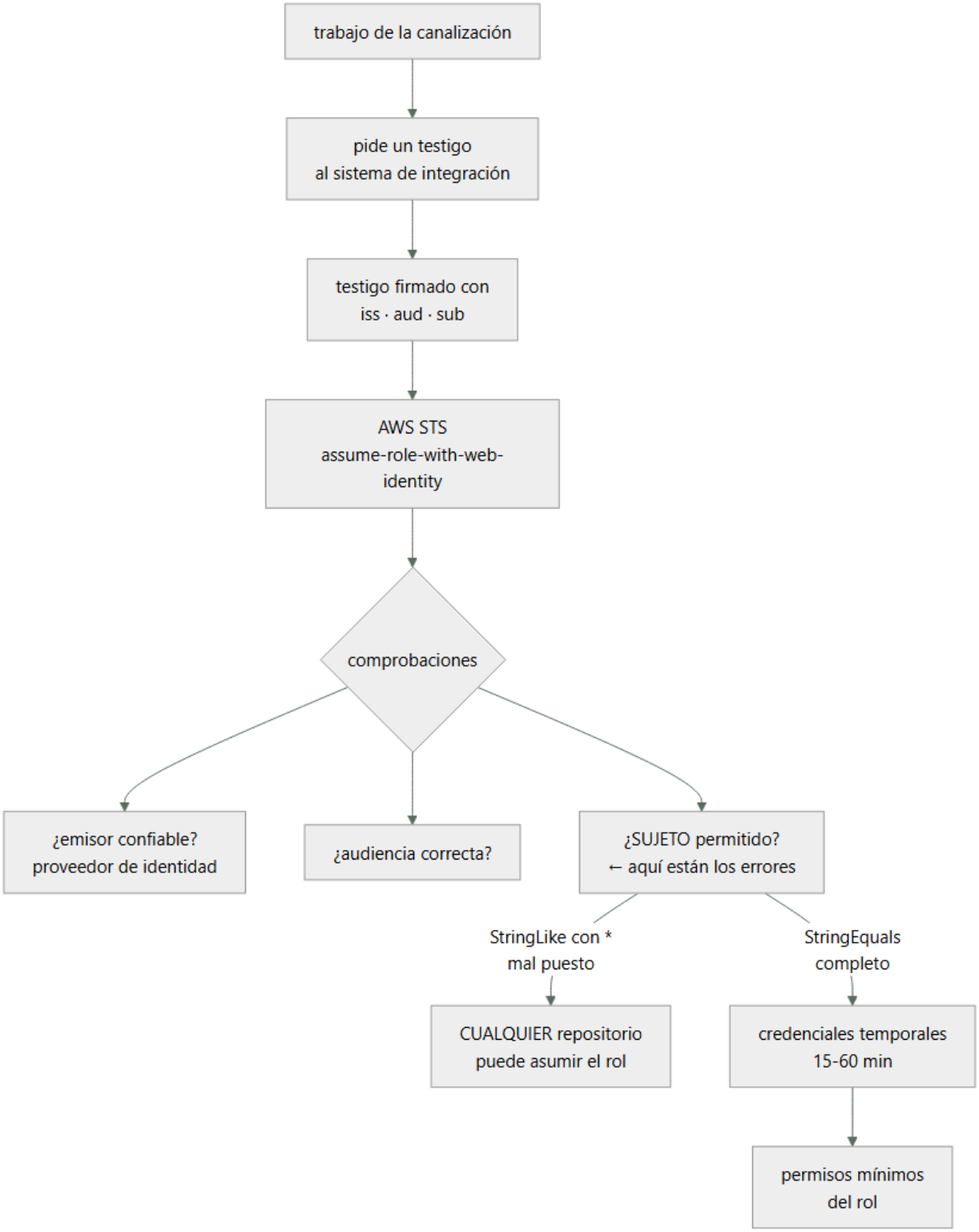
Concepto	Comprensión verificable
federación por identidad de carga	Intercambiar un testigo firmado por el sistema de integración por credenciales temporales del proveedor de nube.
proveedor de identidad	Recurso que declara en qué emisor se confía y con qué audiencia.
política de confianza	Regla del rol que dice quién puede asumirlo y bajo qué condiciones.
sujeto	Campo del testigo que identifica el repositorio, la rama, el entorno o la etiqueta que solicita el acceso.
audiencia	Destinatario declarado del testigo. Evita que un testigo emitido para otro servicio sirva aquí.
entorno protegido	Mecanismo del sistema de integración que exige aprobación antes de que un trabajo obtenga el testigo.

Modelo mental

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

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. Cómo funciona el intercambio

El mecanismo tiene cuatro piezas y conviene entenderlas antes de configurarlas.

```
1  EL SISTEMA DE INTEGRACIÓN EMITE UN TESTIGO
   firmado por él, con vida de minutos, que dice
     iss   quién lo emite (el servicio de integración)
     aud   para quién es
     sub   QUIÉN lo pide: organización, repositorio, rama,
           entorno o etiqueta
```

```
     y otros campos: rama, flujo de trabajo, ejecutor…

2  AWS TIENE DECLARADO UN PROVEEDOR DE IDENTIDAD
   que dice «confío en los testigos firmados por este
   emisor, con esta audiencia»

3  EL ROL TIENE UNA POLÍTICA DE CONFIANZA
   que dice «pueden asumirme los testigos de ese proveedor
   cuyo sujeto cumpla ESTAS condiciones»

4  EL TRABAJO INTERCAMBIA testigo por credenciales
   temporales, de 15 a 60 minutos, con los permisos del rol
```

Y lo que se gana, que es lo que justifica la migración:

```
sin claves de acceso guardadas como secretos
  → no hay nada que robar del repositorio
  → no hay nada que rotar                          clase 159
  → si el repositorio se compromete, el atacante necesita
    además cumplir las condiciones del sujeto
credenciales de minutos, no de años
y cada uso queda registrado con el sujeto que lo pidió
  → auditoría atribuible                          clase 141
```

Y una comparación con lo que sustituye:

```
CLAVE DE ACCESO GUARDADA COMO SECRETO
  vive años
  sirve desde cualquier sitio del mundo
  se copia con un simple volcado de variables de entorno
  y su rotación es un procedimiento manual que nadie hace
                                                    ley 22

→ en la clase 179, cuatro servicios compartían una clave
  estática de tres años
```

Y el ámbito de aplicación, que es más amplio de lo que suele usarse:

```
el mismo mecanismo sirve para
  GitHub Actions y GitLab CI hacia AWS
  y también hacia Azure y Google Cloud
  cargas en Kubernetes hacia AWS                  clase 213
  y cualquier emisor que firme testigos con el formato
    estándar
```

2. Donde se cometen los errores graves

La configuración es corta y tiene un punto donde un error convierte el mecanismo en algo peor que una clave estática.

```
LA CONDICIÓN DE SUJETO

  ✗ CATASTRÓFICO
    "StringLike": { "...:sub": "*" }
    → CUALQUIER repositorio de CUALQUIER organización del
      mundo puede asumir tu rol
    → basta con que alguien sepa el nombre del rol

  ✗ MUY MALO
    "StringLike": { "...:sub": "repo:miorg/*" }
    → cualquier repositorio de tu organización, incluidos
      los que cree un becario
    → y si alguien puede crear un repositorio, puede
      desplegar en producción

  ✗ SUTIL Y FRECUENTE
    olvidar la condición de AUDIENCIA
    → un testigo emitido para otro servicio podría valer

  ✓ CORRECTO
    "StringEquals": {
      "...:aud": "<audiencia declarada>",
      "...:sub": "repo:miorg/mirepo:environment:produccion"
    }
    → un repositorio, un entorno, exacto
```

Y las formas de sujeto que conviene conocer:

```
repo:org/repo:ref:refs/heads/main       una rama concreta
repo:org/repo:environment:produccion    un entorno protegido
repo:org/repo:ref:refs/tags/v*          etiquetas (con
                                        StringLike, y con
                                        cuidado)
repo:org/repo:pull_request              propuestas de cambio
                                        ← NUNCA para producción
```

Y la última merece énfasis:

```
una propuesta de cambio la puede abrir cualquiera desde una
bifurcación
→ dar permisos de producción a ese sujeto equivale a dar
  permisos de producción a cualquiera
→ los trabajos de propuesta de cambio usan un rol de solo
  lectura, y nada más
```

El entorno protegido, que es la pieza que casi nadie usa y la que más aporta:

```
un entorno del sistema de integración puede exigir
  aprobación manual de personas concretas
  que la rama sea una determinada
  una espera mínima

y el testigo incluye el nombre del entorno en el sujeto
→ atar el rol de producción al entorno «produccion»
  significa que ningún flujo puede obtenerlo sin pasar por
  la aprobación                                  clase 106
```

Y una comprobación que hay que hacer siempre:

```
intenta asumir el rol desde
  otra rama                    → debe fallar
  otro repositorio             → debe fallar
  una bifurcación              → debe fallar
  un flujo sin el entorno      → debe fallar
→ y si alguna funciona, la condición está mal escrita
                                                  ley 22
```

3. Permisos del rol y separación

Quitar la clave estática no sirve de nada si el rol federado puede hacerlo todo.

```
UN ROL POR PROPÓSITO, NO UNO QUE VALGA PARA CUALQUIER COSA
  rol de lectura           propuestas de cambio: validar,
                           planificar, analizar
  rol de despliegue de
    preproducción          escribir en preproducción
  rol de despliegue de
    producción             escribir en producción, atado al
                           entorno protegido
  rol de publicación de
    artefactos             subir imágenes al registro

→ y cada uno con permisos mínimos y con su condición de
  sujeto propia                                  clase 134
```

Y los permisos que nunca debe tener el rol de la canalización:

```
modificar políticas de identidad o crear roles
  → si puede crear un rol con más permisos, tiene todos
desactivar el registro de auditoría
borrar copias de seguridad
modificar la propia política de confianza
  → podría ampliarse a sí mismo

→ estos permisos se ponen en una barrera de la organización,
  para que ni siquiera un administrador de la cuenta pueda
  concederlos                                    clase 169
```

Y dos controles que refuerzan:

```
DURACIÓN MÍNIMA
  la sesión dura lo que dure el trabajo, no una hora por
  defecto
```

```
CONDICIONES ADICIONALES EN LA POLÍTICA DE PERMISOS
  restringir por región
  restringir por etiqueta de recurso
  exigir que la petición venga de la sesión federada, no de
    una credencial cualquiera
```

Y una decisión de arquitectura sobre cuentas:

```
el rol de producción vive en la CUENTA de producción
la canalización no tiene ningún acceso permanente a esa
cuenta
→ y el paso de producción está en un flujo separado, con
  entorno protegido
→ así el compromiso del repositorio no da acceso a
  producción sin una aprobación humana        clases 108, 189
```

4. Migrar sin romper

Migrar decenas de canalizaciones a la vez es donde se rompen las cosas. El patrón es el de expandir y contraer.

```
1  CREAR el proveedor de identidad y los roles, con
   condiciones estrictas
   → sin tocar nada de lo existente

2  MIGRAR UNA canalización poco crítica
   y comprobar que funciona y que las pruebas negativas
   fallan como deben

3  MIGRAR EL RESTO por lotes
   las claves antiguas SIGUEN existiendo

4  MEDIR EL USO de cada clave antigua
   → una clave sin uso durante N días es candidata a
     desactivar
   → y las que siguen usándose revelan canalizaciones que
     nadie sabía que existían

5  DESACTIVAR, no borrar, y esperar
   → si algo se rompe, se reactiva en segundos

6  BORRAR, y con ellas los secretos del repositorio
```

Y el paso 4 es el que descubre cosas:

```
el informe de credenciales y los registros de acceso dicen
  qué claves existen, cuándo se usaron por última vez y
  desde dónde
→ las claves sin uso en 90 días son riesgo puro
→ y las que se usan desde direcciones inesperadas son otra
  cosa                                          clase 141
```

Lo que hay que vigilar una vez migrado:

```
usos del rol federado, con el sujeto que lo pidió
intentos de asunción DENEGADOS
  → un aumento significa que alguien está probando
claves de acceso de larga duración que existan todavía
  → función de aptitud: cero permitidas            clase 190
cambios en las políticas de confianza
  → auditar siempre; ampliar una condición es el camino
    para saltarse todo lo anterior
```

Y una limitación que hay que conocer:

```
no todo se puede federar
  herramientas de terceros que solo aceptan clave y secreto
  sistemas antiguos
→ para esos, credenciales de rotación automática y ámbito
  mínimo, con fecha de revisión escrita             ley 25
```

Y la lista de comprobación de la clase:

- ■ existe el proveedor de identidad con la audiencia declarada
- ■ ninguna condición de sujeto usa comodín amplio
- ■ la condición de audiencia está presente
- ■ producción está atada a un entorno protegido con

```
  aprobación
■ las propuestas de cambio usan un rol de solo lectura
■ hay un rol por propósito, con permisos mínimos
■ ningún rol de canalización puede crear roles ni modificar
  su confianza
■ la duración de la sesión es la del trabajo
■ las pruebas negativas desde otra rama, repositorio y
  bifurcación fallan
■ no queda ninguna clave de acceso de larga duración
■ hay función de aptitud que lo comprueba
■ se vigilan los intentos denegados y los cambios de
  confianza
```

Y el cierre que enlaza con la clase siguiente: con una canalización que puede desplegar sin secretos, queda desplegar algo. La aplicación sin servidores —funciones, pasarela de API y su definición como código— es la materia de la clase 207.

Ejemplo trabajado

CloudShop migra 41 canalizaciones a federación. Lo que sigue es la condición de sujeto mal escrita que encontraron en la primera revisión, la migración por lotes, y las once claves que aparecieron al medir el uso.

El punto de partida:

```
canalizaciones                                      41
claves de acceso de larga duración en secretos      23
  antigüedad media                            2,1 años
  con permisos de administrador                    6
  compartidas entre varios repositorios            4
rotaciones realizadas en 3 años                    0
```

El primer intento, y el fallo que encontró la revisión.

```
un equipo había migrado ya su canalización, en marzo
su política de confianza decía

  "Condition": {
    "StringLike": {
      "token.actions.githubusercontent.com:sub":
        "repo:cloudshop/*"
    }
  }

lo que eso permitía
  cualquier repositorio de la organización cloudshop podía
  asumir el rol
  → y ese rol tenía permisos de escritura en producción

y lo que lo hacía peor de lo que parecía
  crear un repositorio en la organización lo podía hacer
  cualquiera de los 214 miembros
  → cualquiera podía desplegar en producción creando un
    repositorio nuevo
  → sin aprobación, sin revisión, sin dejar rastro obvio

y faltaba además la condición de audiencia

la prueba negativa que lo demostró
  se creó un repositorio de prueba en la organización
  se escribió un flujo que asumía el rol
  → funcionó a la primera
  tiempo desde la idea hasta desplegar en producción: 6 min
```

Y el análisis:

```
esta configuración era PEOR que la clave estática que
sustituía
  la clave, al menos, estaba en un secreto de un repositorio
  concreto
  esto era accesible desde cualquier repositorio nuevo
→ y se había revisado y aprobado, porque «quitaba el
  secreto»                                    clase 191
```

La configuración corregida.

```
proveedor de identidad, uno por sistema de integración,
con la audiencia declarada

ROLES, uno por propósito

  cloudshop-ci-lectura
    sub   repo:cloudshop/tienda:pull_request
    aud   comprobada
    puede validar plantillas, planificar cambios, leer
          artefactos
    NO puede escribir nada

  cloudshop-ci-preproduccion
    sub   repo:cloudshop/tienda:ref:refs/heads/main
    puede desplegar en la cuenta de preproducción

  cloudshop-ci-produccion
    sub   repo:cloudshop/tienda:environment:produccion
    vive en la CUENTA de producción
    entorno protegido con aprobación de 2 personas de una
      lista, y espera mínima de 5 minutos
    permisos mínimos, restringidos por región y por etiqueta

  cloudshop-ci-artefactos
    sub   repo:cloudshop/tienda:ref:refs/heads/main
    puede subir al registro de imágenes, nada más

BARRERA DE ORGANIZACIÓN
  ningún rol de canalización puede
    crear o modificar roles
    modificar políticas de confianza
    desactivar el registro de auditoría
    borrar copias de seguridad
  → denegación explícita a nivel de organización, no de
    cuenta                                          clase 169
```

Las pruebas negativas, ejecutadas tras corregir:

```
✓  asumir el rol de producción desde otra rama      denegado
✓  asumir desde otro repositorio de la organización denegado
✓  asumir desde una bifurcación externa             denegado
✓  asumir desde un flujo sin el entorno             denegado
✓  asumir con un testigo de otra audiencia          denegado
✗  el rol de preproducción podía leer un bucket de
   producción
   → una política heredada del rol antiguo permitía
     s3:GetObject sobre un comodín
   → corregido; y añadida condición de etiqueta de entorno
✓  el rol de artefactos intentando desplegar        denegado
✓  el rol de canalización creando un rol            denegado
   por la barrera
```

La migración de las 41, por lotes.

```
semana 1     proveedor y roles creados; nada más tocado
semana 2     1 canalización interna, poco crítica
             pruebas negativas ejecutadas
semanas 3-6  32 canalizaciones, en lotes de 8
semanas 5-7  las 8 restantes, que eran las de producción
             → con entorno protegido, requirió coordinar
               aprobadores
semanas 7-11 MEDIR EL USO de las 23 claves antiguas
semana 12    desactivar (no borrar) las sin uso
semana 16    borrar y limpiar los secretos
```

Lo que apareció al medir el uso de las claves antiguas:

```
claves sin uso en 30 días                            12
  → desactivadas sin incidencia

claves aún en uso                                    11
  de canalizaciones ya migradas (residuos)            3
    → un paso del flujo seguía usando la clave
```

```
  de canalizaciones que NO estaban en la lista de 41    8
    · un trabajo programado en una máquina de un equipo
      de datos
    · un script en el portátil de una persona, ejecutado a
      mano cada lunes para subir tarifas       ← clase 192
    · una herramienta de un proveedor externo
    · dos integraciones de sistemas antiguos
    · un sistema de despliegue que se creía retirado en 2023
    · dos usos que nadie pudo explicar

→ el inventario decía 41 canalizaciones
→ la realidad tenía 49                                ley 24
```

Y el tratamiento de los 8:

```
5   migrados a federación o a roles asumidos desde la nube
2   no se pudieron federar (proveedor externo y sistema
    antiguo)
    → credenciales con rotación automática cada 30 días,
      permisos mínimos y fecha de revisión escrita   ley 25
1   se apagó: era el sistema de despliegue retirado
    → llevaba 2 años con credenciales de administrador
      válidas                                        ley 20
```

El resultado:

```
claves de acceso de larga duración            23           2
  con rotación automática                      0           2
  con permisos de administrador                6           0
roles con condición de sujeto amplia           1           0
entornos protegidos con aprobación             0           4
canalizaciones conocidas                      41          49
tiempo para que un miembro cualquiera
  despliegue en producción sin aprobación    6 min   imposible
intentos denegados registrados al mes         n/a          14
  → todos, flujos mal configurados tras un cambio de rama
```

Y la función de aptitud que lo mantiene:

```
«cero claves de acceso de larga duración sin excepción
 registrada»
  excepciones vivas                                    2
  con dueño y fecha de revisión                        2
  → comprobado en cada cambio de infraestructura
                                                clase 190
```

La lección que esta clase deja: la primera migración quitó el secreto y empeoró la seguridad, porque una condición de sujeto con comodín permitía que cualquiera de doscientos catorce miembros desplegara en producción creando un repositorio. Se había revisado y aprobado. Y al medir el uso de las claves antiguas apareció lo de siempre: ocho canalizaciones que no estaban en el inventario, incluida una que llevaba dos años con credenciales de administrador para un sistema que se creía retirado.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-17-aws-production-architecture/206-oidc-de-github-y-gitlab-hacia-aws-sin-secretos/
lab.py
```

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

Evidencia esperada

El artefacto principal es una matriz de acceso mínimo con prueba de denegación. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye aws-oidc-federation 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
Cualquiera de la organización puede desplegar en producción	La condición de sujeto usa un comodín sobre la organización o el repositorio	Usa comparación exacta con repositorio y entorno concretos, y añade siempre la condición de audiencia.
Un flujo de propuesta de cambio obtiene permisos de escritura	El sujeto de propuesta de cambio está permitido en un rol con escritura	Las propuestas de cambio solo asumen un rol de lectura; las bifurcaciones pueden abrirlas desde fuera.
El rol federado puede ampliarse a sí mismo	Tiene permisos para crear roles o modificar su política de confianza	Deniega esos permisos en una barrera de organización, no solo en la política del rol.
Quitar el secreto no mejoró nada	El rol federado conserva permisos amplios heredados del anterior	Un rol por propósito, con permisos mínimos y condiciones de región y etiqueta.
Al borrar las claves antiguas se rompen procesos desconocidos	Se borraron sin medir el uso previo	Mide el uso semanas, desactiva antes de borrar, y trata los usos inesperados como hallazgos de inventario.
Quedan credenciales estáticas que no se pueden federar	Herramientas de terceros o sistemas antiguos que solo aceptan clave y secreto	Rotación automática, ámbito mínimo y excepción registrada con dueño y fecha de revisión.

■ 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 cuatro piezas intervienen en el intercambio de testigo por credenciales?
2. ¿Por qué una condición de sujeto con comodín puede ser peor que una clave estática?
3. ¿Qué aporta atar el rol de producción a un entorno protegido?
4. ¿Qué permisos no debe tener nunca un rol de canalización y dónde se deniegan?
5. ¿Qué se descubre al medir el uso de las claves antiguas antes de borrarlas?

Referencias

- GitHub (2025). About security hardening with OpenID Connect. <https://docs.github.com/en/actions/deployment/security-hardening-your-deployments/about-security-hardening-with-openid-connect>
- GitHub (2025). Configuring OpenID Connect in Amazon Web Services. <https://docs.github.com/en/actions/deployment/security-hardening-your-deployments/configuring-openid-connect-in-amazon-web-services>
- GitLab (2025). Connect to cloud services with OIDC. <https://docs.gitlab.com/ee/ci/cloudservices/>
- AWS (2025). Create an OpenID Connect identity provider in IAM. <https://docs.aws.amazon.com/IAM/latest/UserGuide/idrolesproviderscreateoidc.html>
- AWS (2025). AssumeRoleWithWebIdentity and session policies. <https://docs.aws.amazon.com/STS/latest/APIReference/APIAssumeRoleWithWebIdentity.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 17 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 205 · Hosting progresivo con Amplify, S3 y CloudFront	Parte 17 · Programa	207 · SAM, Lambda, API Gateway y despliegue serverless →

Evaluación — 206 OIDC de GitHub y GitLab hacia AWS sin secretos

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 aws-oidc-federation 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.

207 — SAM, Lambda, API Gateway y despliegue serverless

Parte: 17 — AWS: arquitectura, automatización y operación en producción Nivel: avanzado · Horas estimadas: 4
Laboratorio: serverless · Estado: EXECUTABLECORE

Propósito

Construir y desplegar una aplicación sin servidores en AWS con las decisiones que separan un prototipo de algo que aguanta producción. La clase cubre la definición como código con SAM, la configuración de funciones que de verdad importa —memoria, concurrencia, plazos y arranque en frío—, la elección de pasarela y su coste, y los tres asuntos que rompen estos sistemas: la concurrencia que agota la base, el reintento que duplica y el despliegue que no se puede revertir.

Resultados de aprendizaje

Al finalizar podrás:

1. Definir una aplicación sin servidores como código, con entornos separados.
2. Dimensionar memoria, concurrencia y plazos con criterio y no por defecto.
3. Elegir el tipo de pasarela por coste y por lo que necesitas de ella.
4. Controlar el arranque en frío y decidir cuándo pagar por evitarlo.
5. Desplegar de forma escalonada, con reversión automática.

Conceptos centrales

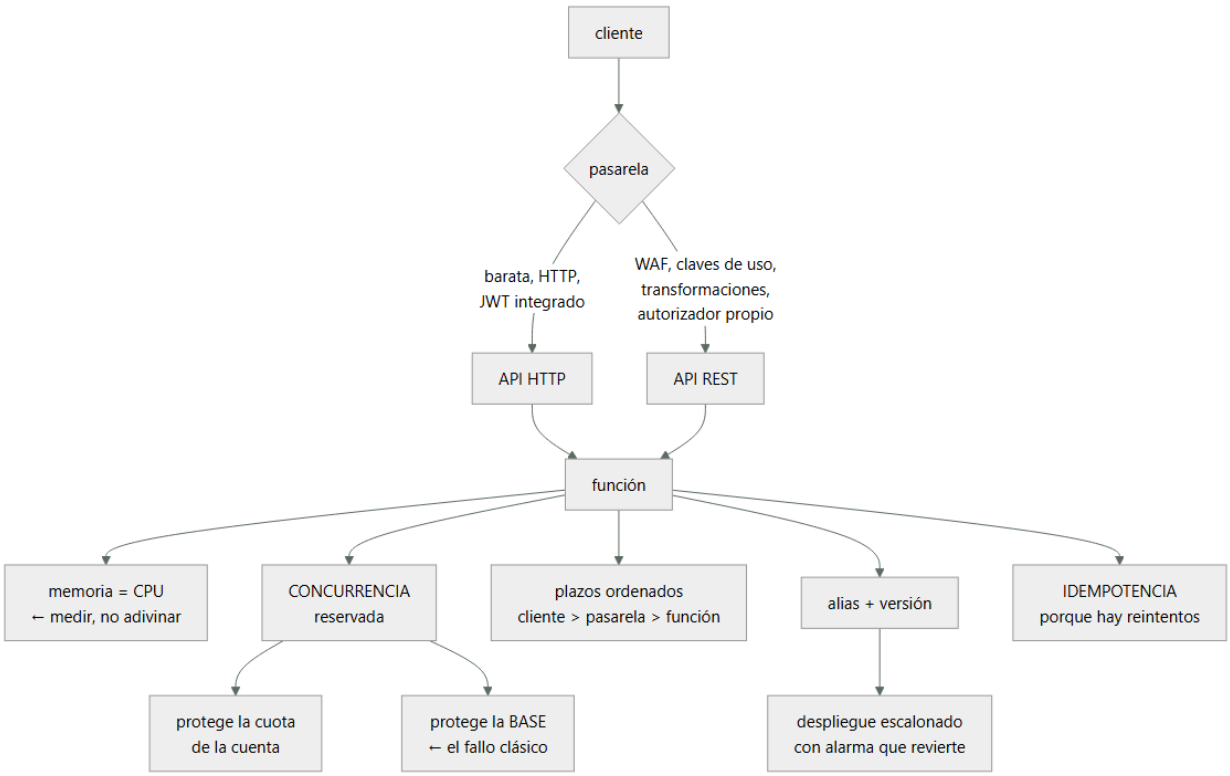
Concepto	Comprensión verificable
SAM	Extensión de CloudFormation que declara funciones, APIs y tablas con menos texto y despliega con un solo comando.
concurrencia reservada	Máximo de ejecuciones simultáneas de una función. Es a la vez un límite y una reserva de la cuota de la cuenta.
concurrencia aprovisionada	Instancias mantenidas calientes. Elimina el arranque en frío y se paga por hora.
arranque en frío	Retraso al crear un entorno de ejecución nuevo. Afecta a los percentiles altos, no a la media.
alias y versión	Puntero estable a una versión concreta de la función. Es lo que permite desplazar tráfico y revertir.
despliegue escalonado	Desplazamiento progresivo del tráfico al alias nuevo, con alarmas que lo revierten solo.

Modelo mental

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

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. Definir como código, con entornos

Una plantilla de SAM declara funciones, rutas, permisos y recursos con mucho menos texto que la plantilla equivalente, y se despliega con un comando.

```
LO QUE UNA PLANTILLA DEBE TENER DESDE EL PRINCIPIO
  parámetros por entorno, sin valores incrustados
  nombres derivados del entorno, no fijos
  permisos por función, no un rol compartido
  etiquetas obligatorias: dueño, entorno, servicio  clase 142
  y ninguna referencia a recursos creados a mano
```

Y los errores de estructura que se pagan pronto:

```
✗ UNA PILA GIGANTE con todo
  → cada despliegue toca todo; un fallo bloquea la pila
  → y las pilas tienen límites de recursos

✓ PILAS POR CICLO DE VIDA
  lo que cambia a diario   funciones y API
  lo que cambia poco       tablas, buckets, colas
  → y las tablas de datos NUNCA en la misma pila que el
    código: un borrado de pila no debe poder llevarse los
    datos

✗ EL MISMO ROL PARA TODAS LAS FUNCIONES
  → una función comprometida alcanza todo         clase 189

✓ UN ROL POR FUNCIÓN, con permisos sobre los recursos
  concretos que usa
```

Y la política de borrado, que casi nadie configura y que evita desastres:

```
los recursos con datos llevan política de retención
  → al borrar la pila, la tabla y el bucket SOBREVIVEN
→ sin esto, un `delete-stack` equivocado en la cuenta
  equivocada borra los datos                      clase 166
```

Los entornos, que se hacen con cuentas y no con prefijos:

```
cuenta de desarrollo · preproducción · producción
```

```
→ la separación por cuenta es la única que impide de verdad
  que un despliegue de preproducción toque producción
                                                  clase 169
→ dentro de cada cuenta, una pila por servicio
```

Y el desarrollo local, que ahorra mucho tiempo y engaña en una cosa:

```
la emulación local sirve para la lógica
no reproduce
  latencias reales, límites de concurrencia, permisos,
  arranques en frío ni el comportamiento de la pasarela
→ las pruebas que importan se hacen en un entorno real,
  aunque sea efímero                              clase 104
```

2. Memoria, concurrencia y plazos

Tres ajustes que vienen con valores por defecto y que deciden coste, latencia y estabilidad.

La memoria, que es en realidad la CPU:

```
en Lambda, la CPU asignada es proporcional a la memoria
→ más memoria puede salir MÁS BARATO, porque la función
  termina antes

cómo se decide
  medir la misma carga con 512, 1.024, 1.792 y 3.008 MB
  y comparar duración × precio
  → hay un mínimo de coste, y casi nunca está en el valor
    por defecto

típicamente
  funciones de cálculo         se benefician mucho
  funciones de espera de red   no; la CPU no acelera esperar
```

La concurrencia, que es donde se rompen estos sistemas:

```
EL FALLO CLÁSICO
  una función sin límite de concurrencia escala a miles de
  ejecuciones simultáneas
  cada una abre una conexión a la base relacional
  → la base agota sus conexiones y CAE
  → y con ella, todo lo demás que la usa

Y es exactamente el problema de la clase 186
  el recurso saturado no es la función: es la base

LAS CORRECCIONES
  concurrencia reservada por función
    = número de conexiones que la base puede dar / conexiones
      por ejecución
  y un intermediario de conexiones (RDS Proxy) si la base
    es relacional
  → o una base pensada para esto                  clase 208
```

Y el otro efecto de la concurrencia reservada:

```
la cuota de concurrencia es DE LA CUENTA, compartida
→ una función que se dispara consume la cuota y las demás
  empiezan a ser rechazadas
→ reservar concurrencia a las funciones críticas es
  reservarles capacidad, no solo limitarlas
```

Los plazos, con la misma regla de la clase 196:

```
cliente > pasarela > función > llamadas de la función

y dos particularidades
  la API HTTP y la REST tienen un plazo máximo propio
    → si la función tarda más, la pasarela corta y la
      función SIGUE ejecutándose y facturando
  el plazo por defecto de una función es bajo, y el máximo
    es alto
    → poner el máximo «por si acaso» significa pagar
      15 minutos de una función colgada

regla   plazo = p99 esperado × 2 o 3, nunca el máximo
```

Y una consecuencia que sorprende:

```
una función con plazo de 15 min que se queda esperando una
dependencia caída
  consume concurrencia durante 15 minutos por invocación
  → agota la cuota y tumba lo demás
→ el plazo corto es un mecanismo de contención  clase 153
```

3. Pasarela, arranque en frío y coste

La elección de pasarela, que tiene consecuencias de coste importantes:

```
API HTTP
  más barata (del orden de un tercio)
  menor latencia añadida
  autorizador de testigo integrado                clase 209
  CORS por configuración
  NO tiene   claves de uso y planes, transformaciones de
             petición y respuesta, WAF directo en algunos
             casos, caché integrada

API REST
  todo lo anterior
  autorizadores propios
  WAF asociado
  caché de respuestas
  y despliegues por etapa con variables

FUNCIÓN CON URL DIRECTA
  sin pasarela; la más barata y la más limitada
  → útil para webhooks internos, no para una API pública

BALANCEADOR DE APLICACIÓN → función
  útil si ya hay balanceador y se quiere una sola entrada
```

Y el criterio:

```
empieza por API HTTP
pasa a REST solo si necesitas algo de su lista
→ y si la aplicación va detrás de CloudFront, muchas de
  esas necesidades (WAF, caché) se resuelven ahí
                                                clase 205
```

El arranque en frío, que se exagera y a la vez se ignora donde importa:

```
QUÉ ES
  crear un entorno de ejecución nuevo: descargar el
  paquete, arrancar el intérprete, ejecutar el código de
  inicialización

CUÁNTO
  lenguajes interpretados y paquetes pequeños  100-400 ms
  máquinas virtuales de lenguajes con arranque
    pesado                                     0,5-3 s
  con conexión a una red virtual, hoy, poco más
  y el código de inicialización propio, lo que tarde

A QUIÉN AFECTA
  a los percentiles altos, no a la media
  → con 100 peticiones/s, casi todo va caliente
  → con 1 petición cada 5 minutos, casi todo va frío
```

Y las correcciones, por orden de coste:

```
1  REDUCIR EL PAQUETE                           gratis
   quitar dependencias, empaquetar solo lo necesario

2  MOVER TRABAJO FUERA DE LA INICIALIZACIÓN     gratis
   → pero reutilizar conexiones ENTRE invocaciones sí va
     en la inicialización: se crean una vez

3  ELEGIR UN TIEMPO DE EJECUCIÓN LIGERO         gratis

4  CONCURRENCIA APROVISIONADA                   se paga
   instancias calientes permanentes
```

```
   → tiene sentido en el camino crítico con tráfico
     irregular
   → y se puede programar por horas

5  MANTENER CALIENTE CON INVOCACIONES PERIÓDICAS
   → truco antiguo, poco fiable con concurrencia; evitarlo
```

El coste, que se estima mal siempre en la misma dirección:

```
lo que se cuenta   invocaciones y duración
lo que se olvida
  peticiones de la pasarela            ← suele ser mayor
  transferencia de datos
  registros: ingesta y almacenamiento  ← el sorprendente
  llamadas a otros servicios por petición
  concurrencia aprovisionada, por hora

→ en sistemas de mucho tráfico y poca duración, la pasarela
  y los registros pueden superar al cómputo
```

4. Desplegar, revertir y ser idempotente

El despliegue escalonado, que SAM da hecho y casi nadie activa:

```
versión + alias
  cada despliegue publica una VERSIÓN inmutable
  el alias apunta a una versión
  → el tráfico va al alias

desplazamiento progresivo
  10 % durante 5 min, luego 100 %
  o lineal: 10 % cada minuto

alarmas asociadas
  si la tasa de error o la latencia superan el umbral
  durante el desplazamiento, el alias VUELVE solo a la
  versión anterior
  → reversión automática, sin nadie mirando      clase 102

comprobación previa y posterior
  funciones que validan antes de desplazar y después
```

Y lo que hay que tener en cuenta al revertir:

```
revertir el código es inmediato
revertir un cambio de ESQUEMA de datos, no
→ los cambios de datos siguen la regla de expandir y
  contraer, con el código tolerando ambas formas
                                                clase 188
```

La idempotencia, que en este entorno no es opcional:

```
POR QUÉ HAY REINTENTOS SIEMPRE
  invocación asíncrona: reintenta 2 veces por defecto
  desde una cola: reintenta hasta agotar y va a la cola de
    fallidos                                    clase 210
  desde un flujo de eventos: reprocesa el lote entero si
    falla un elemento
  el cliente reintenta ante un plazo vencido

→ toda función con efecto debe ser idempotente  clase 117
```

Y el mecanismo:

```
clave de idempotencia por operación
  del mensaje, de la cabecera del cliente, o derivada del
  contenido
registro de claves procesadas, con caducidad
y la comprobación ANTES del efecto, con escritura
  condicionada                                  clase 149
```

Lo que hay que vigilar:

```
invocaciones, errores y duración por función
EJECUCIONES RECHAZADAS por límite de concurrencia
  → señal de que la reserva es corta o algo se disparó
concurrencia usada frente a la reservada
```

```
mensajes en la cola de fallidos, y su antigüedad ley 13
plazos vencidos, separados de los errores
y el coste por función, no solo el total
```

Y la lista de comprobación de la clase:

- la plantilla no tiene valores incrustados por entorno
- los datos están en pilas separadas del código
- los recursos con datos tienen política de retención
- hay un rol por función, con permisos concretos
- la memoria se eligió midiendo, no por defecto
- hay concurrencia reservada en las funciones que tocan bases con conexiones limitadas
- los plazos son múltiplos del p99, no el máximo
- los plazos están ordenados cliente > pasarela > función
- la pasarela elegida corresponde a lo que se necesita
- el arranque en frío se midió antes de pagar por evitarlo
- el despliegue es escalonado con alarma y reversión automática
- toda función con efecto es idempotente
- hay alerta de ejecuciones rechazadas y de cola de fallidos
- el coste incluye pasarela, transferencia y registros

Y el cierre que enlaza con la clase siguiente: una función que escala a miles de ejecuciones necesita un almacén que escale igual, y eso cambia por completo cómo se diseña el modelo de datos. DynamoDB por patrones de acceso es la materia de la clase 208.

Ejemplo trabajado

CloudShop pasa su API de pedidos a funciones. Lo que sigue es el incidente del primer día de campaña, las cinco correcciones que salieron, y el análisis de coste que reveló dónde estaba el dinero de verdad.

El montaje inicial, con valores por defecto:

```
API REST → 14 funciones
memoria                        128 MB (por defecto)
plazo                          30 s
concurrencia                   sin límite
base                           PostgreSQL gestionada,
                               máximo 400 conexiones
despliegue                     directo, sin alias
funciona en pruebas            perfectamente
```

El incidente del primer día de campaña.

```
11:02   arranca la campaña; el tráfico se multiplica por 22
11:04   la base de datos deja de aceptar conexiones
11:04   CUANTO usa la base falla, incluidos el panel
        interno y los informes
11:31   se identifica la causa
12:20   servicio restablecido

qué pasó
  las funciones escalaron a 3.100 ejecuciones simultáneas
  cada una abría su propia conexión
  3.100 > 400
  → la base agotó conexiones y empezó a rechazar
  → las funciones fallaban, el cliente reintentaba, y los
    reintentos abrían más conexiones                clase 186

y un efecto secundario
  las 3.100 ejecuciones consumieron la cuota de concurrencia
  de la CUENTA
  → funciones de otros servicios empezaron a ser rechazadas
  → incluida la que procesa los correos de confirmación
```

Las cinco correcciones.

```
1  CONCURRENCIA RESERVADA, calculada
   conexiones disponibles para la API        320
   conexiones por ejecución                    1
   margen                                    20 %
   → concurrencia reservada de las funciones que tocan la
```

```
     base: 256 en total, repartida
       crear pedido        120
       consultar pedido     80
       modificar pedido     40
       resto                16
   y las funciones que NO tocan la base quedan sin reserva

   efecto   por encima de 256, las peticiones se rechazan
            con 429 rápido, en vez de tumbar la base
            → rechazar pronto es mejor que encolar
                                                clase 186

2  INTERMEDIARIO DE CONEXIONES
   se añadió un proxy de base de datos
   → 3.100 ejecuciones comparten 180 conexiones reales
   → la concurrencia reservada se pudo subir a 900
   coste    +140 €/mes

3  PLAZOS ORDENADOS Y CORTOS
   antes   cliente 60 s · pasarela 29 s · función 30 s
   → la pasarela cortaba a los 29 s y la función seguía
     ejecutándose 1 s más, facturando y ocupando
     concurrencia
   después cliente 12 s · pasarela 10 s · función 8 s
           llamadas de la función: 3 s

   efecto durante una degradación de la pasarela de pago
     antes    cada invocación ocupaba 30 s de concurrencia
     después  8 s
     → la cuota aguanta 3,7 veces más tiempo

4  MEMORIA, MEDIDA
   la función de crear pedido, con la misma carga

     memoria   duración   coste relativo
      128 MB    3.900 ms       1,00
      512 MB    1.010 ms       1,04
    1.024 MB      520 ms       1,07
    1.792 MB      340 ms       1,22

   → el mínimo estaba en 128 MB por poco, pero la latencia
     era inaceptable
   → se eligió 1.024 MB: 7 % más caro y 7,5 veces más rápido
   → y el p99 de la API bajó de 4,2 s a 0,7 s

   y la función de generar informes, de cálculo puro
      128 MB   48 s   1,00
    3.008 MB    2,4 s  0,52
   → más memoria, la MITAD de coste

5  DESPLIEGUE ESCALONADO
   alias y versiones
   desplazamiento 10 % durante 5 min, luego 100 %
   alarmas: tasa de error > 1 % o p99 > 1,5 s
   reversión automática

   en los 8 meses siguientes
     despliegues                          186
     revertidos automáticamente             7
     tiempo medio de reversión          2 min
     incidentes causados por despliegue     0
```

La idempotencia, que faltaba.

```
en el incidente, los reintentos del cliente crearon pedidos
duplicados
  pedidos duplicados detectados                  64
  cobrados dos veces                             11

corrección
  la petición de crear pedido lleva una clave de
  idempotencia generada por el cliente
  la función escribe con condición «si no existe esa clave»
```

```
  → un reintento devuelve el pedido original, no crea otro

y la prueba negativa
  enviar la misma petición 50 veces en paralelo
  → 1 pedido creado, 49 respuestas idénticas      ✓
```

El análisis de coste, que dio la sorpresa.

```
estimación inicial del equipo
  cómputo de funciones                        340 €/mes
  «lo demás es despreciable»

factura real del primer mes completo
  cómputo de funciones                        410 €
  peticiones de API Gateway (REST)            890 €   ←
  registros: ingesta                          620 €   ←
  registros: almacenamiento                   180 €
  transferencia                               210 €
  proxy de base de datos                      140 €
  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  total                                     2.450 €

  → el cómputo era el 17 % de la factura
```

Y las tres correcciones de coste:

```
PASARELA
  se revisó qué se usaba de la API REST
    claves de uso        no
    transformaciones     no
    caché                no (la hace CloudFront)  clase 205
    WAF                  sí → pero se puede poner en
                         CloudFront
  → migración a API HTTP
  890 € → 310 €

REGISTROS
  las funciones registraban cada petición completa, con
  cuerpo
  → nivel de registro por entorno; en producción solo
    errores y muestreo del 1 % de las peticiones correctas
  → caducidad de 14 días en lugar de indefinida
  620 € → 95 €;  almacenamiento 180 € → 25 €

TRANSFERENCIA
  las respuestas incluían campos que el cliente no usaba
  210 € → 120 €

total                            2.450 € → 1.100 €
```

El resultado, seis meses después:

```
p99 de la API                            4,2 s       0,7 s
caídas de la base por agotamiento       1 (campaña)      0
pedidos duplicados                        64           0
despliegues revertidos a mano              3           0
  (automáticos)                            —           7
coste mensual                          2.450 €     1.100 €
  proporción de cómputo                   17 %        37 %
ejecuciones rechazadas en pico           n/a      412/día
  → a propósito: mejor rechazar que tumbar la base
```

La lección que esta clase deja: el sistema funcionaba perfectamente en pruebas y cayó en cuatro minutos el día de campaña, porque el recurso saturado no era la función sino las conexiones de la base —el mismo problema de la clase 186, con otra tecnología—. Y del análisis de coste salió lo que la hipótesis de la parte anticipaba: el cómputo era el 17 % de la factura, y las dos partidas mayores eran la pasarela y los registros, ninguna de las cuales se había estimado.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-17-aws-production-architecture/207-sam-lambda-api-gateway-y-despliegue-serverless/
lab.py
```

El laboratorio selecciona el motor de práctica serverless 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 aws-serverless-api 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 función con límites, reintentos e idempotencia. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye aws-serverless-api 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 base de datos cae en cuanto hay un pico de tráfico	Las funciones escalan sin límite y cada ejecución abre una conexión	Calcula y aplica concurrencia reservada según las conexiones disponibles, y usa un intermediario de conexiones.
Una función descontrolada afecta a servicios que no tienen nada que ver	La cuota de concurrencia es de la cuenta y se consume entera	Reserva concurrencia a las funciones críticas; reservar es también proteger su capacidad.
Se factura tiempo de funciones que ya nadie está esperando	El plazo de la función es mayor que el de la pasarela, o se puso el máximo por si acaso	Ordena los plazos cliente > pasarela > función y fija el de la función en dos o tres veces el p99.
Se crean registros duplicados tras un pico o un fallo	La función no es idempotente y algo la reintenta	Clave de idempotencia por operación y escritura condicionada antes del efecto.
Un despliegue defectuoso llega a todo el tráfico	Se despliega sin alias ni desplazamiento progresivo	Publica versiones, desplaza tráfico por porcentaje y asocia alarmas que reviertan solas.
La factura triplica la estimación	Solo se estimó el cómputo; la pasarela y los registros suelen pesar más	Estima peticiones de pasarela, transferencia e ingesta de registros; ajusta nivel de registro y caducidad por entorno.

■ 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 más memoria puede salir más barato en una función?
2. ¿Cómo se calcula la concurrencia reservada de una función que usa una base relacional?
3. ¿Qué ocurre si el plazo de la función supera el de la pasarela?
4. ¿Cuándo compensa pagar concurrencia aprovisionada?
5. ¿Qué partidas de coste se olvidan al estimar una aplicación sin servidores?

Referencias

- AWS (2025). Serverless Application Model developer guide. <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html>
- AWS (2025). Lambda function scaling and reserved concurrency. <https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html>
- AWS (2025). Choosing between HTTP APIs and REST APIs. <https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html>
- AWS (2025). Operating Lambda: performance optimization and cold starts. <https://aws.amazon.com/blogs/compute/operating-lambda-performance-optimization-part-1/>
- AWS (2025). Amazon RDS Proxy — agrupación de conexiones para funciones. <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/rds-proxy.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 17 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 206 · OIDC de GitHub y GitLab hacia AWS sin secretos	Parte 17 · Programa	208 · DynamoDB por patrones de acceso y single-table design →

Evaluación — 207 SAM, Lambda, API Gateway y despliegue serverless

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 aws-serverless-api 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.

208 — DynamoDB por patrones de acceso y single-table design

Parte: 17 — AWS: arquitectura, automatización y operación en producción Nivel: avanzado · Horas estimadas: 4
Laboratorio: data · Estado: EXECUTABLECORE

Propósito

Diseñar en DynamoDB, donde el modelo se deduce de los patrones de acceso y no al revés, y donde equivocarse en la clave es la decisión más cara de cambiar de todo un sistema. La clase da el método —escribir los patrones antes de tocar nada—, explica el diseño de tabla única con sus ventajas y su coste real, cubre los índices, la capacidad y las particiones calientes, y aborda con honestidad cuándo DynamoDB no es la elección correcta.

Resultados de aprendizaje

Al finalizar podrás:

1. Escribir los patrones de acceso antes de diseñar el modelo.
2. Elegir clave de partición y de ordenación que repartan y sirvan las consultas.
3. Aplicar el diseño de tabla única donde aporta, y no donde no.
4. Dimensionar capacidad y detectar particiones calientes.
5. Decidir cuándo usar otra base de datos.

Conceptos centrales

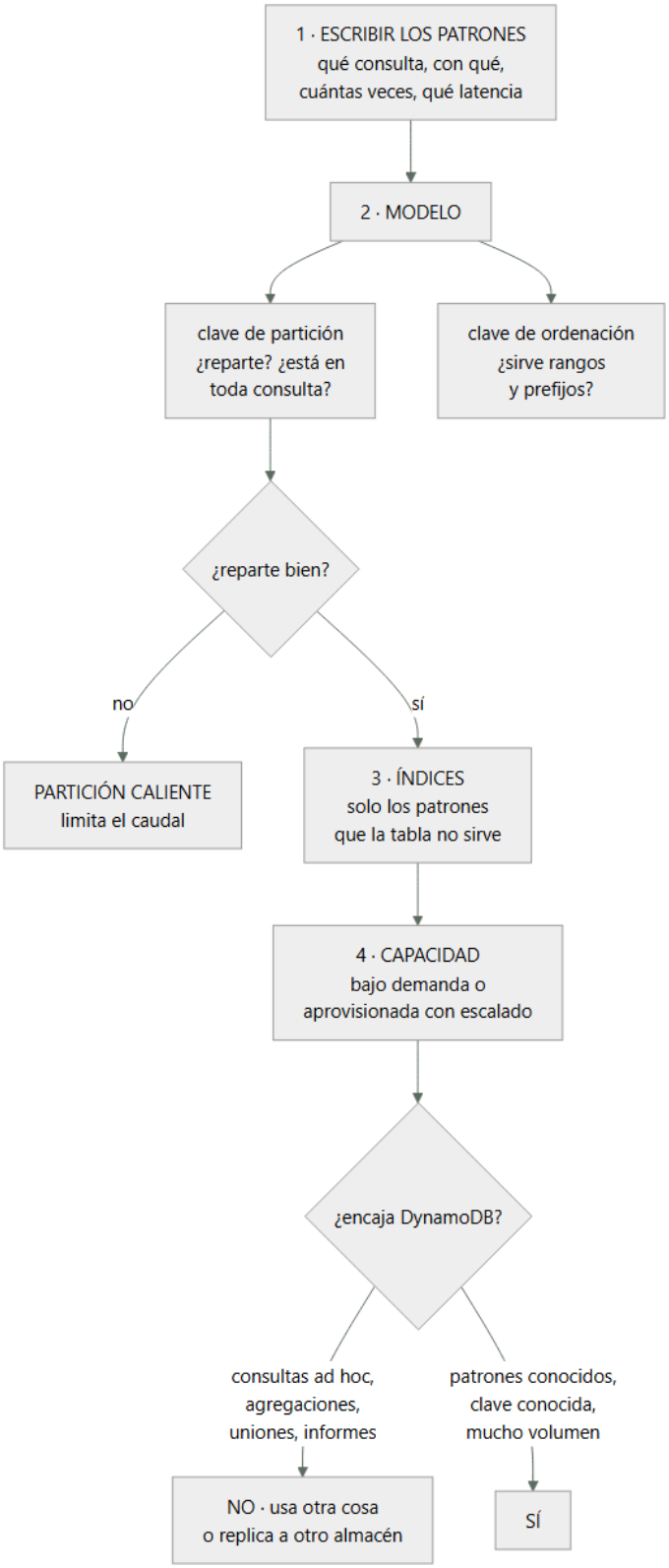
Concepto	Comprensión verificable
patrón de acceso	Consulta o escritura concreta que el sistema debe hacer, con su frecuencia y su latencia exigida.
clave de partición	Determina en qué partición vive el elemento. Decide el reparto de carga y no se puede cambiar.
clave de ordenación	Ordena los elementos dentro de una partición y permite consultas por rango y por prefijo.
tabla única	Guardar varias entidades en una tabla con claves genéricas, para servir consultas relacionadas en una sola operación.
índice secundario global	Vista con otra clave. Tiene su propia capacidad y es eventualmente consistente.
partición caliente	Clave que concentra el tráfico. Limita el caudal aunque la tabla tenga capacidad de sobra.

Modelo mental

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

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. Primero los patrones, siempre

En una base relacional se modela el dominio y las consultas salen después. En DynamoDB eso no funciona: el modelo se deduce de las consultas.

```
EL DOCUMENTO QUE HAY QUE ESCRIBIR ANTES
  nº  patrón                        clave      frecuencia
```

```
   1  obtener pedido por id        id         alta
   2  listar pedidos de un cliente,
      del más reciente al más      cliente    alta
      antiguo, paginado
   3  listar pedidos de un cliente
      en un rango de fechas        cliente+f  media
   4  obtener las líneas de un
      pedido                       pedido     alta
   5  listar pedidos por estado
      para el panel de operaciones estado     baja
   6  obtener el cliente de un
      pedido                       cliente    alta

  y por cada uno
    ¿con qué dato se pide? ← esto es la clave
    ¿cuántas veces por segundo?
    ¿qué latencia se exige?
    ¿se necesita el dato más reciente o vale eventual?
```

Y la razón de que sea obligatorio:

```
DynamoDB solo sabe hacer dos cosas rápido
  obtener un elemento por su clave completa
  obtener un rango de elementos con la misma clave de
    partición, ordenados por la de ordenación

todo lo demás es un recorrido completo de la tabla
  → caro, lento, y que empeora al crecer
```

Y el patrón que revela que el modelo está mal:

```
si para servir una pantalla hacen falta 5 consultas y luego
unir en el código, el modelo no está bien
→ o el modelo cambia, o esa consulta no debería ir aquí
```

Y una advertencia sobre el proceso:

```
los patrones nuevos aparecen. El modelo debe poder
absorberlos con un índice, no con una migración
→ por eso conviene dejar atributos genéricos libres para
  índices futuros                                clase 181
```

2. Claves, y la partición caliente

La clave de partición es la decisión más cara de todo el diseño, porque no se puede cambiar: cambiarla es crear otra tabla y migrar.

```
UNA BUENA CLAVE DE PARTICIÓN
  1  está presente en casi todas las consultas
  2  tiene MUCHOS valores distintos
  3  el tráfico se reparte entre esos valores
  4  y ningún valor concentra demasiados elementos

UNA MALA
  estado del pedido      ← 5 valores; todo cae en «entregado»
  fecha del día          ← todo el tráfico de hoy en una
  tipo de entidad        ← peor todavía
  país                   ← si el 80 % es de un país
```

La partición caliente, que es el fallo característico:

```
CADA PARTICIÓN TIENE UN LÍMITE PROPIO
  del orden de 3.000 lecturas y 1.000 escrituras por segundo

→ una tabla con capacidad para 40.000 escrituras/s puede
  estar rechazando peticiones si todas van a la misma clave
→ el síntoma es un error de caudal excedido con la tabla
  «al 8 % de uso»
```

Y las soluciones, por orden de preferencia:

```
1  CAMBIAR LA CLAVE por una que reparta
   → lo correcto, si aún se está a tiempo

2  AÑADIR UN SUFIJO REPARTIDOR
   cliente#7 → cliente#7#3, con 10 sufijos
```

```
   → reparte la escritura
   → y obliga a hacer 10 consultas al leer y unir
   → solo si el patrón de lectura lo tolera

3  CACHÉ DELANTE para lecturas calientes
   → resuelve lectura, no escritura

4  AGREGAR ANTES DE ESCRIBIR
   si son 5.000 escrituras/s de contadores, agregar en
   ventanas y escribir una vez
```

La clave de ordenación, que es donde está la potencia:

```
con la misma clave de partición se puede pedir
  todo lo que empieza por un prefijo
  todo lo que está entre dos valores
  en orden ascendente o descendente
  con límite y paginación

y por eso las claves de ordenación se construyen para que
el orden lexicográfico signifique algo
  FECHA en formato ISO ordena cronológicamente
  jerarquías con separador: PEDIDO#2025-03#4471
```

Y la técnica que sirve la mayoría de las consultas:

```
clave de ordenación con prefijo por tipo
  CLIENTE#123 · PERFIL
  CLIENTE#123 · PEDIDO#2025-03-14#4471
  CLIENTE#123 · PEDIDO#2025-03-02#4390
  CLIENTE#123 · DIRECCION#casa

→ una consulta con prefijo «PEDIDO#» da los pedidos
  ordenados por fecha
→ una consulta sin prefijo da el cliente entero de una vez
```

3. Tabla única: qué aporta y qué cuesta

El diseño de tabla única consiste en guardar varias entidades juntas para poder traerlas en una sola operación.

```
QUÉ RESUELVE
  la pantalla que necesita cliente, sus pedidos y sus
  direcciones
  con tablas separadas: 3 consultas y unir en el código
  con tabla única: 1 consulta
  → y en un sistema con muchas llamadas, eso importa
                                                clase 152

QUÉ CUESTA, y se cuenta poco
  el modelo deja de ser legible: PK y SK genéricos
  cada patrón nuevo obliga a pensar de nuevo
  las herramientas de exploración no ayudan
  la incorporación de gente nueva es más lenta
  y un error de modelo afecta a todas las entidades
```

Y el criterio honesto:

```
USA TABLA ÚNICA CUANDO
  hay patrones que necesitan varias entidades juntas
  el volumen y la latencia lo justifican
  y el equipo entiende el modelo

NO LA USES CUANDO
  las entidades no se consultan juntas nunca
    → tablas separadas, más simples y sin coste
  el volumen es modesto
    → la ganancia no compensa la complejidad
  el equipo no ha trabajado así antes y no hay tiempo de
    aprender
    → un modelo de tabla única mal hecho es peor que tablas
      separadas
```

Y una regla práctica que reduce el riesgo:

```
agrupa en una tabla lo que se consulta junto y comparte
ciclo de vida
```

```
separado, lo demás
→ es el mismo criterio de cohesión de la clase 183
```

Los índices secundarios, con lo que hay que saber:

```
ÍNDICE GLOBAL
  otra clave de partición y de ordenación
  capacidad PROPIA, que se paga aparte
  eventualmente consistente, siempre
  → si su capacidad se agota, las ESCRITURAS de la tabla
    principal empiezan a fallar     ← el efecto que sorprende

ÍNDICE LOCAL
  misma partición, otra ordenación
  se define al crear la tabla y no se puede añadir después
  limita el tamaño de la partición a 10 GB
  → rara vez merece la pena

PROYECCIÓN
  qué atributos se copian al índice
  solo claves · algunos · todos
  → proyectar todo duplica el almacenamiento y el coste de
    escritura
  → proyectar solo lo necesario y buscar el resto en la
    tabla, si es poco frecuente
```

Y el error de índices más común:

```
crear un índice por cada consulta que se ocurre
→ cada escritura se replica a todos los índices
→ 5 índices = 6 escrituras por cada escritura lógica
→ y el coste de escritura se multiplica por seis
```

4. Capacidad, coste y cuándo no usarla

La capacidad, con la decisión que hay que tomar:

```
BAJO DEMANDA
  se paga por petición; escala sola
  + sin gestión, ideal para tráfico impredecible o bajo
  − más caro por unidad si el tráfico es alto y estable
  − tiene su propio escalado: un pico de 10× desde reposo
    puede encontrar límites al principio

APROVISIONADA CON ESCALADO AUTOMÁTICO
  se reserva capacidad y se ajusta por uso
  + más barata con tráfico alto y previsible
  − el escalado tarda minutos: no cubre un pico brusco
  − requiere vigilar el estrangulamiento

REGLA PRÁCTICA
  empieza bajo demanda, mide un mes, y cambia si el patrón
  es estable y el ahorro es real
```

El coste, con lo que suele olvidarse:

```
lectura y escritura                    lo que se cuenta
almacenamiento
ÍNDICES: su propia escritura y almacenamiento   ← olvidado
copias continuas y su almacenamiento
flujos de cambios y su consumo                  ← olvidado
tablas globales: escritura replicada en cada región
transferencia entre regiones

y dos decisiones que cambian mucho el coste
  LECTURA EVENTUAL en vez de fuerte: la mitad de precio
  → y en la mayoría de las consultas basta      clase 187
  ELEMENTOS PEQUEÑOS: la unidad de lectura cubre 4 KB
  → un elemento de 5 KB cuesta el doble que uno de 4
```

Cuándo NO usar DynamoDB, que es la parte honesta:

```
NO LA USES SI
  las consultas no se conocen de antemano
    informes, exploración, analítica ad hoc
  hacen falta agregaciones (sumas, medias, agrupaciones)
```

```
    → hay que mantenerlas a mano al escribir
  hacen falta uniones entre entidades no relacionadas
  hace falta búsqueda por texto
  el volumen es pequeño y el equipo conoce SQL
    → una base relacional gestionada será más rápida de
      construir y más barata de operar

Y EL PATRÓN QUE RESUELVE LA MAYORÍA DE ESTOS CASOS
  DynamoDB para el camino operativo, de alto volumen y
  patrones conocidos
  + flujo de cambios que replica a un almacén analítico
    para informes y exploración                  clase 150
  → cada uno hace lo que sabe hacer
```

Y las transacciones, que existen y tienen límites:

```
se pueden escribir varios elementos de forma atómica
  con límite de elementos por transacción
  y coste doble
→ útiles para invariantes concretos; no para sustituir a
  una base transaccional                         clase 187
```

Y la lista de comprobación de la clase:

- los patrones de acceso están escritos, con frecuencia y latencia
- la clave de partición está en casi todas las consultas
- la clave de partición reparte: muchos valores y tráfico distribuido
- la clave de ordenación permite prefijos y rangos útiles
- el uso de tabla única está justificado, no copiado
- los índices son los mínimos, con proyección ajustada
- se sabe que un índice saturado hace fallar las escrituras
- las lecturas usan consistencia eventual salvo donde no se pueda
- hay alerta de estrangulamiento y de clave caliente
- el coste incluye índices, flujos y copias
- lo que no encaja se replica a otro almacén, no se fuerza

Y el cierre que enlaza con la clase siguiente: con una API y un almacén que escalan, falta decidir quién puede llamar y qué se hace con lo que llega de fuera. Identidad de usuario, autorizadores y defensa en profundidad es la materia de la clase 209.

Ejemplo trabajado

CloudShop diseña el almacén de pedidos. Lo que sigue es el documento de patrones, el modelo que salió, el error de clave que costó una migración de seis semanas, y la decisión de qué NO poner ahí.

Los patrones, escritos antes de nada:

nº	patrón	se pide con	veces/s	latencia
1	obtener pedido por id	id	400	< 20 ms
2	pedidos de un cliente, recientes primero, paginado	cliente	180	< 50 ms
3	pedidos de un cliente en rango de fechas	cliente+fecha	20	< 50 ms
4	líneas de un pedido	pedido	400	< 20 ms
5	pedidos en estado «pendiente de envío», para el almacén	estado	2	< 1 s
6	datos del cliente de un pedido	cliente	400	< 20 ms
7	pedidos de un día para el cierre contable	fecha	1 vez/día	< 5 min
8	buscar pedidos por texto libre (atención al cliente)	texto	15	< 1 s
9	ventas por categoría y semana	—	informes	—

Y la primera decisión, tomada antes de modelar:

```
los patrones 8 y 9 NO van a DynamoDB
  8  búsqueda por texto → índice de búsqueda, alimentado
     por el flujo de cambios
  9  agregaciones y exploración → almacén analítico, por el
     mismo flujo                                   clase 150
```

```
→ forzarlos habría destrozado el modelo
```

El error de clave, y lo que costó.

```
primer diseño, hecho antes de escribir los patrones
  clave de partición    estado del pedido
  clave de ordenación   fecha#id
  razón                 «el panel del almacén consulta por
                        estado, y es la consulta que nos
                        pidieron primero»

lo que pasó en producción
  los estados son 6
  el 71 % de los pedidos acaba en «entregado»
  y todos los pedidos nuevos entran en «pendiente de pago»
  → dos particiones concentraban el 94 % de la escritura

  síntoma
    errores de caudal excedido a 900 escrituras/s
    con la tabla configurada para 20.000
    y el panel de uso mostrando un 5 %

  y el patrón 1, el más frecuente, no se podía servir:
  para obtener un pedido por id había que conocer su estado
  → se hacía un recorrido completo de la tabla
  → 340 ms de latencia y coste creciente

coste de la corrección
  tabla nueva, doble escritura durante la migración,
  comparación, y corte
  6 semanas de trabajo de 2 personas

qué lo habría evitado
  escribir los nueve patrones antes de elegir la clave
  → 3 horas de trabajo                          ley 14
```

El modelo definitivo, con tabla única.

```
PK                  SK                          entidad
CLIENTE#123         PERFIL                      cliente
CLIENTE#123         DIRECCION#casa              dirección
CLIENTE#123         PEDIDO#2025-03-14#4471      resumen de
                                                pedido
CLIENTE#123         PEDIDO#2025-03-02#4390      resumen
PEDIDO#4471         CABECERA                    pedido
PEDIDO#4471         LINEA#001                   línea
PEDIDO#4471         LINEA#002                   línea

cómo se sirve cada patrón
  1  obtener PEDIDO#4471 / CABECERA          → 1 operación
  2  consultar CLIENTE#123 con prefijo
     «PEDIDO#», descendente, límite 20       → 1 operación
  3  consultar CLIENTE#123 entre
     «PEDIDO#2025-01» y «PEDIDO#2025-03»     → 1 operación
  4  consultar PEDIDO#4471 con prefijo
     «LINEA#»                                → 1 operación
  6  el resumen de pedido bajo CLIENTE ya trae lo necesario
     → 0 operaciones extra

y la pantalla de detalle de cliente
  consultar CLIENTE#123 sin prefijo
  → perfil, direcciones y últimos pedidos en UNA operación
  → antes eran 3 consultas y unión en el código
```

Y la justificación de la tabla única, escrita:

```
se usa porque los patrones 2, 3 y 6 necesitan cliente y
pedidos juntos, con 400 peticiones/s
no se usa para las líneas de producto del catálogo, que
nunca se consultan con lo anterior y viven en su propia
tabla                                          clase 183
```

Los índices, los mínimos.

```
GSI-1   para el patrón 5 (panel del almacén)
```

```
  PK    ESTADO#pendiente-envio
  SK    fecha#id
  proyección   solo id, cliente, fecha, importe
  → 2 consultas/s; no justifica proyectar todo

  y aquí SÍ hay concentración: todos los pendientes en una
  clave
  → aceptable porque son 2 consultas/s y pocos elementos
  → decisión registrada, con la señal que la reabriría:
    «si el panel pasa de 50 consultas/s o si los pendientes
     superan 100.000 elementos»                clase 190

GSI-2   para el patrón 7 (cierre diario)
  PK    FECHA#2025-03-14
  SK    id
  proyección   solo claves
  → se ejecuta una vez al día, fuera de hora punta

NO se creó ningún índice más
  se propusieron 4 más para consultas del panel interno
  → se resolvieron en el almacén analítico          clase 150
  → habrían multiplicado por 3 el coste de escritura
```

La capacidad y el coste.

```
primer mes, bajo demanda
  lecturas                    980 M     490 €
  escrituras                   41 M     205 €
  almacenamiento              210 GB     53 €
  índices (escritura)          82 M     410 €   ←
  flujo de cambios             41 M      18 €
  copias continuas                        68 €
  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  total                                1.244 €

los índices costaban más que las escrituras de la tabla
  → porque GSI-1 recibía cada escritura de pedido y
    proyectaba 4 atributos
  → se ajustó para que solo se indexen los pedidos en
    estados activos, borrando la clave del índice al
    entregar
  → escrituras de índice: 82 M → 19 M;  410 € → 95 €

segundo ajuste
  el 92 % de las lecturas no necesitaban consistencia fuerte
  → cambiadas a eventual:  490 € → 268 €

tercer ajuste, tras 2 meses de datos
  el tráfico resultó estable y alto
  → capacidad aprovisionada con escalado automático
  → 268 € → 171 € en lecturas

total                          1.244 € → 620 €
```

La comprobación de reparto, hecha antes de salir:

```
se simuló un día de campaña con datos reales
  claves de partición distintas                 41.200
  máxima proporción de escritura en una clave      0,8 %
  máxima proporción de lectura en una clave        2,1 %
    → un cliente empresarial con muchos pedidos
    → aceptable; muy por debajo del límite de partición

y la alerta que quedó montada
  «errores de caudal excedido > 0 durante 1 minuto»
  → distingue el estrangulamiento del error de aplicación
```

El resultado:

```
latencia p99 del patrón 1                 340 ms       11 ms
operaciones para la pantalla de cliente        3           1
errores de caudal con la tabla al 5 %         sí          no
coste mensual                              1.244 €     620 €
índices                                        6           2
patrones servidos fuera de DynamoDB            0           2
```

La lección que esta clase deja: la clave de partición se eligió por la primera consulta que pidió negocio y no por los nueve patrones, y esa decisión de una tarde costó seis semanas de migración. El segundo hallazgo fue de coste: los índices costaban más que la tabla, porque recibían cada escritura y proyectaban atributos que casi nadie leía. Y las dos consultas que peor encajaban —búsqueda por texto y agregaciones— se resolvieron sacándolas de DynamoDB, no forzando el modelo.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-17-aws-production-architecture/208-dynamodb-por-patrones-de-acceso-y-single-table-design/lab.py
```

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

Evidencia esperada

El artefacto principal es un modelo de datos ligado a patrones de acceso. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

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

■ Criterio de aceptación

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

Errores frecuentes

Síntoma	Causa probable	Corrección
Errores de caudal excedido con la tabla al 5 % de uso	Partición caliente: el tráfico se concentra en pocas claves	Elige una clave con muchos valores y tráfico repartido; si ya es tarde, reparte con sufijo, cachea o agrega antes de escribir.
Servir una pantalla exige cinco consultas y unir en el código	El modelo se diseñó desde el dominio y no desde los patrones	Escribe los patrones con su frecuencia y latencia antes de modelar, y agrupa en la clave de ordenación lo que se consulta junto.
Obtener un elemento por su identificador requiere recorrer la tabla	La clave de partición no está presente en el patrón más frecuente	La clave de partición debe aparecer en casi todas las consultas; si no, el modelo está al revés.
Las escrituras de la tabla empiezan a fallar sin motivo aparente	Un índice secundario global agotó su capacidad	Vigila la capacidad de cada índice por separado y reduce índices y proyecciones al mínimo necesario.
El coste de escritura es varias veces el esperado	Cada escritura se replica a todos los índices, con proyección completa	Crea solo los índices que sirvan patrones reales, proyecta lo mínimo y saca del índice lo que deja de ser consultable.
Los informes y las búsquedas son lentos y caros	Se fuerzan consultas ad hoc y agregaciones sobre DynamoDB	Replica por el flujo de cambios a un índice de búsqueda y a un almacén analítico, y deja aquí solo el camino operativo.

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 en DynamoDB el modelo se deduce de los patrones y no al revés?
2. ¿Qué cuatro propiedades tiene una buena clave de partición?
3. ¿Por qué puede haber estrangulamiento con la tabla al 5 % de uso?
4. ¿Qué le ocurre a las escrituras de la tabla si se satura un índice global?
5. ¿En qué casos conviene sacar un patrón fuera de DynamoDB?

Referencias

- AWS (2025). DynamoDB developer guide: best practices for designing partition keys. <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-partition-key-design.html>
- AWS (2025). Single-table design considerations. <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-general-nosql-design.html>
- DeBrie, A. (2020). The DynamoDB Book. <https://www.dynamodbbook.com/>
- AWS (2025). Secondary indexes and their capacity implications. <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SecondaryIndexes.html>
- AWS (2025). DynamoDB Streams and change data capture. <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 17 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 207 · SAM, Lambda, API Gateway y despliegue serverless	Parte 17 · Programa	209 · Cognito, JWT authorizers, WAF y defensa en profundidad →

Evaluación — 208 DynamoDB por patrones de acceso y single-table design

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 aws-dynamodb-model con diagrama o configuración, evidencia de ejecución, alternativa descartada, riesgo residual y criterio de rollback.

Criterio de aceptación

- [ ] Resultado reproducible por una segunda persona.
- [ ] Evidencia vinculada a cada afirmación técnica.
- [ ] Prueba negativa o de fallo incluida.
- [ ] Costos y permisos expresados con unidades y alcance.
- [ ] Limitaciones declaradas sin presentar la simulación como producción.

Escala

Nivel	Evidencia
A — competente	Cumple todo y defiende trade-offs con datos.
B — en desarrollo	Cumple lo esencial; falta profundidad en una dimensión.
C — inicial	Artefacto parcial o conclusión sin evidencia suficiente.
0	Sin entrega reproducible.

209 — Cognito, JWT authorizers, WAF y defensa en profundidad

Parte: 17 — AWS: arquitectura, automatización y operación en producción Nivel: avanzado · Horas estimadas: 4
Laboratorio: security · Estado: EXECUTABLECORE

Propósito

Resolver la identidad de los usuarios finales y la defensa del perímetro público sin construir nada propio que haya que mantener. La clase cubre Cognito y los testigos que emite, cómo se validan bien —que es donde están los fallos de seguridad reales—, cuándo conviene un autorizador propio y cuándo no, y el filtrado de aplicación con su parte incómoda: las reglas gestionadas bloquean tráfico legítimo, y desplegarlas en modo bloqueo sin medir antes rompe el negocio.

Resultados de aprendizaje

Al finalizar podrás:

1. Distinguir los tres testigos de Cognito y usar cada uno donde corresponde.
2. Validar un testigo correctamente, comprobando todo lo que hay que comprobar.
3. Elegir entre autorizador integrado y propio con criterio.
4. Desplegar filtrado de aplicación sin bloquear tráfico legítimo.
5. Componer las capas de defensa sabiendo qué cubre cada una.

Conceptos centrales

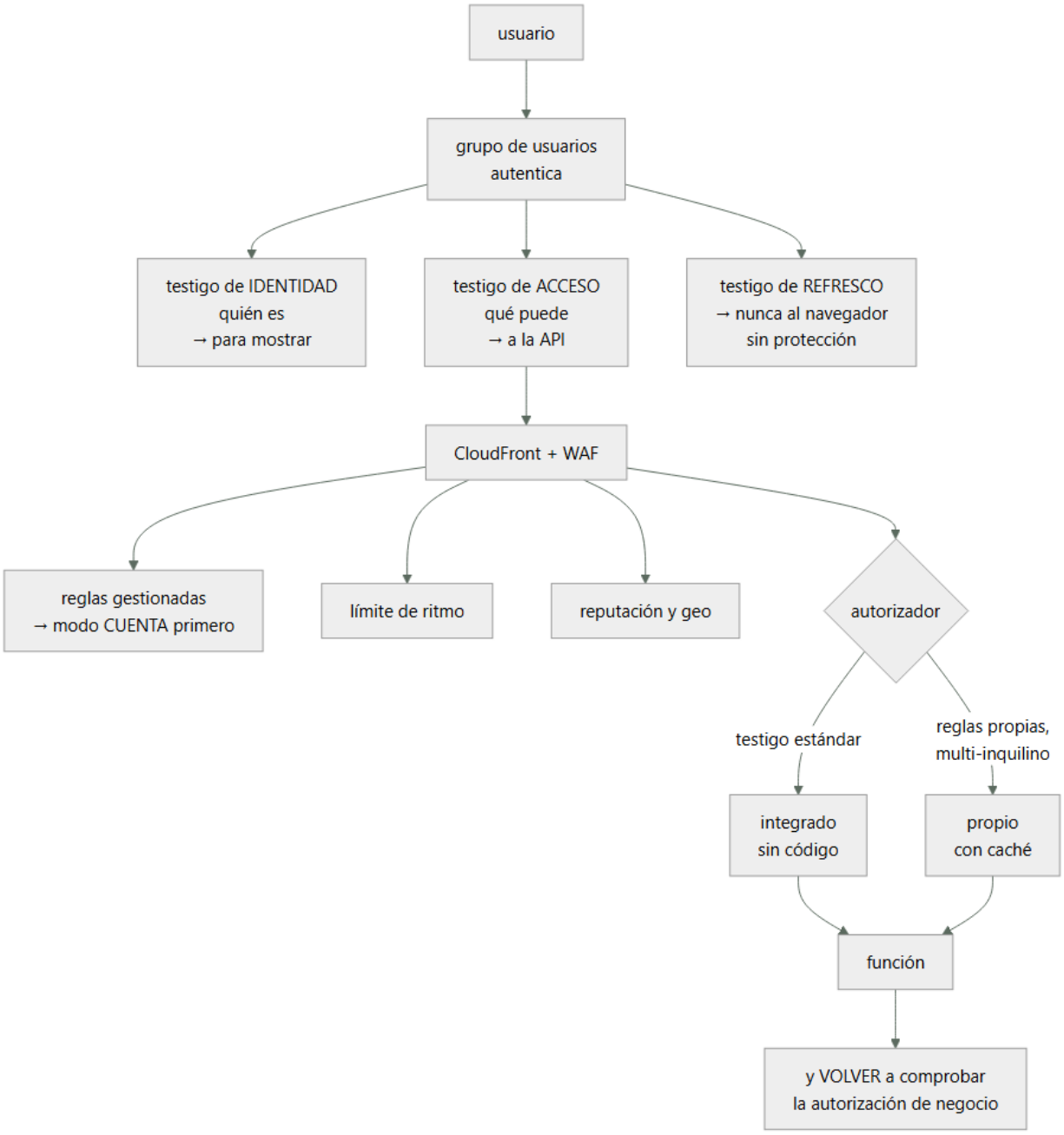
Concepto	Comprensión verificable
grupo de usuarios	Directorio de usuarios finales que autentica y emite testigos. Es el proveedor de identidad.
grupo de identidades	Componente que canjea un testigo por credenciales temporales de AWS. Cosa distinta del anterior.
testigo de identidad	Contiene quién es el usuario. Sirve para mostrar datos, no para autorizar llamadas.
testigo de acceso	Contiene qué puede hacer, con sus ámbitos. Es el que se envía a la API.
autorizador	Componente de la pasarela que valida el testigo antes de invocar la función.
modo cuenta	Despliegue de una regla de filtrado que registra lo que bloquearía sin bloquearlo.

Modelo mental

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

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. Los tres testigos, y cuál va a dónde

Cognito emite tres testigos y confundirlos es la causa de la mitad de los problemas.

```
TESTIGO DE IDENTIDAD
  dice QUIÉN es el usuario: nombre, correo, atributos
  destinado a la aplicación cliente, para mostrar datos
  ✗ NO se usa para autorizar llamadas a la API
  → su audiencia es el cliente, no la API

TESTIGO DE ACCESO
  dice QUÉ puede hacer: ámbitos, grupos, cliente
  es el que se envía en la cabecera de autorización
  ✓ este es el que valida la API

TESTIGO DE REFRESCO
  sirve para obtener testigos nuevos sin volver a
  autenticarse
  vida larga (días o meses)
```

```
  ✗ NUNCA en almacenamiento accesible por scripts del
    navegador
  → cookie con marca de solo servidor, o almacenamiento
    seguro del dispositivo
```

Y el error de diseño más frecuente:

```
enviar el testigo de identidad a la API porque «trae el
correo del usuario»
→ la API acaba autorizando con un testigo que no era para
  ella
→ y si la aplicación cliente cambia, la API se rompe

si la API necesita el correo, se añade como ámbito o se
busca por el identificador del sujeto
```

El grupo de identidades, que es otra cosa y se confunde con el anterior:

```
GRUPO DE USUARIOS      autentica personas y emite testigos
GRUPO DE IDENTIDADES   canjea un testigo por credenciales
                       temporales de AWS

→ el segundo solo hace falta si el cliente debe llamar
  DIRECTAMENTE a servicios de AWS (subir a un bucket, por
  ejemplo)
→ y entonces el rol que recibe debe estar restringido al
  prefijo del propio usuario, no al bucket entero
                                                clase 134
```

Y una advertencia sobre el acceso directo desde el cliente:

```
dar credenciales de AWS al navegador amplía mucho el alcance
→ si es evitable, mejor una URL prefirmada emitida por la
  API, con vida corta y ámbito exacto
```

2. Validar bien un testigo

Aquí están los fallos de seguridad reales, y son siempre los mismos.

```
LO QUE HAY QUE COMPROBAR, SIN SALTARSE NADA

1  LA FIRMA, contra las claves públicas del emisor
   ✗ el fallo grave: decodificar sin verificar
     → cualquiera puede fabricar un testigo

2  EL ALGORITMO ESPERADO
   ✗ aceptar el algoritmo que diga el testigo
     → «ninguno» o cambio a algoritmo simétrico con la
       clave pública como secreto

3  EL EMISOR
   ✗ no comprobarlo → vale un testigo de otro directorio

4  LA AUDIENCIA (o el cliente)
   ✗ no comprobarla → vale un testigo emitido para otra
     aplicación del mismo directorio

5  EL USO DEL TESTIGO
   comprobar que es de acceso y no de identidad

6  LA CADUCIDAD, con margen de reloj pequeño

7  LOS ÁMBITOS o grupos que exige esta operación
```

Y el detalle operativo que rompe en producción:

```
LAS CLAVES PÚBLICAS SE CACHEAN
  descargarlas en cada petición añade latencia y depende
  del emisor
  cachearlas para siempre rompe cuando rotan
  → cachear con caducidad y refrescar si aparece un
    identificador de clave desconocido
```

El autorizador de la pasarela, con la elección:

```
INTEGRADO (validación de testigo estándar)
  la pasarela valida firma, emisor, audiencia y ámbitos
```

```
  sin código propio
  + nada que mantener, sin arranque en frío, gratis
  − no puede decidir con lógica de negocio

PROPIO (función autorizadora)
  código que devuelve permitido o denegado, y contexto
  + reglas propias: multi-inquilino, listas, permisos finos
  − una función más en el camino crítico
  − CACHÉ obligatoria, o se ejecuta en cada petición

  y el detalle de la caché
    la clave de caché debe incluir lo que hace variar la
    decisión
    → si solo cachea por el testigo y la decisión depende
      del recurso, autoriza de más   ← fallo de seguridad
```

Y la regla que este programa mantiene:

```
el autorizador comprueba QUE EL TESTIGO ES VÁLIDO
la función vuelve a comprobar QUE ESTE USUARIO PUEDE HACER
ESTO CON ESTE RECURSO

→ un pedido no se lee porque el testigo sea válido, sino
  porque el pedido es de ese cliente
→ y esa comprobación no la puede hacer la pasarela
```

Y el fallo que resulta de olvidarlo:

```
referencia directa insegura
  GET /pedidos/4471 con un testigo válido de otro cliente
  → si la función no comprueba la propiedad, devuelve el
    pedido ajeno
→ es de los hallazgos más frecuentes en revisiones
                                                clase 189
```

3. Filtrado de aplicación sin romper el negocio

El filtrado de aplicación se despliega mal casi siempre, y la forma de hacerlo bien es lenta a propósito.

```
LO QUE APORTA
  reglas gestionadas contra ataques conocidos
  límite de ritmo por dirección o por identificador
  reputación de origen y bloqueo por geografía
  reglas propias por ruta y por cabecera

LO QUE HAY QUE SABER ANTES
  las reglas gestionadas producen FALSOS POSITIVOS
  → bloquean peticiones legítimas con contenido que parece
    un ataque
  → un campo de texto libre con comillas, una descripción
    con SQL de ejemplo, un cuerpo grande
```

Y por eso el despliegue tiene un orden obligatorio:

```
1  DESPLEGAR EN MODO CUENTA
   registra lo que bloquearía, sin bloquear

2  MEDIR 2-4 SEMANAS
   ¿cuántas peticiones bloquearía? ¿cuáles son legítimas?

3  AJUSTAR
   excluir reglas concretas en rutas concretas
   excluir campos del cuerpo que dan falsos positivos
   → NO desactivar el grupo entero

4  ACTIVAR EL BLOQUEO por grupos de reglas, no todo a la vez

5  VIGILAR los bloqueos y revisarlos
   → cada bloqueo legítimo es un ajuste pendiente

→ activar en bloqueo el primer día es como cerrar la salida
  sin registrar antes                          clase 200
```

Y el límite de ritmo, que es la regla que más aporta y la más fácil:

```
por dirección de origen: protege de lo básico
```

```
por identificador de usuario o de sesión: protege mejor
por ruta: el registro y el acceso llevan límites más
  estrictos que el catálogo

y hay que decidir qué se devuelve al bloquear
  429 con cabecera de reintento es honesto y ayuda al
  cliente legítimo
```

Dónde poner el filtrado, que tiene consecuencias:

```
en CloudFront    filtra en el borde, antes de la pasarela
                 → más barato: lo bloqueado no llega
                 → y protege también el contenido estático
en la pasarela   filtra ahí; si hay CloudFront delante, se
                 duplica
en el balanceador para arquitecturas con contenedores
                                              clase 212

→ con CloudFront delante, ponerlo ahí y restringir la
  pasarela para que SOLO acepte tráfico de esa distribución
  → si no, se puede saltar el filtro llamando a la pasarela
    directamente                              clase 205
```

4. Capas, y qué cubre cada una

La defensa en profundidad se cita mucho y se comprueba poco. Lo útil es saber qué cubre cada capa y qué no.

CAPA	CUBRE	NO CUBRE
filtrado en el borde	ataques conocidos, volumen, geografía	lógica de negocio
autorizador	testigo inválido, caducado, de otro emisor	si ESTE usuario puede ver ESTE recurso
función	propiedad del recurso, reglas de negocio	testigo robado
permisos de la función	alcance del código si es comprometido	errores de lógica
datos	cifrado, perímetro	acceso legítimo mal usado

Y las comprobaciones que corresponden a esta clase:

- llamar a la API con un testigo de identidad en vez de acceso → debe fallar
- llamar con un testigo de otra aplicación del mismo directorio → debe fallar
- llamar con un testigo caducado → debe fallar
- llamar con la firma alterada → debe fallar
- llamar con algoritmo «ninguno» → debe fallar
- pedir el pedido de otro cliente con testigo válido → debe fallar
- llamar a la pasarela saltándose la distribución → debe fallar
- superar el límite de ritmo → 429, no 500
- enviar una carga de ataque conocida → bloqueada
- enviar texto legítimo con comillas → NO bloqueada

Y la última es tan importante como las demás: una defensa que bloquea a los clientes es un incidente.

Lo que hay que vigilar:

```
autorizaciones denegadas, por motivo
  → un pico de «firma inválida» es alguien probando
  → un pico de «caducado» suele ser un fallo de refresco
    en el cliente
bloqueos del filtrado, por regla
límite de ritmo alcanzado, por ruta
intentos de autenticación fallidos por usuario
y el uso de testigos de refresco
```

Y dos decisiones operativas que conviene tomar pronto:

```
DURACIÓN DE LOS TESTIGOS
  acceso   corta (15-60 min): limita el daño de uno robado
  refresco larga, pero REVOCABLE
  → y hay que tener probado cómo se revoca a un usuario
    concreto                                        ley 22

QUÉ PASA CUANDO EL DIRECTORIO NO RESPONDE
  los testigos ya emitidos siguen valiendo hasta caducar
  → la validación no llama al directorio en cada petición
  → pero no se pueden emitir nuevos: los usuarios ya dentro
    siguen; los que entran, no
  → conviene saberlo antes de calcular el techo de
    disponibilidad                                clase 185
```

Y la lista de comprobación de la clase:

- la API valida el testigo de ACCESO, no el de identidad
- se comprueban firma, algoritmo, emisor, audiencia, uso y caducidad
- las claves públicas se cachean con caducidad y se refrescan ante clave desconocida
- el testigo de refresco no es accesible desde scripts
- si hay autorizador propio, su caché incluye lo que hace variar la decisión
- la función comprueba la propiedad del recurso
- el filtrado se desplegó en modo cuenta antes de bloquear
- hay límite de ritmo por ruta y por usuario
- la pasarela solo acepta tráfico de la distribución
- la revocación de un usuario está probada
- las diez pruebas negativas se han ejecutado

Y el cierre que enlaza con la clase siguiente: con la entrada protegida, el trabajo que no puede hacerse en la petición se envía a procesar más tarde, y ahí aparecen los duplicados, los mensajes fallidos y el reproceso. Bus de eventos, colas, cola de fallidos e idempotencia es la materia de la clase 210.

Ejemplo trabajado

CloudShop protege su API pública. Lo que sigue son los tres fallos que encontró la revisión de seguridad, el despliegue del filtrado que empezó rompiendo el registro de usuarios, y el estado final con sus pruebas negativas.

Fallo 1 · La API validaba el testigo equivocado.

```
el cliente enviaba el testigo de IDENTIDAD
el autorizador estaba configurado para aceptarlo
razón   «trae el correo, que la API necesita»

lo que eso permitía
  el testigo de identidad no lleva ámbitos
  → la API no podía distinguir un usuario normal de uno
    con permisos de administración
  → la distinción se hacía leyendo un atributo del testigo
    que el propio usuario podía modificar desde su perfil

  la prueba negativa
    un usuario cambió su atributo «rol» a «admin» desde la
    pantalla de perfil
    → obtuvo acceso al panel de administración
    tiempo desde la idea hasta conseguirlo: 4 minutos

corrección
  la API valida el testigo de ACCESO
  los permisos vienen de GRUPOS del directorio, no de
  atributos editables por el usuario
  el atributo «rol» se eliminó de los editables
  y el correo se obtiene por el identificador del sujeto
```

Fallo 2 · La función no comprobaba la propiedad.

```
GET /pedidos/{id}
  el autorizador validaba el testigo    ✓
  la función devolvía el pedido          ✗ sin comprobar de
                                           quién era

la prueba
```

```
  un usuario con sesión válida pidió /pedidos/4471
  → recibió el pedido de otro cliente, con dirección,
    teléfono e importe

y los identificadores eran secuenciales
  → se podían recorrer todos                    clase 189

corrección
  la función comprueba que el pedido pertenece al sujeto
  del testigo
  → y como el modelo tiene el pedido bajo CLIENTE#, la
    comprobación es la propia clave de la consulta
                                                clase 208
  identificadores cambiados a opacos
  y la comprobación añadida como prueba negativa permanente
```

Fallo 3 · Se podía saltar el filtrado.

```
el filtrado estaba en CloudFront
la pasarela aceptaba peticiones de cualquier origen
→ llamando a la URL de la pasarela directamente se
  saltaba el filtrado, el límite de ritmo y los registros

se comprobó en los registros de acceso
  peticiones directas a la pasarela              8.400/día
  de ellas, de rastreadores y escáneres          7.900
  de ellas, de un cliente móvil antiguo mal
    configurado                                    500

corrección
  la pasarela exige una cabecera secreta que solo añade la
  distribución, y rechaza el resto
  el cliente móvil antiguo se corrigió en la versión
  siguiente, con un plazo de 8 semanas para la retirada
                                                clase 188
```

El despliegue del filtrado, que empezó mal.

```
primer intento, febrero
  se activaron 4 grupos de reglas gestionadas en modo
  BLOQUEO, el mismo día

  a las 3 horas
    el registro de usuarios nuevos había caído un 34 %
    nadie relacionó las dos cosas

  a los 2 días, atención al cliente reportó que había
  usuarios que no podían registrarse

  causa
    una regla gestionada bloqueaba cuerpos con ciertos
    patrones
    los apellidos con apóstrofo —O'Brien, D'Angelo— y las
    direcciones con comillas coincidían
    → 1 de cada 3 registros con esos caracteres se
      bloqueaba, devolviendo un 403 sin explicación

  pérdida estimada                      2 días × 34 %
                                        de registros

segundo intento, marzo, con el orden correcto
  1  los 4 grupos en MODO CUENTA
  2  medición de 3 semanas
       peticiones que se bloquearían            41.200
       claramente maliciosas                    38.900
       LEGÍTIMAS                                 2.300
         · apellidos y direcciones con apóstrofo
         · descripciones de producto de proveedores con
           fragmentos de HTML
         · un cliente que subía ficheros grandes por la API
  3  ajustes
       exclusión de la regla concreta en el campo de
         apellido y dirección, en la ruta de registro
       exclusión en el campo de descripción de la ruta de
```

```
        catálogo interno
      límite de tamaño ajustado en la ruta de subida
      → NO se desactivó ningún grupo entero
4  bloqueo activado grupo a grupo, con una semana entre
   cada uno
5  revisión semanal de bloqueos

resultado tras 6 meses
  peticiones bloqueadas                    36.400/mes
  bloqueos legítimos reportados                   4
  → los 4, ajustados en menos de un día
```

El límite de ritmo, por ruta:

```
ruta                       límite            motivo
/auth/login                5 / 5 min / IP    fuerza bruta
/auth/registro             3 / hora / IP     cuentas falsas
/api/pedidos (POST)        20 / min /usuario abuso
/api/catalogo              600 / min / IP    tráfico normal
resto                      2.000 / 5 min /IP suelo general

respuesta al bloquear   429 con cabecera de reintento
```

Las diez pruebas negativas, ejecutadas:

```
✓  testigo de identidad en vez de acceso        rechazado
✓  testigo de otra aplicación del directorio    rechazado
✓  testigo caducado                             rechazado
✓  firma alterada                               rechazado
✓  algoritmo «ninguno»                          rechazado
✓  pedido de otro cliente con testigo válido    rechazado
✓  llamada directa a la pasarela                rechazada
✓  superar el límite de ritmo                   429 correcto
✓  carga de ataque conocida                     bloqueada
✓  apellido con apóstrofo                       NO bloqueado
```

Y una prueba adicional que se añadió tras el incidente:

```
✓  registrar un usuario llamado O'Brien, con dirección
   «Rúa d'Ouro, 3», y una descripción con <b>negrita</b>
   → los tres pasan
   → se ejecuta en cada despliegue del filtrado
```

El resultado:

```
escalada de privilegios por atributo         posible   imposible
lectura de pedidos ajenos                    posible   imposible
peticiones que saltan el filtrado          8.400/día         0
registros bloqueados por falso positivo      34 %       0,01 %
intentos de fuerza bruta que llegan a
  la función                                 todos            0
coste del filtrado                             —       210 €/mes
coste de cómputo evitado por bloquear
  en el borde                                  —      -390 €/mes
```

La lección que esta clase deja: los tres fallos de seguridad no estaban en la configuración del filtrado ni del directorio: estaban en usar el testigo equivocado, en no comprobar de quién era el recurso y en dejar un camino que se saltaba todo. Y el incidente más caro del capítulo lo causó la propia defensa: activar las reglas gestionadas en modo bloqueo el primer día tumbó un tercio de los registros durante dos días, y a nadie se le ocurrió relacionarlo hasta que llamó un cliente apellidado O'Brien.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-17-aws-production-architecture/209-cognito-jwt-authorizers-waf-y-defensa-en-profundidad/lab.py
```

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

1. Lee exercise.steps y formula una predicción antes de ejecutar.

2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.
4. Materializa aws-identity-edge en evidence/ usando la plantilla indicada.
5. Para proveedor real, sigue sandbox.requires, registra costo y ejecuta destroy.

Evidencia esperada

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

Reto verificable

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

■ Criterio de aceptación

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

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
Un usuario se concede permisos de administración editando su perfil	La autorización se basa en un atributo del testigo de identidad que el usuario puede modificar	Valida el testigo de acceso y deriva los permisos de grupos del directorio, no de atributos editables.
Un usuario válido lee recursos de otro	El autorizador valida el testigo pero la función no comprueba la propiedad del recurso	Comprueba en la función que el recurso pertenece al sujeto del testigo, y usa identificadores opacos.
Se puede evitar el filtrado llamando a la pasarela directamente	La pasarela acepta tráfico de cualquier origen	Exige una cabecera secreta que solo añade la distribución y rechaza el resto.
Cae el registro de usuarios tras activar el filtrado	Reglas gestionadas en modo bloqueo desde el primer día, con falsos positivos	Despliega en modo cuenta, mide semanas, excluye reglas concretas en campos concretos y activa el bloqueo por grupos.
Un autorizador propio autoriza de más	La clave de caché no incluye lo que hace variar la decisión	Incluye en la clave de caché el recurso y cuanto influya en el resultado, o desactiva la caché.
El cliente pierde la sesión cada poco o la mantiene demasiado	Duraciones de testigo mal elegidas y refresco mal guardado	Testigo de acceso corto, refresco largo pero revocable y guardado fuera del alcance de scripts, con revocación probada.

■ Seguridad, ética y costo

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

■ Preguntas de comprobación

1. ¿Qué diferencia hay entre el testigo de identidad y el de acceso, y cuál va a la API?
2. ¿Qué siete comprobaciones exige validar bien un testigo?
3. ¿Qué comprueba el autorizador y qué debe volver a comprobar la función?
4. ¿Por qué no se despliega el filtrado en modo bloqueo el primer día?
5. ¿Qué ocurre con los testigos ya emitidos si el directorio deja de responder?

Referencias

- AWS (2025). Amazon Cognito user pools: using tokens. <https://docs.aws.amazon.com/cognito/latest/developerguide/amazon-cognito-user-pools-using-tokens-with-identity-providers.html>
- AWS (2025). Verifying a JSON Web Token. <https://docs.aws.amazon.com/cognito/latest/developerguide/amazon-cognito-user-pools-using-tokens-verifying-a-jwt.html>
- AWS (2025). AWS WAF: testing rules in count mode. <https://docs.aws.amazon.com/waf/latest/developerguide/web-acl-testing.html>
- OWASP (2025). API Security Top 10: broken object level authorization. <https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/>
- RFC 8725 — JSON Web Token best current practices. <https://www.rfc-editor.org/rfc/rfc8725>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 17 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 208 · DynamoDB por patrones de acceso y single-table design	Parte 17 · Programa	210 · EventBridge, SQS, DLQ, replay e idempotencia →

Evaluación — 209 Cognito, JWT authorizers, WAF y defensa en profundidad

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 aws-identity-edge 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.

210 — EventBridge, SQS, DLQ, replay e idempotencia

Parte: 17 — AWS: arquitectura, automatización y operación en producción Nivel: avanzado · Horas estimadas: 4
Laboratorio: messaging · Estado: EXECUTABLECORE

Propósito

Procesar trabajo fuera de la petición sin perder mensajes, sin duplicar efectos y sin quedarse con una cola de fallidos que nadie mira. La clase distingue el bus de eventos de la cola por lo que cada uno resuelve, fija los parámetros que de verdad importan —visibilidad, reintentos, lotes—, y desarrolla las dos cosas que este programa lleva señalando desde la parte 09: la idempotencia no es opcional y la cola de fallidos sin alerta ni procedimiento de reproceso es una pérdida de datos en diferido.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir entre bus de eventos, cola y flujo según lo que haga falta.
2. Configurar visibilidad, reintentos y lotes de forma coherente.
3. Implementar idempotencia con clave y escritura condicionada.
4. Operar la cola de fallidos con alerta, diagnóstico y reproceso.
5. Evitar los tres fallos que arruinan estos sistemas.

Conceptos centrales

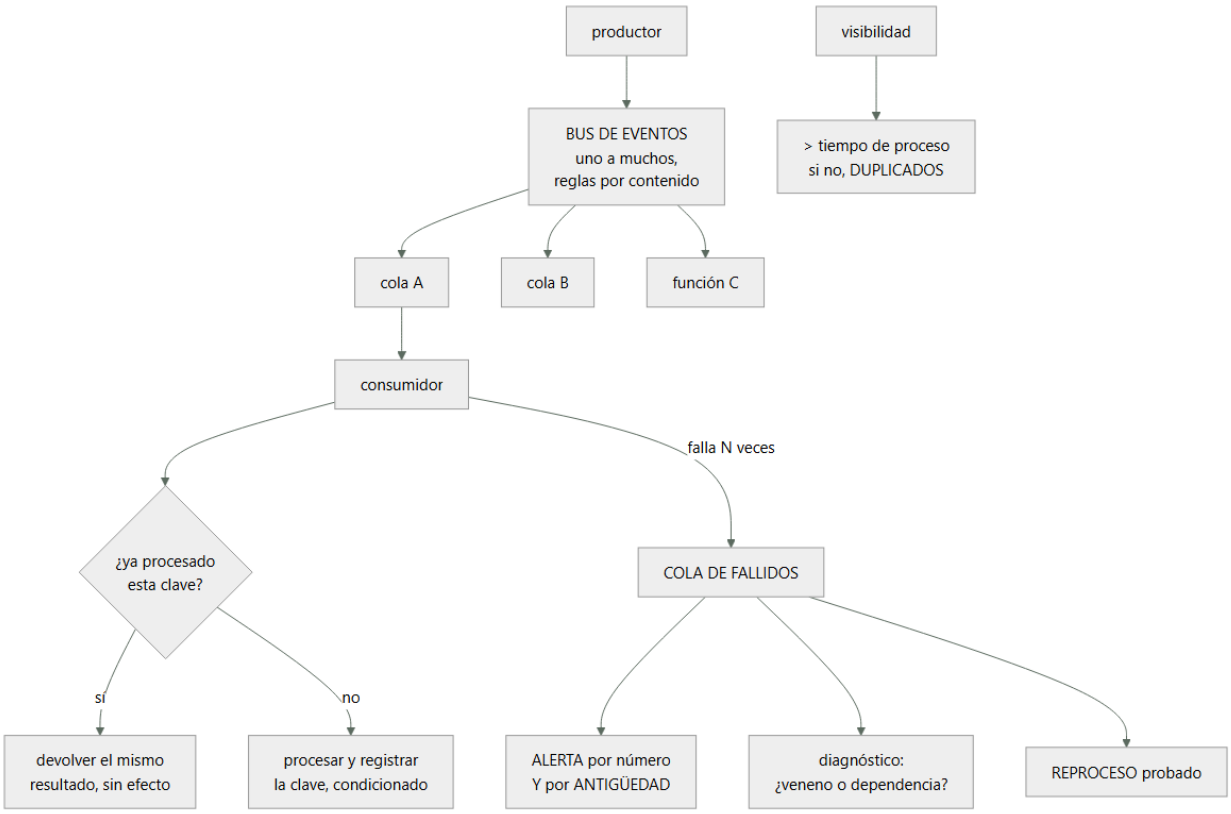
Concepto	Comprensión verificable
bus de eventos	Encaminador que entrega un evento a varios destinos según reglas de contenido. Uno a muchos.
cola	Almacén de mensajes con un consumidor lógico, entrega al menos una vez y control de visibilidad.
tiempo de visibilidad	Plazo durante el que un mensaje tomado no se entrega a otro consumidor. Debe superar el tiempo de proceso.
cola de fallidos	Destino de los mensajes que agotaron sus reintentos. Sin alerta ni reproceso, es una papelera.
idempotencia	Que procesar el mismo mensaje dos veces produzca el mismo resultado que procesarlo una.
fallo parcial de lote	Informar de qué elementos del lote fallaron para que solo esos se reintenten.

Modelo mental

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

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. Bus, cola y flujo: qué resuelve cada uno

Los tres transportan mensajes y no son intercambiables.

```
BUS DE EVENTOS (EventBridge)
  encamina un evento a VARIOS destinos según reglas sobre
  su contenido
  + desacopla al productor de sus consumidores: añadir un
    consumidor no toca al productor              clase 148
  + reglas declarativas, archivo y reproducción
  + integración con eventos del propio proveedor
  – no garantiza orden
  – sin control fino de reintentos por consumidor

COLA (SQS)
  almacena mensajes para UN consumidor lógico
  + control de visibilidad, reintentos y cola de fallidos
  + amortigua picos: el consumidor va a su ritmo
  + variante ordenada con agrupación por clave
  – uno a uno; para varios consumidores hacen falta varias
    colas

FLUJO (Kinesis)
  registro ordenado por partición, con varios lectores
  independientes y reproducción por posición
  + orden dentro de la partición y reproceso desde un punto
  – capacidad por fragmento; el reparto se decide con la
    clave                                        clase 116
```

Y el patrón que combina bien y es el habitual:

```
productor → BUS → regla → COLA → función

por qué la cola en medio y no la función directamente
  amortigua picos: la función consume a su ritmo
  da reintentos controlados y cola de fallidos
  y protege a la base de la concurrencia         clase 207

→ conectar el bus directamente a la función pierde las tres
```

Y la decisión sobre el contenido del mensaje:

```
HECHO CON DATOS   «pedido confirmado» + los datos
  + el consumidor no necesita llamar de vuelta
  − el mensaje envejece; hay que versionarlo   clase 188

SOLO REFERENCIA   «pedido 4471 confirmado»
  + mensaje pequeño y siempre válido
  − cada consumidor llama al productor: N llamadas más
  − y si el productor está caído, no se puede procesar

→ en general, HECHO CON LOS DATOS que el consumidor
  necesita, versionado y con esquema validado en el
  productor                                    clase 188
```

2. Los parámetros que importan

Los valores por defecto de una cola producen duplicados y pérdidas. Estos son los que hay que decidir.

```
TIEMPO DE VISIBILIDAD
  cuánto tiempo un mensaje tomado queda invisible

  REGLA   visibilidad > tiempo máximo de proceso
          típicamente 6 veces el plazo de la función

  si es MENOR que el proceso
    el mensaje reaparece mientras aún se está procesando
    → otro consumidor lo coge
    → DOS ejecuciones del mismo mensaje
    → y el síntoma es «duplicados aleatorios bajo carga»

RECUENTO MÁXIMO DE RECEPCIONES
  cuántas veces se reintenta antes de ir a la cola de
  fallidos
  demasiado bajo   un fallo transitorio pierde el mensaje
  demasiado alto   un mensaje venenoso bloquea durante horas
  típico   3 a 5

TAMAÑO DE LOTE
  cuántos mensajes recibe la función por invocación
  lote grande   más eficiente, menos invocaciones
  lote grande   y un fallo reintenta EL LOTE ENTERO
  → salvo que se informe de fallo parcial

FALLO PARCIAL DE LOTE
  la función devuelve qué identificadores fallaron
  → solo esos se reintentan
  → SIN esto, un mensaje malo en un lote de 10 hace que
    los 9 buenos se reprocesen una y otra vez
  → y si no son idempotentes, se duplican nueve efectos

RETENCIÓN
  cuánto vive un mensaje en la cola: hasta 14 días
  → y en la cola de fallidos conviene el máximo, para tener
    tiempo de reprocesar

ESPERA LARGA
  el consumidor espera si no hay mensajes
  → menos peticiones vacías, menos coste y menos latencia
  → debería estar siempre activada
```

Y una relación que hay que respetar:

```
visibilidad ≥ plazo de la función × tamaño de lote
→ porque la función procesa el lote entero dentro de un
  solo tiempo de visibilidad
```

Y el orden, cuando hace falta:

```
la cola ordenada garantiza orden DENTRO de un grupo
  → el grupo es la clave: cliente, pedido, cuenta
  → y limita el caudal por grupo

y la pregunta previa
  ¿de verdad hace falta orden, o basta con que las
```

```
  operaciones sean conmutativas?              clase 203
  → «−1 unidad» no necesita orden; «stock = 4», sí
```

3. Idempotencia, en concreto

Con entrega al menos una vez, reintentos y lotes, el mismo mensaje se procesará dos veces. No es una posibilidad remota: ocurre.

```
DE DÓNDE VIENEN LOS DUPLICADOS
  visibilidad menor que el proceso
  reintento tras un plazo vencido en el que el trabajo sí
    se hizo
  lote reintegrado entero por un solo elemento fallido
  el productor que reintenta al no recibir confirmación
  y el reproceso desde la cola de fallidos
```

El mecanismo, en tres piezas:

```
1  CLAVE DE IDEMPOTENCIA
   identificador del mensaje, o del suceso de negocio
   → nunca generada por el consumidor
   → y debe ser la misma en el reintento: si el productor
     genera una nueva cada vez, no sirve de nada

2  REGISTRO DE CLAVES PROCESADAS
   tabla con la clave, el resultado y una caducidad
   → la caducidad debe superar la retención de la cola

3  ESCRITURA CONDICIONADA, antes del efecto
   «inserta esta clave si no existe»
   si falla la condición → ya se procesó: devolver el
   resultado guardado
   si tiene éxito → procesar
```

Y el detalle que decide si funciona:

```
¿QUÉ PASA SI EL PROCESO FALLA DESPUÉS DE REGISTRAR LA
CLAVE?
  el mensaje se reintenta, ve la clave y no hace nada
  → el efecto NUNCA se produce

→ por eso el registro se hace en tres estados
    EN CURSO   con caducidad corta
    HECHO      con el resultado
    y si un reintento encuentra EN CURSO caducado, lo
    reclama
```

Y la alternativa que evita todo esto cuando se puede:

```
OPERACIONES NATURALMENTE IDEMPOTENTES
  «poner el estado a ENVIADO» lo es
  «incrementar el contador» no lo es
  «insertar con clave única del negocio» lo es

→ modelar la operación como asignación en vez de como
  incremento resuelve el problema sin infraestructura
                                              clase 149
```

Y la prueba que hay que ejecutar:

```
enviar el mismo mensaje 50 veces en paralelo
→ un solo efecto, 50 respuestas idénticas         ley 22
```

4. La cola de fallidos, operada de verdad

La cola de fallidos se configura y luego se olvida. Este programa la ha visto llena y sin mirar en la clase 120 y en la 132.

```
LO QUE PASA SI NADIE LA MIRA
  los mensajes caducan a los 14 días
  → pérdida de datos silenciosa                    ley 13
  → y cuando se descubre, ya no se pueden recuperar
```

Lo que hace falta, en cuatro piezas:

```
1  ALERTA POR NÚMERO Y POR ANTIGÜEDAD
   «hay mensajes» es la alerta obvia
   «hay un mensaje de más de 1 hora» es la que sirve
```

```
   → la antigüedad detecta el goteo que el número esconde

2  DIAGNÓSTICO: ¿por qué está ahí?
   VENENO       el mensaje es incorrecto y fallará siempre
     → esquema inválido, campo que falta, dato imposible
     → reprocesarlo no sirve; hay que corregir o descartar
   DEPENDENCIA  el mensaje es correcto y algo estaba caído
     → reprocesar funciona
   CÓDIGO       un fallo de la función
     → corregir y reprocesar

   → y para distinguirlos hay que guardar el motivo del
     fallo con el mensaje

3  REPROCESO PROBADO
   un procedimiento —no un comando improvisado— que
     mueve mensajes de la cola de fallidos a la principal
     con límite de ritmo, para no tumbar lo que se
       recuperó
     y con posibilidad de reprocesar uno solo
   → y probado antes de necesitarlo                ley 22

4  DESCARTE EXPLÍCITO
   los mensajes veneno se descartan a propósito, con
   registro de quién y por qué
   → no se dejan caducar
```

Y una advertencia sobre el reproceso masivo:

```
reprocesar 40.000 mensajes de golpe
  → dispara la concurrencia
  → tumba la base                                clase 207
  → y genera una segunda tanda de fallidos

→ reproceso con límite de ritmo, y por lotes
```

Los tres fallos que arruinan estos sistemas, resumidos:

```
1  visibilidad menor que el proceso  → duplicados
2  sin idempotencia                  → duplicados con efecto
3  cola de fallidos sin alerta ni    → pérdida silenciosa
   reproceso
```

Lo que hay que vigilar:

```
antigüedad del mensaje más viejo en la cola  ← la clave
profundidad de la cola y su tendencia
mensajes en la cola de fallidos, y su antigüedad
invocaciones fallidas del consumidor
recepciones repetidas del mismo mensaje
  → señal directa de visibilidad mal puesta
y el retraso entre que ocurre el hecho y se procesa
  → es la medida que le importa al negocio
```

Y la lista de comprobación de la clase:

- ■ el bus se usa para uno a muchos y la cola para amortiguar
- ■ hay cola entre el bus y la función, no conexión directa
- ■ los eventos son hechos con datos, versionados y validados
- ■ la visibilidad supera el proceso máximo por lote
- ■ el recuento de reintentos está decidido, no por defecto
- ■ el fallo parcial de lote está activado
- ■ la espera larga está activada
- ■ toda operación con efecto es idempotente
- ■ la clave de idempotencia la genera el productor
- ■ el registro de claves distingue en curso de hecho
- ■ hay alerta por antigüedad en la cola de fallidos
- ■ se guarda el motivo del fallo con el mensaje
- ■ el procedimiento de reproceso existe y se ha ejecutado
- ■ el reproceso tiene límite de ritmo
- ■ los mensajes veneno se descartan con registro

Y el cierre que enlaza con la clase siguiente: con la API, el almacén, la seguridad y el proceso asíncrono en pie, falta saber qué está pasando cuando algo va mal. Observabilidad en AWS, definida como código, es la materia de la clase 211.

Ejemplo trabajado

CloudShop procesa la confirmación de pedidos de forma asíncrona. Lo que sigue son los duplicados que aparecieron bajo carga, la cola de fallidos con 41.000 mensajes que nadie miraba, y el reproceso que tumbó la base la primera vez.

El montaje inicial:

```
función de crear pedido → bus de eventos
  regla «pedido.confirmado» → función de correo
  regla «pedido.confirmado» → función de inventario
  regla «pedido.confirmado» → función de facturación

todo conectado directamente del bus a las funciones
sin colas en medio
sin idempotencia
cola de fallidos configurada en las funciones asíncronas
```

Problema 1 · Correos duplicados bajo carga.

```
síntoma   en campaña, algunos clientes recibían 2 y hasta 3
          correos de confirmación
          en tráfico normal, nunca

diagnóstico
  la invocación asíncrona desde el bus reintenta 2 veces
  ante error
  bajo carga, la función de correo superaba su plazo por
  la latencia del proveedor de correo
  → el envío SÍ se había hecho; el plazo venció al esperar
    la respuesta
  → el bus reintentaba, y se enviaba otra vez

  correos duplicados en la campaña                  1.840
  quejas recibidas                                     61

corrección
  1  cola entre el bus y cada función
     → amortigua, y da control de reintentos propio
  2  idempotencia con el identificador del evento
  3  el plazo de la función subido a 3 × el p99 del
     proveedor, y el envío marcado como hecho ANTES de
     esperar la confirmación final
```

Problema 2 · Duplicados aleatorios en inventario.

```
síntoma   tras añadir las colas, aparecieron descuentos de
          stock duplicados, sin patrón claro

diagnóstico
  tiempo de visibilidad de la cola          30 s (por defecto)
  plazo de la función                       25 s
  tamaño de lote                            10 mensajes
  tiempo real de proceso del lote           hasta 90 s

  → el lote tardaba más que la visibilidad
  → los mensajes reaparecían mientras se procesaban
  → otro consumidor los tomaba

  la relación que se había incumplido
    visibilidad ≥ plazo × tamaño de lote
    30 s  <  25 s × 10

corrección
  visibilidad                              300 s
  tamaño de lote                            10
  plazo de la función                       25 s
  y fallo parcial de lote activado

  y la señal que lo habría detectado antes
    «recepciones repetidas del mismo mensaje»
    → estaba disponible y nadie la miraba          ley 15
```

Y el efecto del fallo parcial de lote:

```
antes   1 mensaje malo en un lote de 10
        → el lote entero se reintentaba
        → 9 mensajes buenos reprocesados hasta 5 veces
        → 45 efectos duplicados por cada mensaje malo
después → solo el malo se reintenta
```

Problema 3 · La cola de fallidos con 41.000 mensajes.

```
se descubrió al revisar el coste de almacenamiento de colas

  mensajes en la cola de fallidos de facturación     41.200
  el más antiguo                                  13 días
  → 14 días es la retención: a la mañana siguiente
    empezaban a caducar

  alerta configurada   «número de mensajes > 0»
  → sí existía
  → salía a un canal creado en 2023, sin suscriptores
                                          ley 15, clase 194

  qué eran los 41.200
    38.900  fallos de un periodo de 4 horas en que el
            servicio de facturación estuvo caído
            → mensajes CORRECTOS, reprocesables
     2.180  mensajes con un campo nuevo que la función
            antigua no entendía
            → veneno por versión de esquema      clase 188
       120  mensajes con importe negativo, de un error de
            un integrador
            → veneno real
```

El reproceso, que salió mal la primera vez.

```
primer intento
  se movieron los 41.200 mensajes a la cola principal de
  golpe

  a los 40 segundos
    la función de facturación escaló a 2.800 ejecuciones
    la base de datos agotó conexiones           clase 207
    → cayó facturación Y el resto de servicios que usan
      esa base
    → y 9.400 mensajes volvieron a la cola de fallidos

  duración del incidente                          38 min

segundo intento, con procedimiento
  concurrencia reservada de la función           40
  reproceso por lotes de 500, con espera entre lotes
  ritmo efectivo                              ~120 mensajes/s
  duración                                         6 min
  fallos                                              0

y los venenos
  2.180 de esquema  → la función se actualizó primero para
                      tolerar el campo nuevo, y luego se
                      reprocesaron
    120 de importe  → descartados con registro de quién y
                      por qué; y se avisó al integrador
```

Lo que quedó montado.

```
ARQUITECTURA
  productor → bus → regla → COLA → función
  una cola por consumidor, con su cola de fallidos

PARÁMETROS, por consumidor
  consumidor    visibilidad  lote  reintentos  plazo
  correo           180 s      5        3        30 s
  inventario       300 s     10        5        25 s
  facturación      600 s     10        5        50 s
  todos con espera larga y fallo parcial activados

IDEMPOTENCIA
  tabla de claves procesadas, con estados EN CURSO y HECHO
```

```
  clave = identificador del evento, generado por el
    productor
  caducidad de la clave: 21 días (> 14 de retención)
  escritura condicionada antes del efecto

  prueba negativa
    mismo mensaje 50 veces en paralelo → 1 efecto      ✓

COLA DE FALLIDOS
  retención al máximo: 14 días
  motivo del fallo guardado como atributo del mensaje
  ALERTAS
    «hay mensajes con más de 1 hora»    → canal de guardia
    «el más antiguo supera 7 días»      → escalado
    → la de antigüedad es la que sirve; la de número se
      dispara con un solo mensaje transitorio
  panel con desglose por motivo

  PROCEDIMIENTO DE REPROCESO
    escrito, con límite de ritmo y por lotes
    con opción de reprocesar un mensaje concreto
    ejecutado por alguien que no lo escribió       ley 22
    → en el primer ensayo, 3 de 9 pasos necesitaron
      aclaración
    tiempo de reproceso de 10.000 mensajes         2 min
```

El resultado, seis meses después:

```
correos duplicados por campaña             1.840           0
descuentos de stock duplicados               340           0
mensajes en cola de fallidos           41.200 (max)   máx. 92
antigüedad máxima en cola de fallidos     13 días      41 min
mensajes caducados y perdidos          casi 41.200         0
reprocesos ejecutados                          1           7
  que causaron incidente                       1           0
mensajes descartados con registro              0         214
```

La lección que esta clase deja: los duplicados no eran aleatorios: eran visibilidad menor que el proceso del lote, una relación aritmética que se puede comprobar antes de desplegar. La cola de fallidos tenía cuarenta y un mil mensajes correctos a un día de caducar, con una alerta que existía y salía a un canal sin nadie. Y el reproceso, hecho sin procedimiento, tumbó la base y generó nueve mil cuatrocientos fallidos nuevos: la recuperación se convirtió en el segundo incidente.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-17-aws-production-architecture/210-eventbridge-sqs-dlq-replay-e-idempotencia/lab.
py
```

El laboratorio selecciona el motor de práctica messaging 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 aws-event-platform 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 flujo que declara orden, entrega y manejo de errores. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye aws-event-platform 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
Aparecen duplicados solo bajo carga	El tiempo de visibilidad es menor que el proceso del lote y el mensaje reaparece mientras se procesa	Fija visibilidad mayor que plazo por tamaño de lote y vigila las recepciones repetidas del mismo mensaje.
Un mensaje incorrecto provoca decenas de efectos duplicados	El lote entero se reintegra por un solo elemento fallido	Activa el informe de fallo parcial de lote para que solo se reintenten los elementos que fallaron.
Se pierden mensajes sin que nadie se entere	La cola de fallidos caduca a los 14 días y su alerta va a un canal sin nadie	Alerta por antigüedad además de por número, dirígela a un canal con guardia y comprueba que llega.
El reproceso masivo provoca un segundo incidente	Se devolvieron todos los mensajes de golpe y la concurrencia tumbó la base	Reprocesa por lotes con límite de ritmo y concurrencia reservada, siguiendo un procedimiento probado.
Reprocesar no arregla nada para parte de los mensajes	Son mensajes veneno: esquema inválido o datos imposibles	Guarda el motivo del fallo, distingue veneno de dependencia, corrige el consumidor antes de reprocesar y descarta lo irreparable con registro.
El efecto nunca llega a producirse pese a los reintentos	Se registró la clave de idempotencia y el proceso falló después	Registra en dos estados, en curso con caducidad y hecho con resultado, y permite reclamar los en curso caducados.

■ 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 resuelve el bus que no resuelve la cola, y por qué conviene poner una cola en medio?
2. ¿Qué relación deben cumplir visibilidad, plazo y tamaño de lote?
3. ¿Qué aporta informar del fallo parcial de un lote?
4. ¿Por qué la alerta por antigüedad es mejor que la de número en la cola de fallidos?
5. ¿Cómo se distingue un mensaje veneno de uno que falló por una dependencia caída?

Referencias

- AWS (2025). Amazon SQS: visibility timeout and dead-letter queues. <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-visibility-timeout.html>
- AWS (2025). Lambda event source mapping: partial batch responses. <https://docs.aws.amazon.com/lambda/latest/dg/services-sqs-errorhandling.html>
- AWS (2025). Amazon EventBridge: rules, archives and replay. <https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-what-is.html>
- AWS (2025). Powertools for AWS Lambda: idempotency. <https://docs.powertools.aws.dev/lambda/python/latest/utilities/idempotency/>
- Hohpe, G. y Woolf, B. (2003). Enterprise Integration Patterns — canal de mensajes inválidos y reintentos. <https://www.enterpriseintegrationpatterns.com/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 17 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 209 · Cognito, JWT authorizers, WAF y defensa en profundidad	Parte 17 · Programa	211 · CloudWatch, X-Ray y observabilidad como código →

Evaluación — 210 EventBridge, SQS, DLQ, replay e idempotencia

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 aws-event-platform 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.

211 — CloudWatch, X-Ray y observabilidad como código

Parte: 17 — AWS: arquitectura, automatización y operación en producción Nivel: avanzado · Horas estimadas: 4
Laboratorio: observability · Estado: EXECUTABLECORE

Propósito

Montar observabilidad en AWS que sirva para diagnosticar y no para decorar, y declararla como código junto al servicio que la produce. La clase cubre registros estructurados con su coste —que en la clase 207 resultó ser la segunda partida de la factura—, métricas propias sin arruinarse en dimensiones, trazado distribuido con muestreo, y la parte que decide si esto funciona: la alerta se define en el mismo cambio que el servicio, o no existe.

Resultados de aprendizaje

Al finalizar podrás:

1. Emitir registros estructurados con contexto y controlar su coste.
2. Publicar métricas propias sin multiplicar el gasto por dimensiones.
3. Trazar peticiones extremo a extremo con muestreo razonable.
4. Declarar paneles, objetivos y alertas como código.
5. Distinguir lo que hay que vigilar de lo que solo hay que poder consultar.

Conceptos centrales

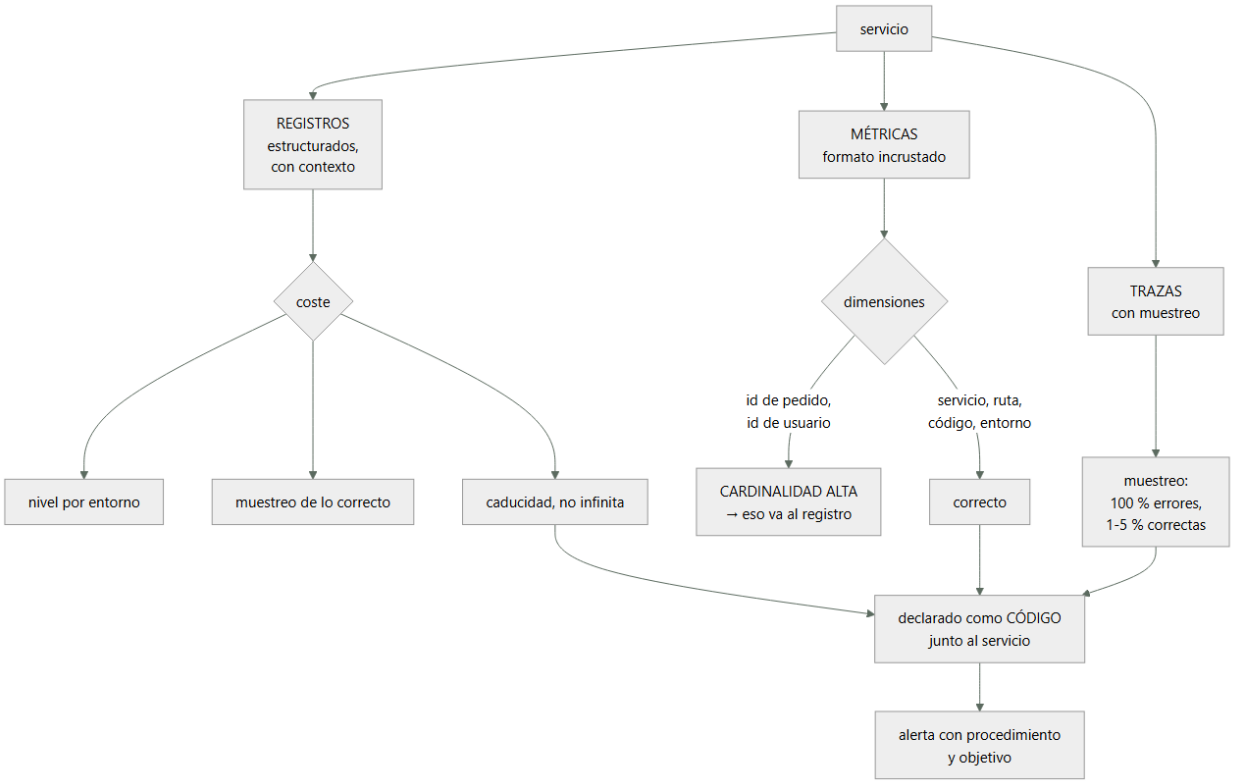
Concepto	Comprensión verificable
registro estructurado	Línea en formato de datos con campos consultables, no texto libre.
métrica de formato incrustado	Métrica publicada dentro de un registro, sin llamada aparte ni coste de API.
dimensión	Etiqueta de una métrica. Cada combinación distinta es una métrica facturable.
cardinalidad	Número de valores distintos de una dimensión. Alta cardinalidad multiplica el coste.
traza	Registro del recorrido de una petición por los servicios, con tiempos por tramo.
observabilidad como código	Paneles, objetivos y alertas declarados junto al servicio y desplegados con él.

Modelo mental

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

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. Registros: estructura y coste

En AWS los registros se cobran por ingesta y por almacenamiento, y la ingesta suele ser la partida sorpresa.

```
LO PRIMERO: ESTRUCTURA
  ✗ "Error procesando pedido 4471 del cliente 123"
  ✓ {"nivel":"error","mensaje":"fallo al confirmar",
     "pedido":"4471","cliente":"123",
     "traza":"1-65f...","servicio":"pedidos",
     "version":"1.4.2","duracion_ms":412}

→ el primero se busca con expresiones regulares y falla
→ el segundo se consulta con filtros por campo
```

Y el contexto que debe llevar toda línea:

```
identificador de traza          ← el que une todo   clase 121
servicio y versión
entorno
identificador de la petición
identificador del usuario o del inquilino, si aplica
y la duración, cuando la operación termina
```

El coste, con los tres controles que lo dominan:

```
1  NIVEL POR ENTORNO
   desarrollo   depuración
   producción   información y errores; depuración activable
                por variable sin redesplegar
   → registrar cada petición con su cuerpo en producción es
     el error más caro y el más común

2  MUESTREO DE LO CORRECTO
   el 100 % de los errores
   el 1-5 % de las peticiones correctas
   → y el 100 % de las peticiones de un usuario concreto
     cuando se está diagnosticando, activable

3  CADUCIDAD
   ✗ por defecto, muchos grupos se crean sin caducidad
```

```
  ✓ 14-30 días en producción, y lo que haga falta
    conservar se exporta a almacenamiento barato
  → función de aptitud: ningún grupo sin caducidad
                                              clase 190
```

Y el dato de la clase 207, que justifica todo esto:

```
ingesta de registros                        620 €/mes
almacenamiento                              180 €/mes
cómputo de las funciones                    410 €/mes
→ los registros costaban el doble que el cómputo
tras aplicar nivel, muestreo y caducidad    120 €/mes
```

Y una advertencia sobre lo que nunca debe ir a un registro:

```
contraseñas, testigos, claves
datos personales completos
números de tarjeta
cuerpos de petición sin filtrar
→ los registros se consultan por mucha gente y se conservan
→ y una captura de registros es un dato sensible
                                              clase 189
```

2. Métricas propias y la trampa de las dimensiones

Las métricas de infraestructura vienen dadas. Las que importan —las de negocio y las de servicio— hay que publicarlas.

```
CÓMO PUBLICARLAS
  ✗ una llamada a la API de métricas por cada valor
    → latencia añadida en el camino crítico y coste por
      llamada
  ✓ FORMATO INCRUSTADO: la métrica va dentro del registro
    con una sección de metadatos
    → sin llamada, sin latencia, y el registro sirve además
      como detalle
```

La trampa de las dimensiones, que produce facturas absurdas:

```
cada COMBINACIÓN distinta de dimensiones es una métrica
facturable

  dimensiones      valores      métricas resultantes
  servicio            8
  ruta               24
  código de estado    6
  → 8 × 24 × 6 = 1.152 métricas          razonable

  añade id_de_pedido con 400.000 valores
  → 460 millones de métricas             desastre

REGLA
  dimensión = algo de lo que quieras ver la EVOLUCIÓN
  identificador = algo con lo que quieras BUSCAR
  → la evolución va a métricas; la búsqueda, a registros
```

Y las métricas que de verdad hacen falta, por servicio:

```
LAS CUATRO SEÑALES                            clase 121
  latencia por percentil
  tráfico
  errores, separando los del cliente de los del sistema
  saturación: el recurso que se agota primero

Y LAS DE NEGOCIO, que son las que se echan de menos
  pedidos confirmados por minuto
  pagos rechazados
  retraso entre el hecho y su procesamiento    clase 210
  → una caída de pedidos confirmados detecta cosas que
    ninguna métrica técnica ve                   ley 15
```

Y una advertencia sobre los percentiles:

```
las métricas agregadas por minuto no permiten calcular el
percentil real del periodo mayor
→ el «p99 del día» calculado a partir de p99 por minuto no
```

```
  es el p99 del día
→ para percentiles exactos hacen falta métricas con
  distribución o el detalle en registros
```

Las trazas, con la decisión de muestreo:

```
el trazado distribuido muestra el recorrido de una petición
y cuánto tarda cada tramo                          clase 121

muestreo
  100 % de las peticiones con error
  100 % de las lentas (por encima de un umbral)
  1-5 % del resto
  → trazar todo cuesta mucho y aporta poco

y lo que hay que propagar
  la cabecera de traza a TODAS las llamadas, incluidas las
  asíncronas: por la cola, dentro del mensaje
  → sin esto, la traza se corta en el primer salto
    asíncrono                                      clase 210
```

3. Declararlo como código

La observabilidad montada a mano por la consola envejece y desaparece. Declararla junto al servicio es lo que la mantiene viva.

```
QUÉ SE DECLARA
  el grupo de registros, con su caducidad
  las métricas y sus filtros
  el panel del servicio
  los objetivos de nivel de servicio
  las alertas, con su destino y su procedimiento enlazado
  → todo en la misma plantilla que el servicio

QUÉ SE GANA
  un servicio nuevo nace observable                clase 171
  las alertas se revisan en la misma propuesta de cambio
  y al retirar el servicio, se retira su observabilidad
                                                     ley 23
```

Y la regla que lo hace efectivo:

```
FUNCIÓN DE APTITUD                                 clase 190
  ningún servicio se despliega sin
    grupo de registros con caducidad
    las cuatro señales publicadas
    al menos un objetivo declarado
    al menos una alerta con procedimiento enlazado
→ y el carril fácil lo trae hecho, así que casi nunca falla
```

Alertas: qué merece despertar a alguien.

```
ALERTA (despierta)
  el objetivo de nivel de servicio se está consumiendo
  demasiado rápido                                 clase 125
  el flujo crítico no funciona
  la cola de fallidos tiene mensajes antiguos      clase 210
  el retraso de proceso supera el acordado

AVISO (revisar en horario)
  tendencias, saturación creciente, coste

NI UNA NI OTRA
  «la CPU está al 80 %»        ← no es un síntoma de usuario
  «hubo un error»              ← los errores ocurren
  «un despliegue terminó»
```

Y las alertas que este programa ha demostrado que faltan siempre:

```
ALERTA POR AUSENCIA
  «esta función no se ha ejecutado en N minutos»
  → un trabajo programado que deja de dispararse no genera
    ningún error                                     ley 13

ALERTA POR ANTIGÜEDAD
  «el mensaje más viejo de la cola de fallidos supera 1 h»
```

```
  «este certificado no se renueva desde hace 45 días»
                                                clase 196
  «este dispositivo lleva 2 h sin reportar»      clase 203
```

Y la comprobación de que la alerta funciona:

```
provocar la condición y ver si llega, a quien tiene que
llegar                                              ley 22
→ en este programa, las alertas que salían a canales sin
  suscriptores han aparecido en cuatro clases distintas
```

Y una decisión sobre destino:

```
las alertas van a UN canal con guardia, no a canales
nuevos por servicio
→ crear un canal por servicio garantiza que alguno quede
  sin nadie                                         ley 15
```

4. Consultar, y lo que no hay que vigilar

Hay una diferencia importante entre lo que hay que vigilar y lo que hay que poder consultar.

```
VIGILAR   pocas cosas, con alerta, revisadas
CONSULTAR mucho, sin alerta, disponible cuando haga falta

→ confundirlos produce cientos de alertas y ningún
  diagnóstico                                    clase 125
```

Las consultas que hay que tener preparadas, no improvisar:

```
«dame todas las líneas de esta traza»
«dame los errores de este servicio en la última hora,
 agrupados por tipo»
«dame las peticiones de este usuario»
«dame las peticiones más lentas y qué tramo domina»
«compara esta hora con la misma hora de ayer»

→ guardadas y enlazadas desde el procedimiento de la alerta
→ improvisar consultas durante un incidente cuesta minutos
  que no se tienen                              clase 127
```

Y la correlación, que es lo que hace útil todo lo demás:

```
de una alerta → al panel del servicio
del panel → a las trazas del periodo
de una traza → a los registros de esa petición
de los registros → a la línea de cambios: ¿qué se desplegó?
                                                clase 122

→ si esa cadena tiene un eslabón roto, el diagnóstico se
  hace a ojo
```

Los objetivos de nivel de servicio, declarados:

```
por flujo de usuario, no por componente         clase 123
  «el 99,5 % de las confirmaciones de pedido en menos de
   800 ms, medidas en el borde»
con presupuesto de error y alerta por ritmo de consumo
  → y esa es la alerta que despierta, no el error suelto
```

Y la lista de comprobación de la clase:

- ■ todos los registros son estructurados
- ■ toda línea lleva identificador de traza, servicio, versión y entorno
- ■ el nivel de registro depende del entorno y se puede cambiar sin redesplegar
- ■ hay muestreo de las peticiones correctas
- ■ ningún grupo de registros carece de caducidad
- ■ no se registran secretos ni datos personales completos
- ■ las métricas se publican en formato incrustado
- ■ ninguna dimensión tiene cardinalidad alta
- ■ hay métricas de negocio, no solo técnicas
- ■ la cabecera de traza se propaga también por las colas
- ■ el muestreo de trazas cubre el 100 % de errores y lentas
- ■ paneles, objetivos y alertas están en la plantilla del servicio

- hay alertas por ausencia y por antigüedad
- cada alerta tiene procedimiento enlazado
- las alertas se han probado provocando la condición
- las consultas de diagnóstico están guardadas

Y el cierre que enlaza con la clase siguiente: hasta aquí todo ha sido sin servidores. Cuando la carga no encaja en ese modelo —procesos largos, dependencias pesadas, control del entorno— aparecen los contenedores. Registro de imágenes, orquestación gestionada y escalado es la materia de la clase 212.

Ejemplo trabajado

CloudShop monta la observabilidad de su plataforma. Lo que sigue es la factura de registros que disparó el proyecto, la métrica que costaba 3.100 € al mes por una dimensión, y la alerta que llevaba ocho meses sin llegar a nadie.

El punto de partida:

```
coste mensual de observabilidad              4.980 €
  ingesta de registros                       2.140 €
  almacenamiento de registros                  680 €
  métricas propias                           1.740 €
  trazas                                       420 €

y lo que se obtenía a cambio
  paneles                                       31
  abiertos en el último mes                      4
  alertas configuradas                          89
  disparadas en el último mes                  612
  que resultaron accionables                    41 (6,7 %)
```

Los registros: dónde estaba el gasto.

```
ingesta por servicio
  api-pedidos                              1.190 €
  api-catalogo                               420 €
  procesador-eventos                         310 €
  resto                                      220 €

qué registraba api-pedidos
  cada petición, con el cuerpo completo de entrada y salida
  cada consulta a la base, con sus parámetros
  cada llamada saliente, con cabeceras
  y en nivel de depuración, porque nunca se cambió al
    pasar a producción

  volumen                                 1,4 TB/mes
  líneas por petición                           14
```

Y dos hallazgos al revisar el contenido:

```
1  los cuerpos de petición incluían el testigo de
   autorización completo
   → 41 personas tenían acceso de lectura a los registros
   → un testigo válido en texto plano, conservado sin
     caducidad                                  clase 189

2  el grupo de registros no tenía caducidad
   → 19 meses de historia acumulada
   → 8,4 TB almacenados, de los cuales lo consultado en el
     último año eran los últimos 11 días
```

Las correcciones:

```
nivel por entorno                    depuración → información
  con variable para activar depuración por 30 min sin
  redesplegar
muestreo de peticiones correctas      100 % → 3 %
  errores                             100 %, siempre
cuerpos                               eliminados; solo campos
                                      seleccionados
testigos y datos personales           filtrados en el emisor
caducidad                             sin límite → 21 días
exportación de lo necesario           a almacenamiento barato

líneas por petición                        14 → 2
```

```
volumen                              1,4 TB → 96 GB/mes
ingesta                              2.140 € → 165 €
almacenamiento                         680 € → 40 €
```

La métrica que costaba 3.100 €… que resultaron ser 1.740 €.

```
al desglosar el coste de métricas propias

  métrica                        dimensiones      series
  pedidos_procesados      servicio, ruta, estado     144
  latencia_operacion      servicio, operacion        320
  errores_por_tipo        servicio, tipo              96
  tiempo_confirmacion     servicio, ID_DE_PEDIDO  412.000  ←

  la última la había añadido un equipo para «poder ver el
  tiempo de cada pedido»
  → 412.000 series, cada una con muy pocos puntos
  → 1.410 € de los 1.740 €

corrección
  el tiempo por pedido va al REGISTRO, con el identificador
  como campo consultable
  la MÉTRICA queda con dimensiones servicio y tipo de
  pedido: 24 series
  y para ver un pedido concreto, se consulta el registro

coste de métricas               1.740 € → 290 €
```

Y la regla que quedó escrita:

```
¿quieres ver la EVOLUCIÓN de esto agrupado?   → métrica
¿quieres BUSCAR un caso concreto?             → registro
→ y una función de aptitud que rechaza dimensiones con más
  de 200 valores previstos                     clase 190
```

La alerta que llevaba ocho meses sin llegar a nadie.

```
al auditar las 89 alertas
  con destino a un canal con guardia                34
  con destino a canales creados por equipos          41
    de ellos, canales SIN suscriptores               11
  con destino a un buzón de correo compartido        14
    → el buzón tenía 12.400 correos sin leer

y entre las 11 de canales vacíos
  «la cola de fallidos de facturación tiene mensajes»
  → la del incidente de la clase 210, que llevaba 8 meses
    disparándose sin que nadie la viera            ley 15

corrección
  destino único: canal de guardia por turno
  las 41 alertas de equipos se revisaron
    → 23 se convirtieron en avisos de horario laboral
    → 12 se retiraron por no ser accionables
    →  6 pasaron a alerta real
  prueba de extremo a extremo de cada alerta:
    provocar la condición y comprobar que llega
    → 7 de 40 no llegaron a la primera             ley 22
```

La observabilidad como código.

```
la plantilla de cada servicio incluye ahora
  grupo de registros con caducidad
  filtros de métrica
  panel del servicio
  objetivo de nivel de servicio
  alertas con procedimiento enlazado

y la plantilla de servicio nuevo lo trae hecho
  → en 6 meses se crearon 9 servicios
  → 9 de 9 nacieron con panel, objetivo y alerta
  → antes, la media era de 5 semanas hasta tener alerta,
    y 3 de 11 servicios nunca la tuvieron

función de aptitud
```

```
  «ningún servicio sin grupo con caducidad, cuatro señales,
   objetivo y alerta con procedimiento»
  fallos en 6 meses                                    2
    → los 2, servicios antiguos, corregidos
```

La cadena de diagnóstico, comprobada con un incidente real:

```
03:14  alerta: el objetivo de confirmación de pedido está
       consumiendo presupuesto de error 14× más rápido de
       lo normal
03:14  el mensaje enlaza el panel del servicio
03:15  el panel muestra p99 disparado desde las 03:02
03:15  la línea de cambios: despliegue de precios a las
       03:01                                  clase 122
03:16  las trazas del periodo: el tramo lento es la llamada
       a precios
03:17  los registros de esas trazas: error de conexión al
       almacén de precios
03:18  reversión automática ya en curso; confirmada
03:22  restablecido

tiempo hasta identificar la causa                4 min
  antes de este proyecto, la media era de       52 min
```

El resultado:

```
coste de observabilidad                  4.980 €      620 €
ingesta de registros                      1,4 TB      96 GB
series de métricas                       412.500         584
alertas configuradas                          89          63
  disparadas al mes                          612          71
  accionables                           41 (6,7 %)  63 (89 %)
alertas que no llegaban a nadie                11           0
servicios sin panel ni objetivo                 3           0
tiempo medio hasta identificar causa         52 min       4 min
testigos en texto plano en registros            sí          no
```

La lección que esta clase deja: de casi cinco mil euros mensuales de observabilidad, una sola dimensión con cuatrocientos mil valores costaba mil cuatrocientos, y lo que ese equipo quería —ver el tiempo de un pedido concreto— se resuelve con un campo en el registro. Los registros costaban más que el cómputo y contenían testigos válidos en texto plano. Y de ochenta y nueve alertas, once salían a canales sin nadie, incluida la de la cola de fallidos que ocho meses después provocó el incidente de la clase 210.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-17-aws-production-architecture/211-cloudwatch-x-ray-y-observabilidad-como-codigo/
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 aws-observability 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 aws-observability 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 factura de registros supera a la de cómputo	Nivel de depuración en producción, cuerpos completos y sin muestreo ni caducidad	Nivel por entorno activable sin redesplegar, muestreo de las peticiones correctas, campos seleccionados y caducidad obligatoria.
El coste de métricas se dispara sin explicación	Una dimensión de alta cardinalidad crea una serie por valor	Las dimensiones son para ver evolución agrupada; lo que sirve para buscar un caso concreto va al registro.
La traza se corta al llegar a un proceso asíncrono	La cabecera de traza no se propaga dentro del mensaje de la cola	Incluye el contexto de traza en el mensaje y recupéralo en el consumidor.
Una alerta lleva meses disparándose sin que nadie actúe	Sale a un canal creado por un equipo y sin suscriptores	Destino único con guardia y prueba de extremo a extremo provocando la condición.
Un servicio lleva semanas en producción sin panel ni alerta	La observabilidad se monta a mano después del despliegue	Declárala en la plantilla del servicio y exígela con una función de aptitud; el carril por defecto debe traerla hecha.
Los registros contienen testigos o datos personales	Se registran cuerpos completos sin filtrar	Filtra en el emisor, registra solo campos seleccionados y trata el archivo de registros como dato sensible.

■ 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 campos debe llevar toda línea de registro y por qué?
2. ¿Qué distingue algo que debe ser dimensión de algo que debe ser campo de registro?
3. ¿Qué muestreo de trazas resulta razonable y por qué no se traza todo?
4. ¿Qué dos tipos de alerta faltan casi siempre y qué detectan?
5. ¿Qué eslabones tiene la cadena de diagnóstico desde la alerta hasta la causa?

Referencias

- AWS (2025). CloudWatch embedded metric format. <https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatchEmbeddedMetricFormat.html>

- AWS (2025). CloudWatch Logs Insights query syntax. <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWLQuerySyntax.html>
- AWS (2025). AWS X-Ray sampling rules. <https://docs.aws.amazon.com/xray/latest/devguide/xray-console-sampling.html>
- OpenTelemetry (2025). Context propagation. <https://opentelemetry.io/docs/concepts/context-propagation/>
- Google (2018). The Site Reliability Workbook: alerting on SLOs. <https://sre.google/workbook/alerting-on-slos/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 17 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 210 · EventBridge, SQS, DLQ, replay e idempotencia	Parte 17 · Programa	212 · ECR, ECS Fargate, ALB y autoscaling →

Evaluación — 211 CloudWatch, X-Ray y observabilidad como código

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 aws-observability 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.

212 — ECR, ECS Fargate, ALB y autoscaling

Parte: 17 — AWS: arquitectura, automatización y operación en producción Nivel: avanzado · Horas estimadas: 4
Laboratorio: container · Estado: EXECUTABLECORE

Propósito

Ejecutar contenedores en AWS sin gestionar servidores, con las decisiones que separan un despliegue que funciona de uno que aguanta. La clase cubre el registro de imágenes y su higiene, la elección entre capacidad gestionada y máquinas propias, la configuración de servicio y balanceador que evita cortar peticiones, y el escalado: por qué la señal correcta casi nunca es la CPU y por qué el escalado siempre llega tarde al pico.

Resultados de aprendizaje

Al finalizar podrás:

1. Publicar imágenes con exploración, inmutabilidad y caducidad.
2. Elegir entre capacidad gestionada y máquinas propias con cifras.
3. Configurar servicio, comprobaciones y drenaje sin cortar peticiones.
4. Escalar con la señal correcta y con margen para el retraso.
5. Dimensionar CPU y memoria midiendo, no copiando.

Conceptos centrales

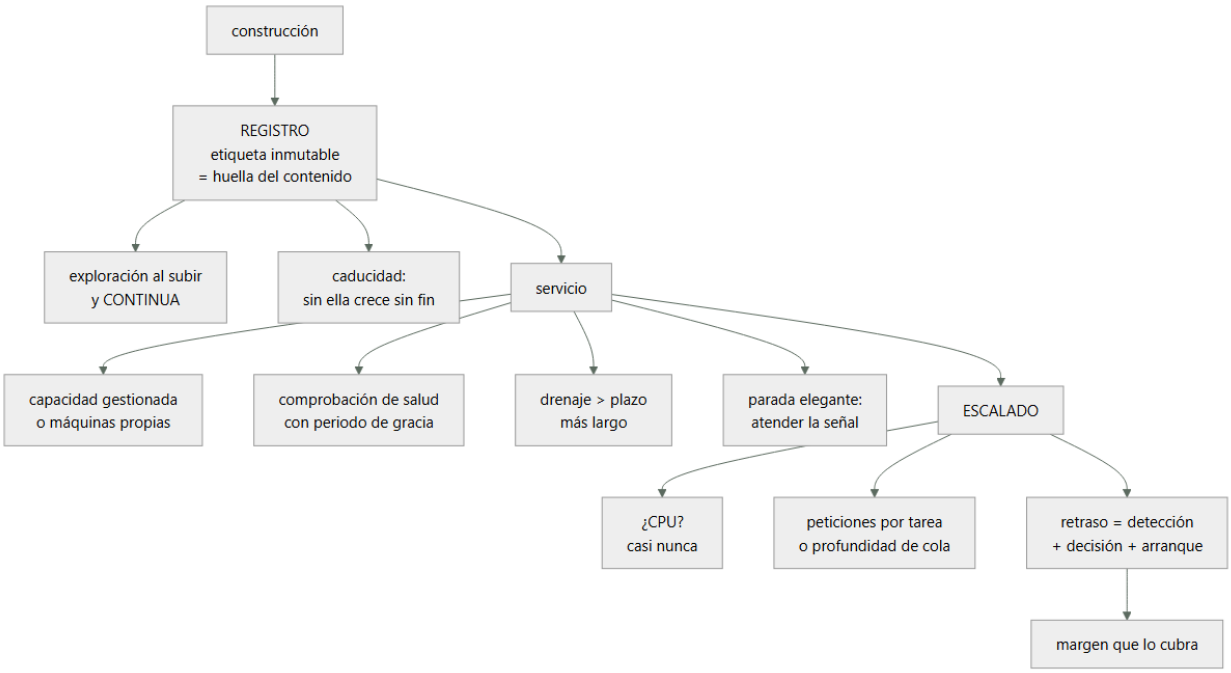
Concepto	Comprensión verificable
registro de imágenes	Almacén de imágenes de contenedor, con exploración de vulnerabilidades y reglas de caducidad.
etiqueta inmutable	Etiqueta que no se puede sobrescribir. Evita que 'la misma versión' cambie de contenido.
definición de tarea	Declaración de qué imagen, con qué recursos, permisos y variables se ejecuta.
capacidad gestionada	Ejecución sin administrar instancias. Se paga por recursos pedidos y por tiempo.
drenaje de destino	Tiempo que el balanceador deja terminar las peticiones en curso antes de retirar una tarea.
retraso de escalado	Suma de detección, decisión y arranque. Es lo que hay que cubrir con margen.

Modelo mental

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

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

■ Flujo de razonamiento



Desarrollo

1. El registro y la higiene de imágenes

El registro parece un detalle y produce dos problemas caros: imágenes que cambian bajo el mismo nombre y crecimiento sin límite.

```
ETIQUETAS INMUTABLES
  ✗ etiqueta «latest» o «v1.4» sobrescribible
    → dos despliegues con la misma etiqueta ejecutan cosas
      distintas
    → y revertir no lleva a lo que había
  ✓ etiqueta = huella del contenido, o versión inmutable
    → la etiqueta identifica un contenido exacto
    → y el despliegue referencia la huella, no la etiqueta
                                                clase 106

EXPLORACIÓN
  al subir: bloquea lo que tenga vulnerabilidades graves
  CONTINUA: una imagen desplegada hace tres meses puede
    tener hoy una vulnerabilidad conocida
  → sin exploración continua, solo se examina lo nuevo
                                                  ley 13

CADUCIDAD
  reglas: conservar las N últimas de cada rama, borrar las
  demás; borrar las sin etiqueta a los pocos días
  → sin reglas, el registro crece indefinidamente y factura
  → y las imágenes viejas con vulnerabilidades siguen ahí
    disponibles                                   ley 25
```

Y dos decisiones sobre la imagen misma:

```
TAMAÑO
  imagen base mínima, construcción en varias etapas
  → el tiempo de arranque depende de la descarga
  → 800 MB frente a 90 MB son segundos en cada arranque,
    y eso se paga en cada escalado

SIN SECRETOS DENTRO
  las variables con secretos se inyectan en ejecución desde
  el gestor de secretos, no se construyen dentro
  → una imagen se puede descargar y examinar
```

Y el permiso que hay que separar y casi nunca se separa:

```
ROL DE EJECUCIÓN     el que usa el agente para descargar la
                     imagen y escribir registros
ROL DE LA TAREA      el que usa TU código para llamar a
                     servicios
→ mezclarlos da al código permisos de infraestructura
                                                clase 134
```

2. Capacidad: gestionada o propia

La elección entre ejecutar sin administrar instancias o gestionarlas se decide con cifras, no con preferencia.

```
CAPACIDAD GESTIONADA (Fargate)
  + sin instancias que parchear, escalar ni vigilar
  + aislamiento por tarea
  + se paga por lo pedido, por segundo
  − precio por unidad de recurso mayor
  − tamaños en combinaciones fijas: pedir 1 vCPU y 2 GB
    puede obligar a pagar más
  − arranque algo más lento
  − sin acceso al anfitrión: agentes especiales, GPU o
    almacenamiento local quedan limitados

MÁQUINAS PROPIAS
  + más barato por unidad, sobre todo con compromisos o
    capacidad sobrante                          clase 143
  + control del anfitrión
  − hay que parchear, escalar y vigilar el conjunto
  − el empaquetado deja hueco desperdiciado
  − y ese trabajo consume capacidad del equipo     ley 23
```

Y el cálculo honesto:

```
el ahorro por unidad de las máquinas propias solo se
realiza si el empaquetado es alto
  utilización real típica de un conjunto mal empaquetado
  35-50 %
  → a esa utilización, la capacidad gestionada suele salir
    igual o más barata

regla práctica
  empieza con capacidad gestionada
  mide utilización y coste durante meses
  pasa a máquinas propias solo si el volumen es alto,
    estable y hay quien lo opere
```

Y el modo intermedio que suele ganar:

```
capacidad gestionada para lo variable y lo poco usado
máquinas propias con compromiso para la base estable
→ y el reparto se decide con el histórico, no a ojo
                                                clase 143
```

Y una nota sobre el precio por interrupción:

```
la capacidad interrumpible es mucho más barata y puede
retirarse con poco aviso
→ vale para trabajo por lotes y para parte de la capacidad
  de servicios tolerantes
→ nunca para el 100 % de un servicio con usuarios
```

3. Servicio y balanceador sin cortar peticiones

La configuración por defecto de un servicio detrás de un balanceador corta peticiones en cada despliegue. Estos son los parámetros que lo evitan.

```
COMPROBACIÓN DE SALUD, con dos puntos       clase 196
  /vivo    ¿el proceso está en pie?      → reinicio
  /listo   ¿puede atender ahora?         → recibir tráfico
  y /listo NO depende de dependencias blandas

PERIODO DE GRACIA AL ARRANCAR
  tiempo antes de empezar a evaluar la salud
  ✗ demasiado corto → la tarea se mata mientras arranca,
    y entra en un bucle de reinicios
  ✓ mayor que el arranque real medido, con margen
```

```
DRENAJE DE DESTINO
  ✗ por defecto suele ser corto
  ✓ mayor que el plazo más largo de las peticiones
  → si no, cada despliegue corta lo que estaba en curso

PARADA ELEGANTE
  la aplicación debe ATENDER la señal de terminación
    dejar de aceptar nuevas
    terminar las que tiene
    cerrar conexiones
  ✗ si no la atiende, se la mata al agotar el plazo
  → y ese plazo debe superar la petición más larga

ORDEN CORRECTO EN UN DESPLIEGUE
  1  arranca la tarea nueva
  2  pasa /listo
  3  el balanceador la registra
  4  el balanceador quita la vieja del reparto
  5  DRENAJE
  6  señal de terminación a la vieja
  7  parada elegante
```

Y los parámetros de despliegue del servicio:

```
PORCENTAJE MÍNIMO SANO   100 %  → nunca menos capacidad de
                                  la nominal
PORCENTAJE MÁXIMO        200 %  → permite arrancar las
                                  nuevas antes de retirar
                                  las viejas
→ con mínimo 100 y máximo 200 el despliegue no reduce
  capacidad en ningún momento
→ con los valores por defecto (50/200 en algunos casos) se
  puede quedar a la mitad durante el despliegue

CIRCUITO DE DESPLIEGUE
  si las tareas nuevas no llegan a estar sanas, el servicio
  vuelve solo a la versión anterior
  → hay que activarlo; no viene activado
```

Y una decisión sobre el tipo de balanceador:

```
de aplicación (capa 7)   rutas, cabeceras, reintentos
de red (capa 4)          latencia mínima, IP fija, otros
                         protocolos
→ y la elección es la misma discusión de la clase 196
```

4. Escalar con la señal correcta

El escalado automático por CPU es el valor por defecto y casi nunca es lo correcto.

```
POR QUÉ LA CPU FALLA COMO SEÑAL
  la mayoría de los servicios de aplicación esperan a la
  red y a la base, no calculan
  → una tarea saturada de peticiones en espera puede tener
    la CPU al 30 %
  → es exactamente el caso de la clase 186: el recurso
    saturado era el grupo de conexiones

LAS SEÑALES QUE SÍ FUNCIONAN
  peticiones por tarea (del balanceador)
    → directa, y fácil de fijar con una prueba de carga
  latencia p99 del destino
    → escalar antes de incumplir el objetivo
  profundidad de cola por consumidor
    → para trabajadores asíncronos              clase 210
  concurrencia en vuelo
    → la mejor medida de saturación real

  y la CPU, solo si el servicio de verdad calcula
```

El retraso de escalado, que es lo que hace que llegue tarde:

```
detección de la métrica         30-120 s
decisión y disparo              30-60 s
arranque de la tarea            20-90 s (descarga + arranque)
```

```
paso de la comprobación         10-60 s
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
TOTAL típico                    1,5 a 5 minutos

→ un pico que sube en 60 segundos NO lo cubre el escalado
→ hay que llegar al pico con capacidad ya presente
```

Y de ahí salen tres decisiones:

```
1  MARGEN QUE CUBRA EL RETRASO
   capacidad base = pico previsible × (1 + tasa de subida ×
   retraso)
   → o dicho simple: no ir al límite

2  ESCALADO PROGRAMADO para lo previsible
   la campaña empieza a las 11:00 → subir a las 10:30
   → es más fiable que reaccionar

3  SUBIR RÁPIDO, BAJAR DESPACIO
   escalar hacia arriba con umbral bajo y sin espera
   escalar hacia abajo con espera de varios minutos
   → evita oscilar, y bajar de más cuesta más que subir de
     más
```

El dimensionado de CPU y memoria, que se copia y debería medirse:

```
lo que ocurre al copiar el valor de otro servicio
  memoria de más   se paga y no se usa
  memoria de menos   el proceso muere sin aviso claro
  CPU de menos     latencia alta con «CPU al 100 %», que en
                   contenedores puede ser estrangulamiento
                   por cuota, no falta real de máquina

cómo se decide
  ejecutar la carga real y observar uso de CPU, memoria
  máxima y estrangulamiento
  fijar memoria por encima del pico observado con margen
  y la CPU según la latencia objetivo, no según el uso medio
```

Y la lista de comprobación de la clase:

- ■ las etiquetas del registro son inmutables
- ■ el despliegue referencia la huella, no la etiqueta
- ■ hay exploración al subir y continua
- ■ el registro tiene reglas de caducidad
- ■ la imagen es mínima y no contiene secretos
- ■ el rol de ejecución y el de la tarea están separados
- ■ la elección de capacidad se justificó con utilización y coste
- ■ hay puntos separados de vivo y de listo
- ■ el periodo de gracia supera el arranque medido
- ■ el drenaje supera el plazo más largo
- ■ la aplicación atiende la señal de terminación
- ■ mínimo sano 100 % y máximo 200 %
- ■ el circuito de despliegue está activado
- ■ la señal de escalado no es la CPU salvo que corresponda
- ■ el margen cubre el retraso de escalado, medido
- ■ hay escalado programado para lo previsible
- ■ CPU y memoria se fijaron midiendo

Y el cierre que enlaza con la clase siguiente: cuando los contenedores dejan de ser unos pocos servicios y pasan a ser una plataforma con muchos equipos, la orquestación gestionada se queda corta y aparece Kubernetes con sus propios problemas. Es la materia de la clase 213.

Ejemplo trabajado

CloudShop mueve tres servicios a contenedores. Lo que sigue es el despliegue que cortaba peticiones, el escalado que llegaba tarde a cada campaña, y la comparación de coste que cambió la decisión de capacidad.

Problema 1 · Cada despliegue cortaba peticiones.

```
síntoma   en cada despliegue, entre 200 y 400 peticiones
          fallaban con error de conexión
          duraba unos 40 segundos
```

```
          se había normalizado: «se despliega de madrugada»

diagnóstico, parámetro a parámetro
  drenaje de destino                  30 s (por defecto)
  plazo más largo de petición         45 s (exportaciones)
  → las peticiones de más de 30 s se cortaban

  parada elegante
  → la aplicación NO atendía la señal de terminación
  → se la mataba al agotar el plazo, con peticiones vivas

  porcentaje mínimo sano              50 %
  → durante el despliegue el servicio operaba con la mitad
    de tareas
  → y en hora punta eso bastaba para saturar el resto

  periodo de gracia                   30 s
  arranque real medido                52 s
  → las tareas nuevas se mataban por «no sanas» antes de
    terminar de arrancar
  → el servicio entraba en bucle y el despliegue tardaba
    12 minutos

corrección
  drenaje                             60 s
  plazo de parada                     70 s
  parada elegante implementada: deja de aceptar, termina lo
    que tiene, cierra
  mínimo sano                         100 %
  máximo                              200 %
  periodo de gracia                   120 s
  circuito de despliegue              activado

resultado
  peticiones cortadas por despliegue     300 → 0
  duración del despliegue             12 min → 3 min
  despliegues en horario laboral         no → sí
```

Problema 2 · El escalado llegaba tarde.

```
configuración inicial
  señal          CPU media > 70 %
  mínimo         4 tareas
  máximo         40

lo que pasaba en cada campaña
  11:00  arranca la promoción; el tráfico ×18 en 90 s
  11:00  latencia p99 sube de 180 ms a 4.100 ms
  11:00  CPU de las tareas: 38 %       ← no dispara nada
  11:03  la latencia dispara alertas
  11:04  alguien escala a mano
  11:09  capacidad suficiente

  9 minutos de degradación en cada campaña

diagnóstico
  el servicio esperaba a la base y al servicio de precios
  → la CPU nunca subía
  → el recurso saturado era la concurrencia y las
    conexiones                                 clase 186

corrección
  1  SEÑAL
     peticiones por tarea, del balanceador
     umbral fijado con prueba de carga: el codo estaba en
     240 peticiones/min por tarea → objetivo 160

  2  RETRASO MEDIDO
     detección                        60 s
     decisión                         40 s
     arranque + descarga de imagen    52 s
     comprobación de salud            25 s
     ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

```
      total                          3 min 17 s

      y la imagen se redujo de 740 MB a 110 MB
      → arranque 52 s → 21 s
      → total 2 min 46 s

   3  MARGEN
      el tráfico sube ×18 en 90 s
      → el escalado no puede cubrirlo
      → capacidad base subida para absorber los primeros
        3 minutos

   4  ESCALADO PROGRAMADO
      las campañas tienen hora conocida
      → subir a 26 tareas a las 10:30
      → y volver a la política automática a las 13:00

   5  SUBIR RÁPIDO, BAJAR DESPACIO
      subida sin espera; bajada con 10 min de espera

resultado en la campaña siguiente
  p99 máximo                      4.100 ms → 310 ms
  minutos de degradación               9 → 0
  intervenciones manuales               1 → 0
  coste del escalado programado    +14 €/campaña
```

La decisión de capacidad, revisada con datos.

```
se empezó con capacidad gestionada; a los 5 meses se
planteó pasar a máquinas propias «porque es más barato»

los datos
  coste actual con capacidad gestionada     3.180 €/mes
  recursos pedidos                          124 vCPU · 248 GB
  utilización media real                    31 %

  estimación con máquinas propias
    instancias necesarias con empaquetado del 65 %
      → 9 instancias grandes            1.940 €/mes
    con compromiso de 1 año             1.360 €/mes
    parcheado, escalado del conjunto,
      vigilancia                        ~2,5 días-persona/mes

  y el cálculo honesto
    ahorro bruto                          1.820 €/mes
    coste del trabajo (2,5 días × ~450 €) 1.125 €/mes
    ahorro neto                             695 €/mes

decisión
  NO migrar todavía
  motivo   695 €/mes no compensa añadir un conjunto de
           instancias que parchear a un equipo de 6
                                                    ley 23
  qué la reabriría
           si el gasto supera 6.000 €/mes
           o si aparece una carga con GPU o con requisitos
           de anfitrión                          clase 190

y lo que sí se hizo, que dio más
  ajustar los tamaños pedidos, que estaban copiados
    servicio A   2 vCPU / 4 GB → 0,5 vCPU / 2 GB
    servicio B   2 vCPU / 4 GB → 1 vCPU / 2 GB
    servicio C   4 vCPU / 8 GB → 2 vCPU / 4 GB
  utilización media                    31 % → 62 %
  coste                          3.180 € → 1.710 €
```

Y el detalle de cómo se fijaron esos tamaños:

```
se ejecutó la carga real durante una semana midiendo
  uso de CPU p95, memoria máxima y estrangulamiento por
  cuota

  servicio A   CPU p95 0,21 vCPU · memoria máx 1,3 GB
               estrangulamiento 0 %
```

```
  servicio C   CPU p95 1,7 vCPU · memoria máx 3,1 GB
               estrangulamiento 4 % con 2 vCPU
               → se dejó en 2 vCPU: el 4 % ocurría en el
                 arranque, no en régimen
```

La higiene del registro, que nadie había mirado:

```
imágenes almacenadas                              8.410
  con etiqueta                                    1.240
  SIN etiqueta (capas huérfanas)                  7.170
tamaño                                             2,1 TB
coste                                            210 €/mes
etiquetas mutables                                  sí
exploración                                      al subir
exploración continua                                no

tras aplicar reglas
  conservar las 10 últimas por rama, borrar el resto
  borrar sin etiqueta a los 7 días
  etiquetas inmutables, con huella del contenido
  exploración continua activada

  imágenes                                 8.410 → 310
  tamaño                                   2,1 TB → 74 GB
  coste                                    210 € → 8 €

y la exploración continua encontró
  3 imágenes en producción con vulnerabilidades graves
  publicadas DESPUÉS de su construcción
  → la más antigua llevaba 4 meses desplegada    ley 13
```

El resultado:

```
peticiones cortadas por despliegue          300          0
duración del despliegue                  12 min       3 min
minutos de degradación por campaña            9          0
utilización de recursos                    31 %        62 %
coste de cómputo                        3.180 €     1.710 €
coste del registro                        210 €         8 €
imágenes con vulnerabilidad grave en
  producción                                  3          0
```

La lección que esta clase deja: el despliegue cortaba peticiones por cuatro parámetros por defecto a la vez —drenaje corto, sin parada elegante, mínimo sano al 50 % y gracia menor que el arranque—, y ninguno era un error de diseño. El escalado llegaba tarde porque la señal era la CPU en un servicio que espera a la red, y la mejora mayor no fue cambiar la señal sino reducir la imagen de 740 a 110 MB. Y el ahorro que se buscaba en el tipo de capacidad estaba en realidad en los tamaños pedidos, que estaban copiados de otro servicio.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-17-aws-production-architecture/212-ecr-ecs-fargate-alb-y-autoscaling/lab.py
```

El laboratorio selecciona el motor de práctica container 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 aws-ecs-platform 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 imagen mínima, escaneada y ejecutada sin privilegios. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

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

■ Criterio de aceptación

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

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
Cada despliegue corta peticiones en curso	Drenaje menor que el plazo más largo y la aplicación no atiende la señal de terminación	Aumenta el drenaje por encima de la petición más larga e implementa la parada elegante.
Las tareas nuevas entran en bucle de reinicio	El periodo de gracia es menor que el tiempo real de arranque	Mide el arranque y fija el periodo de gracia con margen sobre él.
El servicio pierde capacidad durante los despliegues	El porcentaje mínimo sano permite bajar del cien por cien	Fija mínimo sano 100 % y máximo 200 %, y activa el circuito de despliegue.
El escalado no reacciona aunque la latencia se dispare	La señal es la CPU y el servicio espera a la red, no calcula	Escala por peticiones por tarea, latencia o profundidad de cola, con el umbral fijado en una prueba de carga.
El escalado llega tarde a cada pico	El retraso de detección, decisión y arranque no está cubierto por margen	Mide el retraso, reduce el tamaño de la imagen, deja margen de capacidad y programa la subida para los picos previsibles.
Una imagen desplegada hace meses tiene vulnerabilidades conocidas	Solo hay exploración al subir, y las etiquetas son mutables	Activa exploración continua, usa etiquetas inmutables con huella y aplica reglas de caducidad al registro.

■ 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 las etiquetas mutables impiden revertir con seguridad?
2. ¿Qué condiciones deben cumplirse para que las máquinas propias salgan realmente más baratas?
3. ¿Cuál es el orden correcto de un despliegue sin cortar peticiones?
4. ¿Por qué la CPU suele ser mala señal de escalado y cuáles funcionan mejor?
5. ¿Qué compone el retraso de escalado y cómo se compensa?

Referencias

- AWS (2025). Amazon ECR: image tag mutability and lifecycle policies. <https://docs.aws.amazon.com/AmazonECR/latest/userguide/image-tag-mutability.html>

- AWS (2025). Amazon ECS deployment types and circuit breaker. <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/deployment-type-ecs.html>
- AWS (2025). Application Load Balancer: deregistration delay. <https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-target-groups.html>
- AWS (2025). Service auto scaling for Amazon ECS. <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-auto-scaling.html>
- AWS (2025). AWS Fargate pricing and task sizing. <https://aws.amazon.com/fargate/pricing/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 17 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 211 · CloudWatch, X-Ray y observabilidad como código	Parte 17 · Programa	213 · EKS, IRSA, GitOps y operación de clúster →

Evaluación — 212 ECR, ECS Fargate, ALB y autoscaling

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 aws-ecs-platform 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.

213 — EKS, IRSA, GitOps y operación de clúster

Parte: 17 — AWS: arquitectura, automatización y operación en producción Nivel: avanzado · Horas estimadas: 4
Laboratorio: kubernetes · Estado: EXECUTABLECORE

Propósito

Operar Kubernetes gestionado en AWS sabiendo qué se gana y qué se paga, porque es la decisión de plataforma que más capacidad de equipo consume. La clase cubre la identidad de las cargas sin credenciales, la reconciliación desde el repositorio, los tres asuntos que causan casi todos los incidentes —peticiones y límites mal puestos, actualizaciones de versión y complementos— y la pregunta que hay que contestar antes: ¿hace falta Kubernetes, o basta con lo de la clase 212?

Resultados de aprendizaje

Al finalizar podrás:

1. Decidir si Kubernetes compensa frente a orquestación más simple.
2. Dar identidad a las cargas sin credenciales estáticas.
3. Operar el clúster desde el repositorio, con reconciliación.
4. Configurar peticiones y límites sin provocar expulsiones ni desperdicio.
5. Planificar las actualizaciones de versión y de complementos.

Conceptos centrales

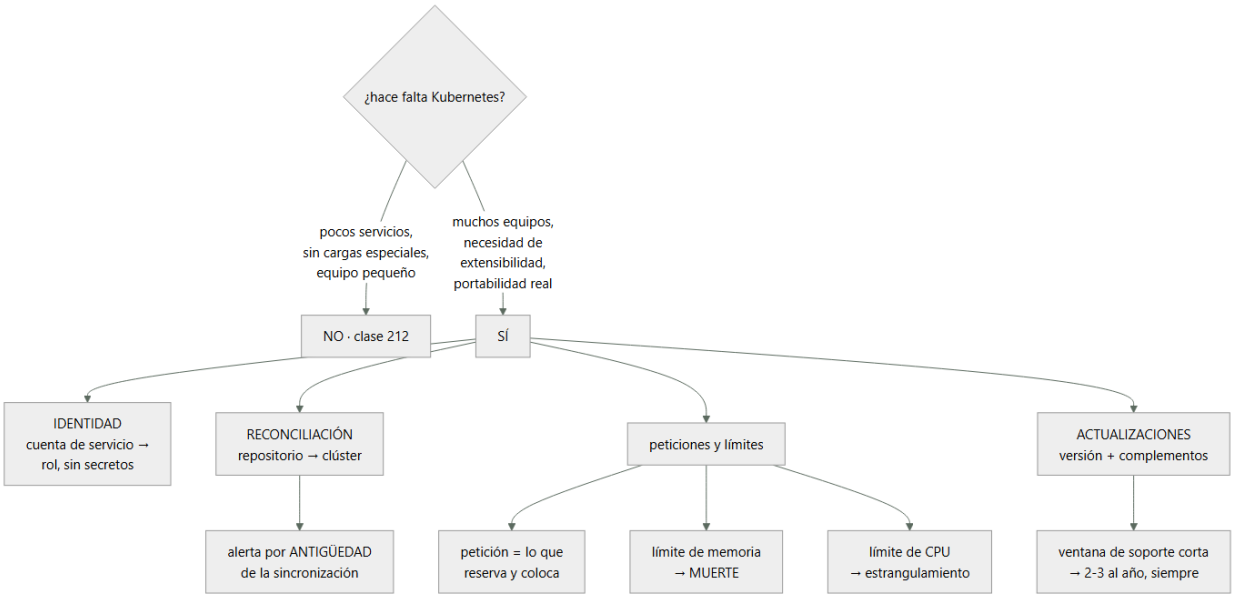
Concepto	Comprensión verificable
identidad de carga (IRSA)	Mecanismo por el que una cuenta de servicio del clúster obtiene credenciales temporales de AWS, sin secretos.
reconciliación	Bucle que compara el estado declarado en el repositorio con el real y corrige la diferencia.
petición de recursos	Lo que la carga declara necesitar. Determina dónde se coloca y la capacidad total requerida.
límite de recursos	Techo que la carga no puede superar. Con memoria significa muerte; con CPU, estrangulamiento.
complemento	Componente añadido al clúster (red, DNS, métricas, controladores). Cada uno es una dependencia de versión.
presupuesto de interrupción	Declaración de cuántas réplicas pueden estar caídas a la vez durante mantenimiento.

Modelo mental

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

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. ¿Hace falta Kubernetes?

Es la pregunta que hay que contestar primero, y la respuesta honesta suele decepcionar.

```
LO QUE KUBERNETES DA DE VERDAD
  un modelo declarativo extensible: se pueden definir
    recursos propios y controladores
  un ecosistema enorme de componentes ya hechos
  la misma interfaz en cualquier nube y en local
  planificación fina: afinidades, tolerancias, prioridades
  y una comunidad con respuestas para casi todo

LO QUE CUESTA
  actualizaciones frecuentes y obligatorias
  complementos que hay que mantener y que se rompen entre sí
  una capa más de red, de identidad y de diagnóstico
  y personas que sepan operarlo

  → entre 1 y 3 personas dedicadas para una plataforma
    seria                                          ley 23
```

Y el criterio de decisión:

```
NO HACE FALTA SI
  son unos pocos servicios web y trabajadores
  el equipo es pequeño
  no hay cargas con requisitos especiales
  → la orquestación gestionada de la clase 212 hace lo
    mismo con una fracción del trabajo

SÍ COMPENSA SI
  hay muchos equipos que necesitan autonomía sobre una base
    común                                       clase 171
  hacen falta operadores y recursos propios
  hay cargas heterogéneas: GPU, procesos largos, trabajos
    por lotes, bases de datos con operador
  la portabilidad entre nubes es un requisito REAL y no
    declarativo                                 clase 158
```

Y una advertencia sobre la portabilidad, que es el argumento más usado:

```
el manifiesto es portable; el resto casi nunca
  balanceadores, almacenamiento, identidad, registro,
  certificados y controladores son específicos
→ la portabilidad real exige diseñarla y probarla, no viene
  de usar Kubernetes                            clase 158
```

Y la decisión sobre el modo de ejecución de los nodos:

```
GRUPOS DE NODOS GESTIONADOS
  instancias que se parchean con ayuda del proveedor
  + control, coste por unidad menor, cualquier carga
  − hay que planificar reemplazos y drenajes

CAPACIDAD SIN NODOS
  un pod por unidad de capacidad, sin instancias
  + nada que parchear
  − más caro por unidad, con límites de funciones

  → mezcla habitual: sin nodos para lo variable y grupos
    gestionados para la base                    clase 212
```

2. Identidad sin credenciales

Dar a una carga acceso a servicios de AWS con una clave montada en un secreto es el patrón que hay que eliminar, y el mecanismo es el mismo de la clase 206.

```
CÓMO FUNCIONA
  el clúster expone un emisor de testigos
  se declara como proveedor de identidad en AWS
  la cuenta de servicio del pod se anota con un rol
  el pod recibe un testigo montado, con vida corta
  y lo intercambia por credenciales temporales

LO QUE SE GANA
  sin claves en secretos ni en variables
  credenciales de minutos
  y permisos por CARGA, no por nodo
```

Y el patrón que sustituye, que sigue siendo frecuente:

```
✗ EL ROL DEL NODO
  dar permisos al rol de la instancia
  → TODOS los pods del nodo los tienen
  → incluidos los de otro equipo que aterricen ahí
  → alcance enorme desde cualquier pod comprometido
                                              clase 189

  y además hay que bloquear el acceso de los pods al
  servicio de metadatos del nodo, o pueden pedir esas
  credenciales aunque tengan la suya
```

Y los errores de configuración de la identidad de carga:

```
la condición del rol debe atar
  el emisor del clúster concreto
  el espacio de nombres
  y el NOMBRE de la cuenta de servicio

✗ condición solo por emisor
  → cualquier cuenta de servicio de cualquier espacio de
    nombres puede asumir el rol
  → es el mismo fallo de la clase 206, con otra forma
```

Y una comprobación obligatoria:

```
desde un pod de otro espacio de nombres, intentar asumir
el rol
→ debe fallar                                   ley 22
```

Los secretos, que en Kubernetes son un objeto sin cifrar por defecto:

```
un objeto de tipo secreto está codificado, no cifrado
→ hay que activar el cifrado en reposo del almacén del
  plano de control
→ y para secretos de verdad, obtenerlos del gestor externo
  en tiempo de ejecución, no guardarlos en el clúster
→ y nunca en un manifiesto del repositorio      clase 106
```

3. Operar desde el repositorio

Aplicar manifiestos a mano desde un portátil produce clústeres que nadie sabe reconstruir. La reconciliación resuelve eso.

```
EL MODELO
  el estado deseado vive en un repositorio
  un controlador dentro del clúster lo compara con el real
  y corrige la diferencia continuamente

LO QUE APORTA sobre desplegar desde la canalización
  el clúster no necesita credenciales de la canalización:
    tira, no le empujan
  la deriva se corrige sola
  el repositorio es la verdad, y se puede reconstruir
  y el historial de cambios es el del repositorio
                                              clase 103
```

Y la alerta imprescindible:

```
«la sincronización lleva N minutos sin completarse»
  o «el estado real difiere del declarado desde hace N»

→ un bucle de reconciliación parado no genera ningún error:
  simplemente deja de aplicar cambios              ley 13
→ y en la clase 179 esa prueba negativa falló porque la
  alerta iba a un canal sin nadie
```

La separación por equipos, que es la razón de usar Kubernetes en muchos casos:

```
espacio de nombres por equipo o por servicio
cuotas de recursos por espacio
  → un equipo no puede consumir la capacidad de los demás
rangos de límites por defecto
políticas de admisión
  → qué imágenes se admiten (firmadas, del registro propio)
  → qué no se permite: privilegios, red del anfitrión,
    montajes sensibles
  → y las etiquetas obligatorias                clase 142
políticas de red
  → por defecto, denegar; y abrir lo declarado  clase 201
```

Y la advertencia sobre la admisión:

```
un control de admisión que rechaza sin explicar por qué se
rodea: la gente pide excepciones o crea recursos por otro
camino                                            ley 16
→ el mensaje debe decir qué falta y cómo arreglarlo
→ y el carril fácil —la plantilla de servicio— debe cumplir
  solo                                          clase 190
```

4. Peticiones, límites y actualizaciones

Peticiones y límites son la causa de la mayoría de los incidentes de plataforma, y se ponen mal en las dos direcciones.

```
PETICIÓN
  lo que el planificador RESERVA
  determina en qué nodo cabe y cuánta capacidad total hace
  falta

LÍMITE
  el techo
  memoria por encima del límite → el proceso MUERE
  CPU por encima del límite     → estrangulamiento, latencia

LOS CUATRO ERRORES
  1  sin petición
     el pod se coloca en cualquier sitio y compite
     → y es el primero en ser expulsado cuando falta
       memoria
  2  petición mucho mayor que el uso
     capacidad reservada y desperdiciada
     → el conjunto crece sin necesidad, y la factura con él
  3  límite de memoria ajustado al uso medio
     un pico normal mata el proceso
```

```
     → y el síntoma es «se reinicia solo y no sé por qué»
  4  límite de CPU muy bajo
     estrangulamiento constante: latencia alta con el nodo
     ocioso                                    clase 186
```

Y las reglas que funcionan:

```
petición de memoria ≈ uso p95 observado
límite de memoria   ≈ petición × 1,5 a 2
petición de CPU     ≈ uso p95 observado
límite de CPU       a menudo, NINGUNO
  → dejar que use el hueco libre cuando lo hay
  → salvo que haga falta aislar por contrato

y medir siempre antes: la carga real durante días
```

Y tres mecanismos que evitan sorpresas:

```
PRESUPUESTO DE INTERRUPCIÓN
  «al menos 2 réplicas siempre disponibles»
  → el drenaje de un nodo lo respeta
  → sin esto, un mantenimiento puede dejar el servicio en
    cero réplicas durante segundos

CLASES DE PRIORIDAD
  lo crítico expulsa a lo que no lo es cuando falta espacio

REPARTO ENTRE ZONAS
  restricciones que obligan a repartir las réplicas
  → sin ellas, las 3 réplicas pueden acabar en el mismo
    nodo o en la misma zona                  clase 185
```

Las actualizaciones, que son el trabajo continuo de esta plataforma:

```
LA VENTANA DE SOPORTE ES CORTA
  cada versión se soporta poco más de un año
  → hay que actualizar 2 o 3 veces al año, siempre
  → no es opcional ni aplazable                  ley 25

QUÉ HAY QUE COMPROBAR EN CADA ACTUALIZACIÓN
  interfaces retiradas: manifiestos que dejan de ser
    válidos
  compatibilidad de cada complemento con la versión nueva
  compatibilidad del cliente y de las herramientas
  y de los operadores instalados

EL ORDEN
  1  revisar avisos de retirada y corregir manifiestos
  2  actualizar en un clúster de prueba idéntico
  3  actualizar el plano de control
  4  actualizar complementos
  5  reemplazar nodos por lotes, respetando el presupuesto
     de interrupción
```

Y lo que se olvida:

```
los COMPLEMENTOS tienen su propio calendario
  red, DNS interno, controlador de balanceador, métricas,
  autoescalado, controlador de certificados
→ cada uno es una dependencia con su matriz de
  compatibilidad
→ y un clúster con quince complementos tiene quince cosas
  que pueden bloquear una actualización          ley 23
```

Y la lista de comprobación de la clase:

- ■ la decisión de usar Kubernetes está justificada
- ■ las cargas obtienen credenciales por identidad de carga
- ■ el rol del nodo no tiene permisos de aplicación
- ■ los pods no pueden alcanzar el servicio de metadatos
- ■ las condiciones del rol atan espacio de nombres y cuenta de servicio
- ■ el clúster se reconcilia desde el repositorio
- ■ hay alerta por antigüedad de la sincronización
- ■ hay cuotas por espacio de nombres
- ■ hay política de admisión con mensajes que explican

- hay política de red con denegación por defecto
- todas las cargas tienen peticiones, medidas
- los límites de memoria dan margen sobre el pico
- hay presupuestos de interrupción
- las réplicas se reparten entre zonas
- el calendario de actualización está planificado
- la matriz de compatibilidad de complementos está escrita

Y el cierre que enlaza con la clase siguiente: todo lo montado en esta parte genera factura, y a estas alturas ya han aparecido tres partidas que nadie había estimado. Presupuestos, análisis de coste, etiquetado y automatización es la materia de la clase 214.

Ejemplo trabajado

CloudShop monta una plataforma de contenedores para siete equipos. Lo que sigue es la decisión de adoptarlo, los tres incidentes del primer año —todos de peticiones y límites o de actualizaciones— y lo que costó de verdad.

La decisión, con el método de la clase 152.

```
situación   14 servicios en orquestación gestionada,
            funcionando bien

lo que se pedía
  7 equipos quieren autonomía para desplegar sin
    coordinarse
  2 cargas con GPU para el equipo de datos     clase 175
  trabajos por lotes con planificación por prioridad
  un operador de base de datos que el equipo de datos ya usa
  y «portabilidad», que al preguntar resultó no ser un
    requisito real                              clase 158

lo que ya estaba resuelto
  despliegue, escalado y balanceo             clase 212

veredicto
  3 de las 5 necesidades exigen Kubernetes o lo facilitan
  mucho
  y hay 2 personas que pueden dedicarse a la plataforma
  → SE ADOPTA, y se migran solo las cargas que lo
    necesitan

y la decisión que se registró
  los 14 servicios web SIGUEN en orquestación gestionada
  motivo   funcionan, y moverlos añade trabajo sin
           beneficio
  qué la reabriría   si la plataforma demuestra reducir el
           tiempo de despliegue de un servicio nuevo por
           debajo del actual                    clase 190
```

Incidente 1 · «Los pods se reinician solos», mes 2.

```
síntoma   el servicio de informes se reiniciaba varias
          veces al día, sin errores en sus registros
          la última línea era siempre distinta

diagnóstico
  el pod moría por exceso de memoria
  límite de memoria                    512 Mi
  uso medio                            410 Mi
  uso p99                              780 Mi (al generar
                                       informes grandes)
  → el límite estaba puesto sobre el uso MEDIO

y por qué no había errores
  el proceso se mata sin darle oportunidad de registrar
  nada
  → el único rastro era el motivo de terminación del
    contenedor, que nadie miraba                ley 15

corrección
  petición de memoria      = p95 observado = 620 Mi
  límite                   = petición × 1,8 = 1.100 Mi
  y alerta nueva: «contenedores terminados por memoria > 0»
```

```
reinicios al día                          6 → 0
```

Incidente 2 · «Latencia alta con los nodos al 30 %», mes 4.

```
síntoma   el servicio de catálogo tenía p99 de 2,4 s
          los nodos estaban al 30 % de CPU
          escalar el número de réplicas no mejoraba nada

diagnóstico
  límite de CPU del contenedor          200 milicores
  el contenedor era estrangulado el 74 % del tiempo
  → la métrica de estrangulamiento existía y no estaba en
    ningún panel                                clase 211

  la confusión
    el panel mostraba «CPU del nodo: 30 %»
    y la conclusión era «sobra capacidad»
    → pero el contenedor tenía su propio techo

corrección
  petición de CPU          = p95 = 320 milicores
  límite de CPU            eliminado
  → el contenedor usa el hueco libre del nodo cuando lo hay

p99                                     2,4 s → 210 ms
réplicas necesarias                        12 → 6
coste                                   -840 €/mes
```

Y la observación:

```
quitar el límite de CPU redujo a la mitad las réplicas
necesarias
→ el límite estaba causando el problema que se intentaba
  resolver escalando
```

Incidente 3 · La actualización de versión, mes 8.

```
situación   la versión del clúster entraba en fin de soporte
            en 6 semanas
            se planificó la actualización para un martes

lo que salió mal
  el clúster de prueba se había creado hacía 4 meses y no
  era idéntico: le faltaban 3 complementos que producción
  sí tenía
  → la prueba pasó sin problemas

  en producción
    el controlador del balanceador no era compatible con la
    versión nueva
    → los servicios nuevos dejaron de obtener balanceador
    → los existentes siguieron funcionando
    tiempo hasta detectarlo                    3 días
    → porque no había despliegues de servicios nuevos hasta
      entonces                                   ley 13

    y 41 manifiestos usaban una interfaz retirada
    → la reconciliación empezó a fallar en silencio para
      esos recursos
    → alerta de antigüedad de sincronización: NO existía

correcciones
  el clúster de prueba se genera del mismo repositorio que
  producción y se recrea antes de cada actualización
  matriz de compatibilidad de los 12 complementos, escrita
    y comprobada antes de actualizar
  revisión automática de interfaces retiradas en la
    canalización                               clase 190
  alerta de antigüedad de sincronización, con destino a
    guardia
  calendario: 3 actualizaciones al año, en fecha fijada

la siguiente actualización, mes 13
  duración                                      4 h
  incidencias                                     0
```

```
  manifiestos corregidos antes                    17
  complementos que hubo que subir primero          4
```

La identidad, montada bien desde el principio:

```
identidad de carga por cuenta de servicio
condición del rol atada a emisor + espacio de nombres +
  nombre de la cuenta
acceso de los pods al servicio de metadatos del nodo:
  bloqueado
rol del nodo: solo lo imprescindible para unirse al clúster

prueba negativa
  desde un pod del espacio de nombres «equipo-datos»,
  intentar asumir el rol de «equipo-pagos»
  → denegado                                       ✓
  desde un pod, leer las credenciales del nodo
  → bloqueado                                      ✓
```

Lo que costó de verdad, medido al año:

```
coste de cómputo del clúster                4.120 €/mes
coste del plano de control                     68 €/mes

y el coste que no aparece en la factura
  actualizaciones (3 al año, ~3 días cada una)  9 días
  mantenimiento de complementos                18 días
  soporte a los 7 equipos                      44 días
  incidentes de plataforma                     11 días
  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  total                                        82 días/año
                                      ≈ 0,4 personas

y lo que se esperaba   «2 personas dedicadas»
→ resultó menos de lo previsto, porque los 14 servicios web
  se quedaron fuera y la reconciliación evitó mucho soporte
```

Y el balance que el equipo escribió:

```
lo que la plataforma dio
  los 7 equipos despliegan sin coordinarse
  las cargas con GPU y los trabajos por lotes funcionan
  el operador de base de datos del equipo de datos también

lo que NO dio
  portabilidad: 4 de los 12 complementos son específicos
  del proveedor, y los balanceadores y el almacenamiento
  también                                       clase 158
  → la portabilidad se había citado como motivo y no se
    materializó, lo cual estaba previsto y registrado
```

El resultado:

```
equipos que despliegan sin coordinarse         0           7
reinicios por memoria                       6/día          0
p99 del catálogo                           2,4 s       210 ms
réplicas del catálogo                         12           6
incidentes por actualización                   1           0
servicios web migrados innecesariamente        0           0
capacidad de equipo consumida                n/d      0,4 pers.
```

La lección que esta clase deja: los dos primeros incidentes fueron peticiones y límites mal puestos, y en el segundo el límite de CPU era la causa del problema que se estaba intentando resolver añadiendo réplicas. El tercero fue una actualización cuya prueba pasó porque el clúster de prueba no era idéntico al de producción, y sus consecuencias tardaron tres días en verse porque nada las hacía visibles. Y la decisión más rentable de todo el proyecto fue no migrar los catorce servicios que ya funcionaban.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-17-aws-production-architecture/213-eks-irsa-gitops-y-operacion-de-cluster/lab.py
```

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

Evidencia esperada

El artefacto principal es manifiestos declarativos con estado observado. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye aws-eks-gitops 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 pods se reinician sin dejar rastro en sus registros	El límite de memoria está ajustado al uso medio y un pico mata el proceso	Fija la petición en el p95 observado y el límite entre 1,5 y 2 veces la petición; alerta sobre terminaciones por memoria.
Latencia alta con los nodos ociosos y escalar no ayuda	Límite de CPU bajo que provoca estrangulamiento del contenedor	Fija la petición de CPU en el p95 y considera no poner límite; vigila la métrica de estrangulamiento.
Cualquier pod puede usar los permisos de otro equipo	Los permisos están en el rol del nodo o la condición del rol solo ata al emisor	Usa identidad de carga con condición por espacio de nombres y cuenta de servicio, y bloquea el acceso al servicio de metadatos.
Los cambios del repositorio dejan de aplicarse sin que nadie lo note	El bucle de reconciliación falla en silencio	Alerta por antigüedad de la sincronización y por diferencia persistente entre lo declarado y lo real.
Una actualización rompe producción pese a haber pasado en pruebas	El clúster de prueba no era idéntico y faltaban complementos	Genera el clúster de prueba del mismo repositorio, mantén una matriz de compatibilidad de complementos y revisa las interfaces retiradas antes.
Un mantenimiento deja un servicio sin réplicas disponibles	No hay presupuesto de interrupción ni reparto entre zonas	Declara presupuestos de interrupción y restricciones de reparto por zona.

■ 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. ¿En qué casos no compensa Kubernetes frente a orquestación gestionada más simple?
2. ¿Por qué dar permisos al rol del nodo es un problema de alcance?
3. ¿Qué diferencia hay entre petición y límite, y qué ocurre al superar cada uno?
4. ¿Qué alerta detecta que la reconciliación ha dejado de funcionar?
5. ¿Qué hay que comprobar antes de actualizar la versión del clúster?

Referencias

- AWS (2025). IAM roles for service accounts (IRSA). <https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts.html>
- AWS (2025). Amazon EKS Kubernetes version support and upgrade. <https://docs.aws.amazon.com/eks/latest/userguide/kubernetes-versions.html>
- Kubernetes (2025). Managing resources for containers: requests and limits. <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>
- Argo CD (2025). GitOps and application health. <https://argo-cd.readthedocs.io/en/stable/>
- AWS (2025). EKS Best Practices Guide. <https://docs.aws.amazon.com/eks/latest/best-practices/introduction.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 17 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 212 · ECR, ECS Fargate, ALB y autoscaling	Parte 17 · Programa	214 · Budgets, Cost Explorer, etiquetado y FinOps automatizado →

Evaluación — 213 EKS, IRSA, GitOps y operación de clúster

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 aws-eks-gitops 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.

214 — Budgets, Cost Explorer, etiquetado y FinOps automatizado

Parte: 17 — AWS: arquitectura, automatización y operación en producción Nivel: avanzado · Horas estimadas: 4
Laboratorio: finops · Estado: EXECUTABLECORE

Propósito

Controlar el coste de lo que se ha montado en esta parte, donde ya han aparecido tres partidas que nadie había estimado. La clase fija el etiquetado como requisito de creación y no como limpieza posterior, explica cómo se lee una factura para encontrar el dinero de verdad, y automatiza lo que funciona: presupuestos con acción, detección de anomalías y retirada de lo que nadie usa. Con la advertencia de siempre: la policía del coste se rodea; el carril fácil, no.

Resultados de aprendizaje

Al finalizar podrás:

1. Imponer etiquetado en la creación, con barrera y no con auditoría.
2. Leer la factura para encontrar las partidas que dominan.
3. Configurar presupuestos y detección de anomalías que actúen.
4. Atribuir el coste por servicio y por unidad de negocio.
5. Retirar lo que no se usa, de forma automática y segura.

Conceptos centrales

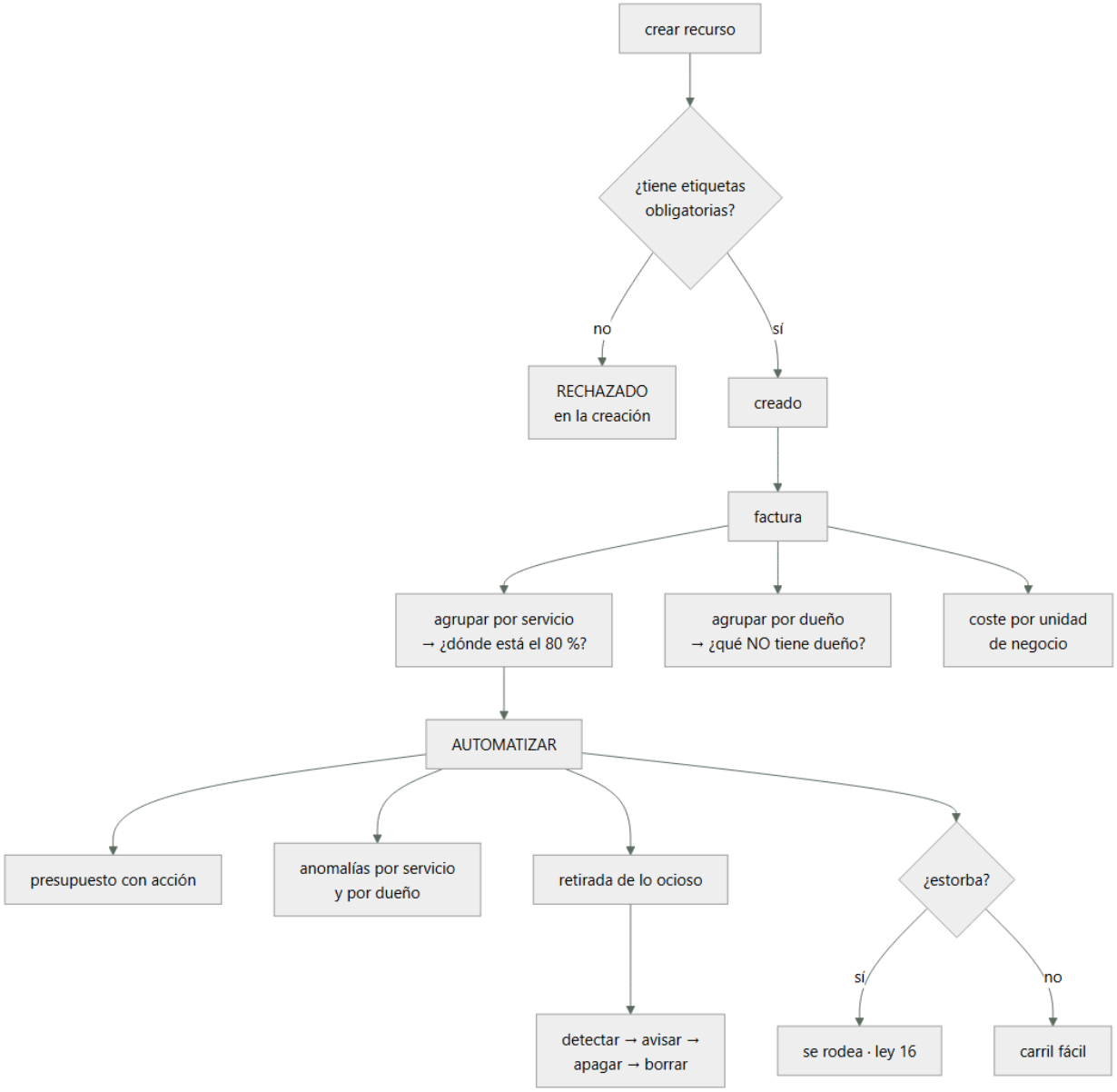
Concepto	Comprensión verificable
etiqueta de asignación	Etiqueta activada para que aparezca como dimensión en el informe de costes. Sin activar, no sirve.
coste por unidad de negocio	Coste dividido entre pedidos, usuarios o transacciones. Es la única cifra comparable en el tiempo.
presupuesto con acción	Presupuesto que además de avisar ejecuta algo: notificar, restringir o apagar.
detección de anomalías	Modelo que compara el gasto con el patrón previo y avisa de desviaciones.
coste no atribuido	Gasto que no cae en ningún servicio ni dueño. Es la parte que nadie reduce.
recurso ocioso	Recurso creado, facturado y sin uso. La categoría más fácil de eliminar y la que más se repite.

Modelo mental

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

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. Etiquetar en la creación, no después

El proyecto de etiquetado que empieza etiquetando lo que ya existe fracasa siempre: mientras se etiqueta lo viejo, se crea lo nuevo sin etiquetas.

```
EL ORDEN CORRECTO
  1  decidir el conjunto MÍNIMO de etiquetas obligatorias
  2  impedir la creación sin ellas
  3  y solo entonces, etiquetar lo existente

→ al revés, nunca se termina                        ley 25
```

Y el conjunto mínimo, que debe ser corto:

```
dueño         el equipo, no una persona          ley 20
servicio      a qué sistema pertenece
entorno       producción, preproducción, desarrollo
centro de coste  a quién se le imputa

→ cuatro. Más de seis y nadie las pone bien
→ y con valores CERRADOS, no texto libre:
  «prod», «Prod», «producción» y «PRD» son cuatro grupos
  distintos en el informe
```

Cómo se impide la creación, por orden de fuerza:

```
1  BARRERA DE ORGANIZACIÓN
   deniega crear sin las etiquetas
   → ni un administrador de la cuenta puede saltárselo
                                                clase 169

2  POLÍTICA DE ETIQUETAS
   fija los valores permitidos por clave

3  FUNCIÓN DE APTITUD EN LA CANALIZACIÓN
   la plantilla que no las declara no despliega
                                                clase 190

4  LA PLANTILLA DE SERVICIO NUEVO LAS TRAE HECHAS
   → esto es lo que hace que casi nunca falle nada
                                                clase 171
```

Y la advertencia de la parte 14:

```
si poner las etiquetas cuesta trabajo, la gente creará los
recursos por otro camino, o pondrá valores basura para pasar
el control                                        ley 16
→ el carril fácil las pone solo; la barrera es la red de
  seguridad, no el mecanismo principal
```

Y lo que no se puede etiquetar, que hay que tratar aparte:

```
transferencia de datos entre zonas y regiones
coste compartido de servicios comunes
tráfico de salida sin recurso asociado
→ estos se reparten por regla, escrita y acordada
→ y hay que decir cuánto se reparte y con qué criterio
```

2. Leer la factura

La factura se mira mal: se abre el total, se dice «ha subido» y se cierra. Estas son las cinco vistas que encuentran el dinero.

```
1  POR SERVICIO, ordenado de mayor a menor
   → el 80 % del gasto suele estar en 3 o 4 líneas
   → y esas son las únicas que merece la pena optimizar

2  POR DUEÑO
   → y sobre todo: cuánto NO tiene dueño
   → el coste no atribuido no lo reduce nadie      ley 20

3  POR ENTORNO
   → si preproducción cuesta más del 25 % de producción,
     hay algo encendido que no debería

4  POR TIPO DE USO dentro del servicio grande
   no basta «almacenamiento: 4.100 €»
   hace falta: peticiones, almacenamiento por clase,
   transferencia, recuperaciones
   → el desglose es donde está la sorpresa

5  COSTE POR UNIDAD DE NEGOCIO
   coste / pedidos del mes
   → es la ÚNICA cifra comparable entre meses
   → sin ella, «el coste ha subido un 20 %» no dice si
     mejoró o empeoró                            clase 142
```

Y las partidas que este programa ha visto sorprender, ya en esta parte:

```
peticiones de la pasarela         > cómputo    clase 207
ingesta de registros              > cómputo    clase 211
escrituras de índices             > tabla      clase 208
transferencia entre zonas         casi siempre olvidada
series de métricas de alta
  cardinalidad                     1.410 €/mes  clase 211
invalidaciones de caché                         clase 205
capas huérfanas del registro de
  imágenes                         210 €/mes    clase 212
```

Y una técnica que ahorra tiempo:

```
no persigas el porcentaje: persigue el importe
  «este servicio subió un 400 %»  → eran 3 € y ahora son 15
  «este subió un 6 %»             → eran 12.000 €
→ ordena siempre por importe de la variación, no por
  porcentaje
```

Los descuentos por compromiso, con la parte que se hace mal:

```
se compromete SOLO la base estable, medida con meses de
histórico
  → comprometer el pico deja capacidad pagada sin usar
  → y comprometer a 3 años algo que cambiará en 1 es peor
    que no comprometer nada

y hay que vigilar la COBERTURA y la UTILIZACIÓN
  cobertura     qué proporción del uso está cubierta
  utilización   qué proporción de lo comprometido se usa
  → utilización por debajo del 95 % es dinero tirado
```

3. Automatizar lo que funciona

Los informes que alguien debe mirar dejan de mirarse. Lo que funciona es lo que actúa.

Presupuestos con acción:

```
POR CUENTA Y POR SERVICIO, no solo global
  → un presupuesto global se supera y nadie sabe por qué

CON UMBRALES ESCALONADOS
  50 %, 80 %, 100 % del previsto para la fecha
  y sobre PREVISIÓN, no solo sobre gasto acumulado
  → avisar al 100 % el día 28 no sirve de nada

CON ACCIONES REALES en entornos no productivos
  al 100 % en desarrollo → restringir la creación de
    recursos nuevos
  al 120 % → apagar lo que se pueda apagar
  → en producción, avisar; nunca apagar automáticamente
```

Detección de anomalías, que encuentra lo que el presupuesto no:

```
el presupuesto detecta que se gasta de más EN TOTAL
la anomalía detecta que un servicio concreto se ha salido
de su patrón, aunque el total esté bien

configurarla por servicio Y por dueño
umbral en importe absoluto, no en porcentaje
  → si no, avisa de cada servicio pequeño que se dobla
y con destino a alguien que actúe                clase 211
```

La retirada de lo ocioso, que es donde está el dinero fácil:

```
LO QUE SIEMPRE APARECE
  volúmenes desconectados de cualquier máquina
  copias antiguas de volúmenes, sin caducidad
  direcciones fijas reservadas y sin asociar
  balanceadores sin destinos
  instancias paradas cuyo disco se sigue pagando
  bases de datos de pruebas de hace un año
  entornos efímeros que no caducaron              ley 25
  puntos privados creados y no usados            clase 200
  registros sin caducidad                        clase 211
  imágenes sin etiqueta                          clase 212

EL PROCESO SEGURO, en cuatro pasos
  1  DETECTAR y listar, con dueño si lo hay
  2  AVISAR al dueño, con plazo
  3  APAGAR o desasociar, sin borrar, y esperar
  4  BORRAR si nadie se ha quejado

→ el paso 3 es el que evita el desastre: si algo se rompe,
  se vuelve a encender en segundos
```

Y dos automatismos que dan mucho por poco:

```
APAGADO POR HORARIO en entornos no productivos
  desarrollo y preproducción apagados por la noche y el
  fin de semana
  → 168 h/semana → 50 h/semana = 70 % menos
  → y el arranque debe ser automático, o la gente
    desactivará el apagado                          ley 16

CADUCIDAD EN LOS ENTORNOS EFÍMEROS
  nacen con fecha de destrucción
  → sin ella, quedan para siempre                    ley 25
```

4. Que no se convierta en policía

La parte 14 ya mostró qué pasa cuando el control de coste se plantea como vigilancia: se rodea.

```
LO QUE NO FUNCIONA
  el informe mensual de «los que más gastan»
  pedir justificación de cada recurso
  bloquear sin explicar
  → produce recursos creados por caminos raros y valores
    de etiqueta inventados                          ley 16

LO QUE SÍ
  que el equipo vea SU coste, en su panel, junto a sus
    demás señales                                  clase 211
  que la plantilla por defecto ya sea la barata
  que el coste por unidad de negocio sea el indicador, no
    el total
  → así el equipo que crece un 40 % no aparece como
    culpable
  y que la retirada de lo ocioso la haga la plataforma, no
    el equipo
```

Y la medida que dice si el sistema funciona:

```
PROPORCIÓN DE COSTE ATRIBUIDO
  ¿qué porcentaje del gasto tiene dueño identificable?
  → por debajo del 90 %, el resto de los esfuerzos son
    ruido
  → en la clase 179 pasó del 38 % al 96 %

y la segunda: COSTE POR UNIDAD DE NEGOCIO, su tendencia
```

Lo que hay que vigilar:

```
gasto diario frente a previsión, por cuenta
coste no atribuido, y su tendencia
coste por unidad de negocio
cobertura y utilización de compromisos
recursos ociosos detectados y retirados
anomalías abiertas
```

Y la lista de comprobación de la clase:

- hay cuatro o cinco etiquetas obligatorias, con valores cerrados
- están activadas como etiquetas de asignación
- no se puede crear un recurso sin ellas
- la plantilla de servicio nuevo las trae hechas
- el coste compartido se reparte por una regla escrita
- se mira el desglose dentro de los servicios grandes
- se ordena por importe de variación, no por porcentaje
- hay coste por unidad de negocio y se sigue su tendencia
- hay presupuestos por cuenta y por servicio, sobre previsión
- en entornos no productivos, los presupuestos actúan
- hay detección de anomalías por servicio y por dueño
- hay apagado por horario en no producción, con arranque automático
- los entornos efímeros nacen con fecha de destrucción
- la retirada de lo ocioso pasa por apagar antes de borrar
- solo se compromete la base estable, y se vigila la utilización
- el coste atribuido supera el 90 %

Y el cierre que enlaza con la clase siguiente: con el sistema construido, protegido, observado y con su coste bajo control, queda lo que este programa exige siempre antes de dar algo por terminado: comprobar que sobrevive a que se caiga una región, y comprobarlo ejecutándolo. Es la materia de la clase 215.

Ejemplo trabajado

CloudShop revisa el coste de la plataforma que ha montado en esta parte. Lo que sigue es la factura desglosada, las cuatro partidas que nadie había estimado, y la automatización que redujo el gasto un 41 % sin tocar la arquitectura.

La factura del primer mes completo:

```
total                                    28.400 €

por servicio, ordenado
  cómputo de contenedores                 5.830   20,5 %
  base de datos gestionada                4.210   14,8 %
  transferencia de datos                  3.940   13,9 %  ←
  registros y métricas                    3.120   11,0 %  ←
  almacenamiento de objetos               2.680    9,4 %
  pasarela de API                         2.140    7,5 %  ←
  DynamoDB                                1.920    6,8 %
  balanceadores                           1.310    4,6 %
  funciones                                 980    3,5 %
  registro de imágenes                      610    2,1 %  ←
  resto                                   1.660    5,9 %

por dueño
  con dueño identificable                17.100   60,2 %
  SIN DUEÑO                              11.300   39,8 %  ←

por entorno
  producción                             16.900
  preproducción                           7.200   43 % de
                                                  producción
  desarrollo                              3.100
  sin etiquetar                           1.200
```

Y las tres lecturas inmediatas:

```
el cómputo, que era lo que se vigilaba, es el 20 %
cuatro de las diez partidas mayores no se habían estimado
y el 40 % del gasto no tiene dueño                ley 20
```

Las cuatro partidas no estimadas, desglosadas.

```
TRANSFERENCIA DE DATOS · 3.940 €
  salida a internet                          1.100
  ENTRE ZONAS                                2.190   ←
  entre regiones                               410
  otros                                        240

  el tráfico entre zonas venía de
    · el clúster hablando con la base sin preferencia de
      zona
    · réplicas de servicio repartidas y balanceo que
      ignoraba la zona
  corrección   preferencia de zona en el balanceo interno
               y lectura desde la réplica de la misma zona
  2.190 € → 640 €

REGISTROS Y MÉTRICAS · 3.120 €
  ya se había reducido de 4.980 en la clase 211, y quedaba
  el clúster
    ingesta del clúster                      1.840
    → cada contenedor escribía a nivel de depuración
  corrección   nivel por espacio de nombres, muestreo y
               caducidad de 14 días
  3.120 € → 780 €

PASARELA DE API · 2.140 €
  la migración de REST a HTTP de la clase 207 se había
  hecho solo en un servicio
  → 3 servicios seguían en REST sin usar nada de lo suyo
```

```
  2.140 € → 810 €

REGISTRO DE IMÁGENES · 610 €
  ya diagnosticado en la clase 212: capas huérfanas
  610 € → 24 €
```

El coste sin dueño: qué era.

```
11.300 € sin dueño identificable

  al inventariar
    transferencia de datos (no etiquetable)       3.940
    servicios comunes: nombres, identidad,
      registro central                            1.870
    recursos con etiquetas mal escritas           2.410
      «prod» / «Prod» / «producción» / «PRD»
      → cuatro grupos para un entorno
    RECURSOS OCIOSOS                              2.180  ←
    recursos creados antes de la política           900

  los ociosos, detallados
    41 volúmenes desconectados                      610 €
    118 copias de volúmenes sin caducidad           480 €
    9 direcciones fijas sin asociar                  32 €
    4 balanceadores sin destinos                    118 €
    2 bases de datos de pruebas de 2024             740 €
    31 entornos efímeros no destruidos              200 €
```

Y el detalle que resume el hallazgo:

```
las dos bases de datos de pruebas llevaban 14 meses
encendidas
la más cara la creó alguien que ya no está en la empresa
y nadie la reclamó al apagarla                ley 20, ley 25
```

Lo que se montó.

```
ETIQUETADO
  cinco etiquetas obligatorias: dueño, servicio, entorno,
  centro de coste, caducidad (opcional pero recomendada)
  valores CERRADOS por política de etiquetas
    entorno ∈ {prod, pre, dev, sandbox}
  barrera de organización: sin etiquetas, no se crea
  plantilla de servicio nuevo: las trae hechas

  y la corrección de lo existente
    2.410 € de etiquetas mal escritas → normalizadas en
    una semana con un script

PRESUPUESTOS
  por cuenta y por servicio, sobre previsión
  umbrales 50 / 80 / 100 %
  en desarrollo y preproducción
    al 100 %  → se restringe la creación de recursos nuevos
    al 120 %  → se apagan las cargas apagables
  en producción, solo aviso

ANOMALÍAS
  por servicio y por dueño, umbral de 150 € absolutos
  destino: el canal de guardia                 clase 211
  → en 6 meses, 11 anomalías
    · 7 reales (una prueba de carga olvidada, un trabajo
      en bucle, un índice nuevo, 4 crecimientos legítimos)
    · 4 falsas, por campañas previstas

APAGADO POR HORARIO
  desarrollo y preproducción: apagados de 20:00 a 7:00 y
  los fines de semana
  arranque automático a las 7:00, y bajo demanda con un
  botón que tarda 4 minutos
  → el botón fue lo que evitó que la gente desactivara el
    apagado                                      ley 16
  7.200 + 3.100 = 10.300 € → 4.180 €

ENTORNOS EFÍMEROS
```

```
  nacen con caducidad de 7 días, ampliable a 30 con motivo
  → los 31 existentes se destruyeron tras avisar

RETIRADA DE LO OCIOSO, automatizada
  detección semanal
  aviso al dueño con 7 días de plazo
  desasociar o apagar (sin borrar) y esperar 14 días
  borrar si nadie reclama
  → en 6 meses: 214 recursos retirados, 3 reclamaciones,
    3 restauraciones en menos de 5 minutos
```

El resultado, a los tres meses:

```
total mensual                            28.400 €   16.700 €
  transferencia                           3.940 €      920 €
  registros y métricas                    3.120 €      780 €
  pasarela                                2.140 €      810 €
  registro de imágenes                      610 €       24 €
  no productivos                         10.300 €    4.180 €
  recursos ociosos                        2.180 €        0 €
coste atribuido                            60,2 %     94,1 %
coste por pedido                          0,082 €    0,046 €
etiquetas con valores inconsistentes        2.410 €      0 €
```

Y la comprobación de que no se había convertido en policía:

```
recursos creados por caminos alternativos
  antes de la barrera (estimado)                desconocido
  después, detectados por inventario                      0

quejas del equipo sobre el control de coste
  al implantar                                            6
  a los 3 meses                                           0
  → las 6 se resolvieron con el botón de arranque bajo
    demanda y con ampliar la caducidad de los entornos
    efímeros de 3 a 7 días

y la señal que se vigila
  «número de excepciones vivas a la política de etiquetas»
  4, todas con dueño y fecha                    clase 190
```

La lección que esta clase deja: el cómputo, que era lo único que se vigilaba, resultó ser el 20 % de la factura, y cuatro de las diez partidas mayores no se habían estimado. El 40 % del gasto no tenía dueño y dos mil ciento ochenta euros al mes eran recursos que no usaba nadie, incluidas dos bases de datos de pruebas encendidas catorce meses. Y del ahorro total, la medida que más aportó no fue ninguna optimización técnica: fue apagar por la noche lo que no es producción.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-17-aws-production-architecture/214-budgets-cost-explorer-etiquetado-y-finops-automatizado/lab.py
```

El laboratorio selecciona el motor de práctica finops 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 aws-cost-control 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 cálculo trazable con unidad, supuesto y sensibilidad. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye aws-cost-control 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 proyecto de etiquetado nunca termina	Se empezó etiquetando lo existente mientras se seguía creando sin etiquetas	Impide primero la creación sin etiquetas y solo después corrige lo antiguo.
El informe de costes muestra el mismo entorno repartido en varios grupos	Las etiquetas admiten texto libre	Fija valores cerrados por política de etiquetas y normaliza lo existente.
Se optimiza el cómputo y la factura no baja	El cómputo no era la partida dominante	Ordena la factura por importe, desglosa dentro de los servicios grandes y ataca las tres o cuatro líneas que suman el ochenta por ciento.
Nadie reduce una parte importante del gasto	Ese gasto no tiene dueño identificable	Mide la proporción de coste atribuido y reparte lo no etiquetable con una regla escrita; por debajo del 90 % lo demás es ruido.
El presupuesto avisa cuando ya no se puede hacer nada	Está definido sobre gasto acumulado y no sobre previsión	Configura umbrales sobre la previsión de cierre y añade acciones reales en entornos no productivos.
El control de coste genera recursos creados por caminos raros	Se planteó como vigilancia y no como camino fácil	Que la plantilla por defecto ya sea la barata, que cada equipo vea su coste junto a sus señales y que la retirada la haga la plataforma.

■ 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 hay que impedir la creación sin etiquetas antes de etiquetar lo existente?
2. ¿Qué cinco vistas de la factura encuentran el dinero de verdad?
3. ¿Por qué se ordena por importe de la variación y no por porcentaje?
4. ¿Qué detecta la detección de anomalías que no detecta el presupuesto?
5. ¿Qué paso del proceso de retirada evita convertir una limpieza en un incidente?

Referencias

- AWS (2025). Cost allocation tags and tag policies. <https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/cost-alloc-tags.html>
- AWS (2025). AWS Budgets and budget actions. <https://docs.aws.amazon.com/cost-management/latest/userguide/budgets-managing-costs.html>
- AWS (2025). AWS Cost Anomaly Detection. <https://docs.aws.amazon.com/cost-management/latest/userguide/manage-ad.html>
- FinOps Foundation (2025). FinOps Framework: capabilities and maturity. <https://www.finops.org/framework/>
- AWS (2025). Savings Plans and Reserved Instances: coverage and utilization. <https://docs.aws.amazon.com/savingsplans/latest/userguide/what-is-savings-plans.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 17 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 213 · EKS, IRSA, GitOps y operación de clúster	Parte 17 · Programa	215 · Multi-región, Route 53, failover y game day →

Evaluación — 214 Budgets, Cost Explorer, etiquetado y FinOps automatizado

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 aws-cost-control 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.

215 — Multi-región, Route 53, failover y game day

Parte: 17 — AWS: arquitectura, automatización y operación en producción Nivel: avanzado · Horas estimadas: 4
Laboratorio: reliability · Estado: EXECUTABLECORE

Propósito

Comprobar que el sistema sobrevive a la pérdida de una región, y comprobarlo ejecutándolo. La clase cubre los modos de despliegue multirregión con su coste real, la conmutación por nombres y sus límites —que casi siempre son el cuello—, la replicación de datos con su decisión de consistencia, y el ejercicio dirigido: cómo se prepara, cómo se ejecuta y por qué la mayoría de lo que encuentra no es de infraestructura.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir el modo multirregión que corresponde al objetivo, con su coste.
2. Conmutar por nombres conociendo el retraso real, no el teórico.
3. Replicar datos decidiendo qué se pierde y qué se bloquea.
4. Preparar y ejecutar un ejercicio dirigido con seguridad.
5. Corregir lo que el ejercicio encuentre, que rara vez es lo previsto.

Conceptos centrales

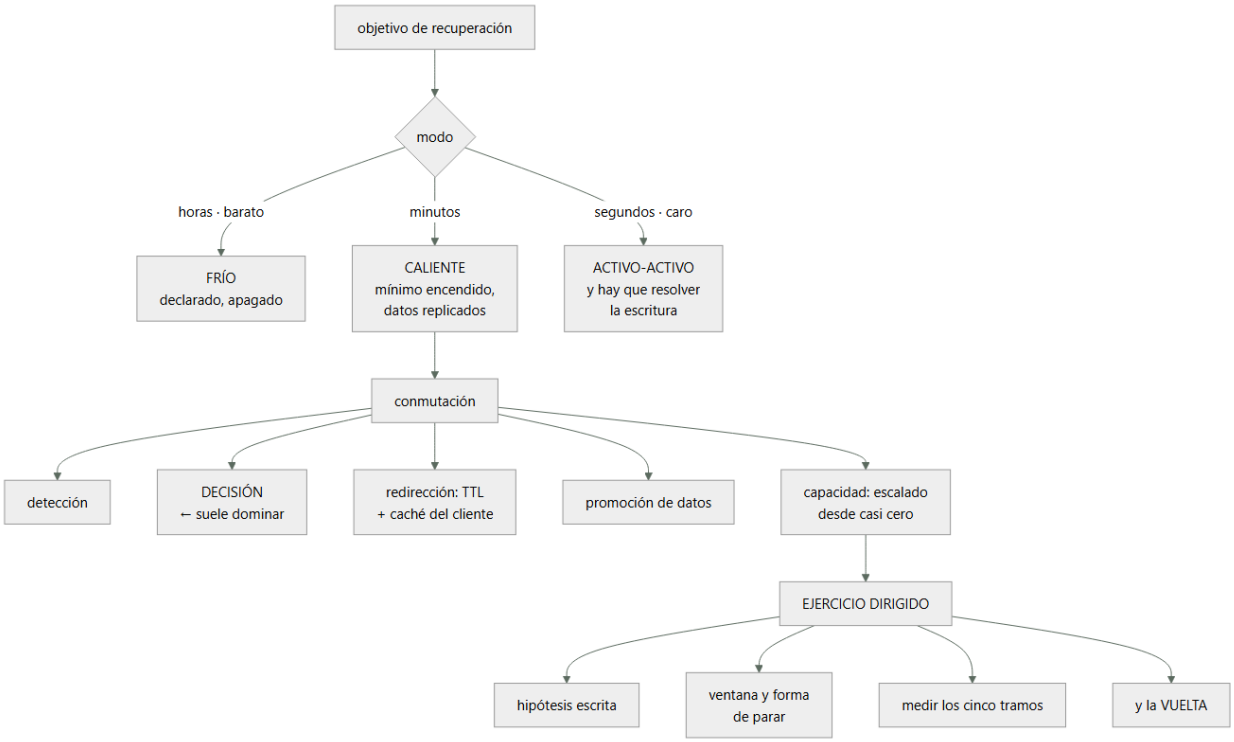
Concepto	Comprensión verificable
activo-pasivo en frío	La segunda región existe declarada y apagada. Barata y lenta de activar.
activo-pasivo en caliente	La segunda región tiene capacidad mínima encendida y datos replicados.
activo-activo	Ambas regiones sirven tráfico. Caro, y obliga a resolver la escritura en dos sitios.
comprobación de salud del nombre	Sonda que decide si un destino se anuncia. Su intervalo y umbrales fijan el tiempo de conmutación.
ejercicio dirigido	Simulación planificada de un fallo real, con hipótesis escrita, ventana acordada y forma de parar.
hipótesis del ejercicio	Lo que se espera que ocurra, escrito antes. Lo valioso es dónde falla.

Modelo mental

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

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. Modos multirregión y su coste

La elección se hace por el objetivo de recuperación, y el objetivo se decide con el coste de la indisponibilidad, no con lo que suene bien.

```
FRÍO
  la infraestructura está declarada como código y no
  desplegada; los datos, en copias replicadas
  plazo   horas
  coste   almacenamiento de copias y poco más
  riesgo  el despliegue completo nunca se ha ejecutado
          → y es exactamente lo que falla         ley 22

CALIENTE (piloto)
  capacidad mínima encendida, datos replicados en continuo
  plazo   minutos
  coste   30-40 % de una región completa
  riesgo  la capacidad mínima tarda en escalar al 100 %

ACTIVO-ACTIVO
  ambas regiones sirven
  plazo   segundos
  coste   más del doble de una región
  riesgo  hay que resolver la escritura en dos sitios, que
          es el problema difícil                clase 187
```

Y la decisión, con la aritmética de siempre:

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

y se compara con el coste del modo siguiente
→ si pasar de caliente a activo-activo cuesta 6.400 €/mes y
  evita 390 € de pérdida esperada, no se compra
                                      clases 164, 185
```

Y la advertencia sobre activo-activo, que se propone más de lo que se justifica:

```
ACTIVO-ACTIVO OBLIGA A DECIDIR DÓNDE SE ESCRIBE
  un escritor global      → latencia alta desde la otra
                            región
```

```
  escritura por región    → hay que particionar los datos
                            por región y que no se crucen
  escritura en ambas      → conflictos, y hay que
                            resolverlos              ley 21

→ la mayoría de los sistemas que dicen ser activo-activo
  son activo-pasivo con lectura en las dos
```

Y lo que hay que replicar además de los datos:

```
imágenes de contenedor y artefactos            clase 212
secretos y claves de cifrado
configuración y banderas de función
certificados
reglas de filtrado
y la propia canalización, capaz de desplegar allí
→ una región secundaria sin registro de imágenes propio no
  puede arrancar nada                           clase 185
```

2. Conmutar por nombres, con los límites reales

La conmutación por nombres es el mecanismo habitual y su tiempo real casi nunca coincide con el previsto.

```
LOS COMPONENTES DEL RETRASO

1  DETECCIÓN
   comprobación de salud: intervalo × fallos consecutivos
   típico 30 s × 3 = 90 s
   → configurable, y es lo más fácil de reducir

2  PROPAGACIÓN DEL CAMBIO
   el servicio de nombres reevalúa y deja de anunciar

3  TTL DEL REGISTRO
   los resolutores mantienen la respuesta hasta que expira
   → si el TTL es 300 s, hasta 5 minutos       clase 195

4  CACHÉ DEL CLIENTE
   navegadores, bibliotecas y entornos de ejecución cachean
   por su cuenta
   → algunos entornos cachean para siempre dentro del
     proceso
   → es el tramo que NADIE mide y que en la clase 195
     retrasó 40 minutos una conmutación

5  CAPACIDAD Y DATOS EN DESTINO
   escalar desde capacidad mínima: minutos   clase 212
   promover la réplica de datos a primaria
```

Y la conclusión práctica:

```
el TTL debe ser menor que el plazo prometido   clase 195
y aun así, el plazo real lo fija la capa que MÁS cachea
→ por eso el plazo se MIDE, no se calcula
```

Lo que hay que comprobar en la comprobación de salud, que suele estar mal:

```
✗ comprueba que el balanceador responde
  → responde aunque la base esté caída
✓ comprueba un punto que ejerza el camino crítico
  → incluida la base, con una consulta ligera

✗ una sola sonda desde un sitio
✓ varias, desde regiones distintas, con acuerdo por mayoría
  → evita conmutar por un problema de red local
```

Y dos decisiones que evitan conmutaciones falsas:

```
HISTÉRESIS
  salir con pocos fallos, volver con muchos aciertos
  → evita ir y venir                            clase 196

CONMUTACIÓN MANUAL DISPONIBLE
  un interruptor que fuerza el cambio, y otro que lo
  impide
  → durante un incidente confuso, decidir a mano suele ser
```

```
    mejor que dejar que un umbral decida
```

Y el tramo que domina en la práctica:

```
en la clase 179, la conmutación tardaba 2 h 10 frente a la
hora declarada
  y los dos tramos que dominaban eran DECIDIR y REDIRIGIR,
  no arrancar nada
→ la decisión humana es parte del plazo, y hay que
  cronometrarla                                  clase 166
```

3. Datos: qué se pierde y qué se bloquea

La parte difícil de multirregión son los datos, y la decisión es siempre la misma: qué se pierde al conmutar y qué se bloquea para no perder nada.

```
REPLICACIÓN ASÍNCRONA
  la escritura confirma en la región primaria y viaja
  después
  + latencia de escritura baja
  − al conmutar se pierde lo que no llegó a viajar
  → el objetivo de pérdida es el retraso de replicación
  → y ese retraso hay que VIGILARLO con alerta  clase 161

REPLICACIÓN SÍNCRONA ENTRE REGIONES
  la escritura confirma cuando llega a las dos
  + no se pierde nada
  − 60-150 ms añadidos a CADA escritura
  → rara vez compensa; es lo que se descartó en la
    clase 187

TABLAS GLOBALES (multi-escritor)
  se escribe en cualquier región y se replica
  + latencia baja en todas
  − resolución de conflictos por «el último gana», con las
    trampas de los relojes                      clase 187
  → solo sirve si los conflictos son imposibles por diseño:
    cada dato lo escribe una sola región         ley 21
```

Y la decisión que hay que tomar y escribir:

```
por cada conjunto de datos
  ¿cuánto se puede perder?     → fija el modo de replicación
  ¿se puede escribir en la
   secundaria?                 → si no, hay que bloquear
  ¿qué pasa con lo que estaba
   a medias?                   → reconciliación   clase 203
```

La vuelta, que es donde se pierden datos de verdad:

```
CONMUTAR ES LA MITAD FÁCIL
  volver exige que la región original se ponga al día con
  lo escrito en la secundaria
  y durante ese tiempo NO puede haber dos escritores
                                                  ley 21

EL ERROR CLÁSICO
  la región original vuelve, la comprobación de salud pasa,
  el nombre la vuelve a anunciar
  → y durante unos minutos se escribe en las dos
  → datos divergentes, imposibles de reconciliar

LA CORRECCIÓN
  la vuelta es SIEMPRE manual y con pasos
  1  bloquear escritura en la secundaria
  2  esperar a que la original se sincronice
  3  promover la original
  4  redirigir
  5  desbloquear
  → y el paso 1 exige que exista un modo de solo lectura
```

Y una comprobación que hay que hacer antes:

```
¿el sistema tiene modo de solo lectura?
  → si no, no se puede conmutar de vuelta con seguridad
  → y conviene tenerlo también para el mantenimiento
```

4. El ejercicio dirigido

Un plan de continuidad no comprobado no funciona. Este programa lo ha demostrado en la clase 166 y en la 179.

La preparación, que es la mitad del trabajo:

```
HIPÓTESIS ESCRITA, antes
  «al perder la región primaria, el tráfico conmutará en
   menos de 15 minutos, con pérdida menor de 1 minuto de
   datos, y las funciones X e Y quedarán degradadas»
  → lo valioso del ejercicio es dónde falla la hipótesis

ALCANCE Y VENTANA
  qué se rompe exactamente, y qué no se toca
  ventana acordada con negocio, no de madrugada la primera
    vez
  → de madrugada no hay nadie para observar y no hay
    tráfico realista

FORMA DE PARAR
  un botón que restaura el estado, decidido antes
  y un criterio claro: «si la pérdida supera X, se para»

OBSERVACIÓN
  quién mira qué, y qué cifras se anotan
  y una persona SOLO tomando notas

COMUNICACIÓN
  atención al cliente y negocio, avisados
  → un ejercicio que genera un incidente en el sistema de
    tiquetes ha fallado en algo más
```

La escala, que evita el desastre en el primer intento:

```
1  en papel: recorrer el procedimiento leyéndolo
   → encuentra los pasos que no existen
2  en un entorno de prueba
3  en producción, con una parte del tráfico
4  en producción, completo

→ empezar por el 4 es cómo se convierte un ejercicio en un
  incidente
```

Los cinco tramos que hay que cronometrar:

```
1  desde el fallo hasta la detección
2  desde la detección hasta la DECISIÓN
3  desde la decisión hasta la redirección efectiva
4  desde la redirección hasta la capacidad completa
5  desde ahí hasta la operación normal

y además
  la pérdida de datos real, medida
  y el tiempo de la VUELTA
```

Qué encuentra un ejercicio, con la evidencia del programa:

```
en la clase 179, de 4 pruebas de continuidad, 1 falló y la
conmutación tardó el doble de lo declarado
en la clase 198, la primera conmutación funcionó y la
replicación no se reanudó sola
en la clase 204, 6 de 15 pruebas fallaron y NINGUNA por
error de diseño

→ lo que encuentran los ejercicios casi nunca es
  infraestructura: son procedimientos, permisos, datos que
  faltan y decisiones que nadie sabía tomar    leyes 22, 24
```

Y la disciplina posterior:

```
cada hallazgo se convierte en una acción con dueño y fecha
el procedimiento se corrige EN EL MOMENTO, no después
y se repite el ejercicio hasta que pase entero
→ un ejercicio que encuentra problemas y no se repite ha
  servido para la mitad
```

Y la lista de comprobación de la clase:

- el modo multirregión corresponde al objetivo y su coste está comparado con la pérdida esperada
- imágenes, secretos, certificados y canalización están replicados
- la comprobación de salud ejerce el camino crítico
- hay sondas desde varias regiones y acuerdo por mayoría
- hay histéresis y conmutación manual disponible
- el TTL es menor que el plazo prometido
- se ha medido cuánto cachea cada cliente
- el retraso de replicación tiene alerta
- existe modo de solo lectura
- el procedimiento de vuelta es manual y con pasos
- el ejercicio tiene hipótesis escrita y forma de parar
- se cronometran los cinco tramos y la pérdida real
- los hallazgos tienen dueño y fecha
- el ejercicio se repite hasta pasar entero

Y el cierre que enlaza con la clase siguiente: con el sistema construido, protegido, observado, con su coste controlado y su continuidad comprobada, queda ponerlo todo junto y llevarlo a producción de verdad. Es la materia de la clase 216, que además cierra la parte 17.

Ejemplo trabajado

CloudShop hace su primer ejercicio dirigido de pérdida de región. Lo que sigue es la hipótesis escrita, lo que ocurrió de verdad, y los ocho hallazgos —de los cuales seis no eran de infraestructura.

El montaje, antes del ejercicio:

```
modo              activo-pasivo en caliente
primaria          eu-west-1
secundaria        eu-central-1, al 15 % de capacidad
datos             base relacional con réplica asíncrona
                  DynamoDB con tabla global
                  objetos replicados
nombres           registro con comprobación de salud,
                  TTL de 60 s
objetivo declarado   conmutar en < 15 min, pérdida < 1 min
coste del modo       4.100 €/mes (frente a 11.800 € de
                     activo-activo)
```

La hipótesis, escrita tres días antes:

```
1  la comprobación de salud detectará el fallo en < 2 min
2  la conmutación por nombres se completará en < 5 min
3  la capacidad escalará al 100 % en < 8 min
4  la pérdida de datos será < 30 s
5  la búsqueda y las recomendaciones quedarán degradadas;
   el flujo de compra funcionará completo
6  el tiempo total será < 15 min
7  la vuelta se podrá hacer en < 30 min
```

La preparación:

```
escala    se hizo en papel (2 semanas antes) y en
          preproducción (1 semana antes)
          → el recorrido en papel encontró que el paso 4 del
            procedimiento no existía: nadie sabía quién
            promueve la base

ventana   martes, 14:00-17:00, con tráfico real del 30 % de
          un día normal
          → no de madrugada, a propósito

parada    restaurar el estado devolviendo el registro de
          nombres a la primaria; criterio de parada: pérdida
          de datos > 5 min o más de 20 min sin servicio

observación  6 personas: 1 ejecuta, 1 toma notas y
          cronometra, 4 observan sus áreas

aviso     atención al cliente, negocio y el socio principal
```

Lo que ocurrió, tramo a tramo:

```
14:00:00  se corta el tráfico a la primaria simulando el
          fallo

14:01:34  detección: la comprobación de salud marca la
          primaria como no sana
          previsto < 2 min                    ✓ 1 min 34 s

14:01:34  el registro deja de anunciar la primaria
14:09:20  el tráfico está mayoritariamente en la secundaria
          previsto < 5 min                    ✗ 7 min 46 s

          por qué
            el TTL era 60 s, correcto
            PERO la aplicación móvil, en su versión 4.1,
            cachea la resolución dentro del proceso hasta
            reiniciar
            → el 22 % del tráfico móvil siguió llegando a
              la primaria durante todo el ejercicio
            → nadie lo había medido                clase 195

14:04:10  se decide promover la base de datos
14:04:10  ...y nadie tiene permiso para hacerlo
14:11:45  se localiza a la persona con permiso
14:13:02  la base secundaria es promovida
          → 8 min 52 s perdidos en un permiso   ← hallazgo

14:13:02  pérdida de datos medida: 41 segundos
          previsto < 30 s                     ✗ por poco

14:14:00  el escalado empieza
14:26:30  capacidad al 100 %
          previsto < 8 min                    ✗ 12 min 30 s

          por qué
            la región secundaria no tenía las imágenes de
            contenedor más recientes: el registro replicaba
            con retraso y faltaban 2 de 9
            → 2 servicios tardaron en arrancar
                                       clases 212, 185

14:26:30  operación normal… casi
          la búsqueda NO funcionaba en absoluto
          previsto: degradada                 ✗ caída

          por qué
            el índice de búsqueda no se replicaba a la
            secundaria; nadie lo había incluido en el plan
            → y el flujo de compra SÍ dependía de la
              búsqueda para el listado inicial
            → dependencia declarada blanda que era dura
                                       clases 185, 201

TIEMPO TOTAL              26 min 30 s
previsto                  < 15 min                  ✗
```

La vuelta, que fue peor:

```
15:30  se inicia la vuelta
15:31  se restaura el tráfico a la primaria
15:31  la primaria estaba en pie y su comprobación de salud
       pasaba
       → el registro la volvió a anunciar automáticamente
       → y durante 6 minutos se escribió en las DOS
         regiones

15:37  se detecta al ver escrituras en ambas
15:37  parada del ejercicio; tráfico forzado a la
       secundaria
15:52  reconciliación manual: 214 pedidos escritos en la
       primaria durante esos 6 minutos, que la secundaria
       no tenía
       → recuperados con esfuerzo, ninguno perdido
```

```
        → pero en un incidente real, con más tráfico, esto
          habría sido grave                          ley 21

causa   no existía modo de solo lectura
        y el procedimiento de vuelta era automático, no
        manual con pasos
```

Los ocho hallazgos:

#	hallazgo	tipo	acción
1	permiso de promoción de base solo lo tenía una persona, no localizable	PERMISOS	3 personas con acceso de emergencia
2	procedimiento sin el paso de quién promueve	PROCESO	escrito y probado por otro
3	la app móvil 4.1 cachea la resolución para siempre	CLIENTE	corregido en 4.3; y medido
4	el registro de imágenes replicaba con retraso	DATOS	replicación síncrona de etiquetas de producción
5	el índice de búsqueda no se replicaba	DATOS	incluido en el plan
6	la búsqueda era dependencia dura del listado	DISEÑO	listado con respaldo cacheado
7	no existía modo de solo lectura	DISEÑO	implementado
8	la vuelta era automática y produjo dos escritores	PROCESO	vuelta manual, con 5 pasos

```
de infraestructura                                    2
de procedimiento, permisos, cliente y diseño          6
```

El segundo ejercicio, tres meses después:

```
detección                              1 min 12 s     ✓
conmutación efectiva                   3 min 40 s     ✓
  → tras corregir la app móvil, el tráfico residual
    bajó al 0,4 %
decisión y promoción                      58 s        ✓
  → con 3 personas con permiso y procedimiento escrito
pérdida de datos                             22 s     ✓
capacidad al 100 %                     6 min 10 s     ✓
búsqueda                                 degradada    ✓
  → el listado usa respaldo cacheado
TIEMPO TOTAL                          10 min 40 s     ✓

vuelta, manual con 5 pasos
  bloquear escritura en secundaria            40 s
  esperar sincronización                   4 min 20 s
  promover la original                        55 s
  redirigir                                2 min 10 s
  desbloquear                                 30 s
  total                                    8 min 35 s
  dos escritores simultáneos                     0
```

El coste, decidido con estos datos:

```
se volvió a plantear activo-activo
  coste adicional                        7.700 €/mes
  mejora del plazo              10 min 40 s → ~40 s
  pérdida esperada evitada, calculada con el histórico
  de incidentes regionales                410 €/mes

decisión   NO
registrado con la señal que lo reabriría
  «si el negocio de empresa con penalización por
   indisponibilidad supera el 15 % de los ingresos»
                                              clase 190
```

La lección que esta clase deja: la hipótesis falló en cinco de sus siete puntos, y el tramo que más tiempo consumió no fue técnico: fueron ocho minutos y cincuenta y dos segundos buscando a quien tuviera permiso para promover la base. De los ocho hallazgos, seis no eran de infraestructura. Y lo más peligroso ocurrió en la vuelta, que nadie había ensayado: seis minutos con dos escritores a la vez, que en un incidente real habrían dejado datos divergentes.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-17-aws-production-architecture/215-multi-region-route-53-failover-y-game-day/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 aws-dr-drill 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 aws-dr-drill 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 conmutación tarda mucho más de lo previsto pese a un TTL corto	Algún cliente cachea la resolución dentro del proceso y no respeta el TTL	Mide cuánto tarda cada cliente en seguir un cambio de nombre y corrige los que cachean indefinidamente.
La región secundaria no puede arrancar los servicios	Faltan imágenes, secretos, certificados o la propia canalización	Replica todo lo necesario para desplegar, no solo los datos, y compruébalo arrancando desde cero allí.
Al volver a la región original se escribe en las dos a la vez	La vuelta es automática y la comprobación de salud vuelve a anunciarla en cuanto está en pie	Haz la vuelta manual con pasos: bloquear escritura, sincronizar, promover, redirigir, desbloquear; y ten modo de solo lectura.
Durante la conmutación nadie puede ejecutar un paso crítico	El permiso lo tiene una sola persona y el procedimiento no lo nombra	Varias personas con acceso, procedimiento escrito y probado por alguien que no lo escribió.
Un servicio declarado degradable resulta imprescindible	La dependencia estaba declarada blanda y nunca se probó apagándola	Inyecta el fallo y comprueba; corrige con respaldo cacheado o valor por defecto antes del ejercicio siguiente.
El ejercicio se convierte en un incidente real	Se empezó por el escenario completo en producción, sin recorrido previo ni forma de parar	Escala el ejercicio: papel, prueba, parte del tráfico y completo; con criterio de parada y botón de restauración acordados.

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 distingue el modo caliente del activo-activo y qué problema añade el segundo?
2. ¿Qué cinco componentes forman el retraso de una conmutación por nombres?
3. ¿Por qué el plazo real se mide y no se calcula?
4. ¿Qué cinco pasos debe tener la vuelta y por qué es manual?
5. ¿Qué tipo de hallazgos produce un ejercicio dirigido con más frecuencia?

Referencias

- AWS (2025). Disaster recovery options in the cloud. <https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-options-in-the-cloud.html>
- AWS (2025). Route 53 health checks and DNS failover. <https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-failover.html>
- AWS (2025). Amazon DynamoDB global tables. <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GlobalTables.html>
- Google (2018). The Site Reliability Workbook: disaster role playing and testing. <https://sre.google/workbook/table-of-contents/>
- Basiri, A. y otros (2016). Chaos engineering — experimentos con hipótesis y radio de alcance. <https://ieeexplore.ieee.org/document/7503833>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 17 en PDF · Recorrido de AWS en PDF · Manual integral

Evaluación — 215 Multi-región, Route 53, failover y game day

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 aws-dr-drill 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.

216 — Proyecto: CloudShop productivo en AWS

Parte: 17 — AWS: arquitectura, automatización y operación en producción Nivel: avanzado · Horas estimadas: 8
Laboratorio: capstone · Estado: EXECUTABLECORE

Propósito

Poner en producción el sistema completo de CloudShop en AWS con lo de las once clases anteriores, y comprobarlo con las pruebas negativas de toda la parte. La clase da el orden de construcción, el entregable y los criterios. Y cierra la parte 17: corrige las cinco predicciones de la clase 204 —tres acertadas, una subestimada y una fallada—, actualiza el recuento de leyes, añade la ley 26 y escribe la hipótesis de la parte 18.

Resultados de aprendizaje

Al finalizar podrás:

1. Construir el sistema completo en el orden que evita rehacer.
2. Comprobar con las pruebas negativas de toda la parte.
3. Medir coste, latencia y disponibilidad con cifras comparables.
4. Corregir las cinco predicciones de la clase 204 con evidencia.
5. Escribir la hipótesis de la parte 18 en forma refutable.

Conceptos centrales

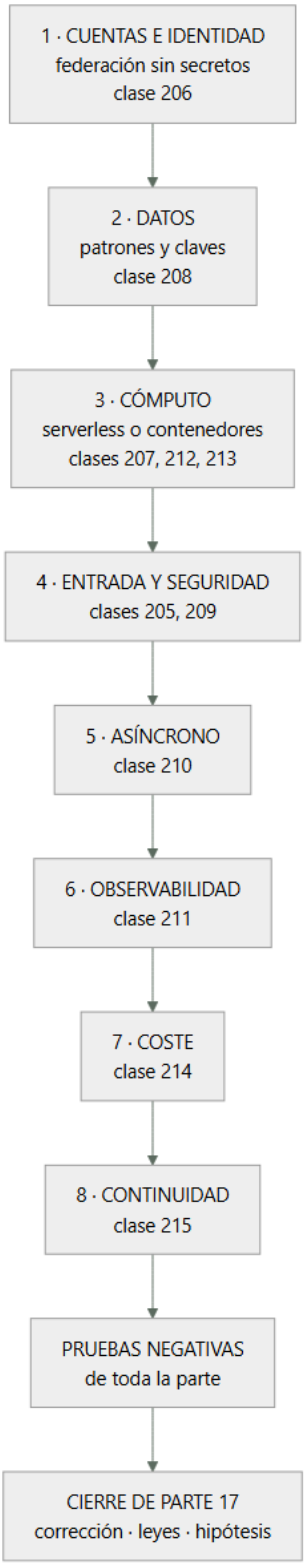
Concepto	Comprensión verificable
sistema productivo	El que tiene usuarios reales, guardia, presupuesto y continuidad comprobada. No el que funciona.
orden de construcción	Cuentas e identidad primero, diagnóstico y coste al final. Lo que condiciona a lo demás, antes.
valor por defecto	Configuración inicial de un servicio. Está elegida para que la demostración funcione.
ley 26	El valor por defecto está elegido para que la demostración funcione, no para que el sistema aguante.
prueba negativa de parte	Comprobación acumulada de las once clases, ejecutada sobre el sistema entero.
hipótesis de parte	Afirmación refutable escrita antes de estudiar, que la parte siguiente corrige con evidencia.

Modelo mental

AWS se aprende como una progresión operativa: identidad federada, infraestructura declarativa, entrega, señales, recuperación y costo controlado.

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

■ Flujo de razonamiento



Desarrollo

1. El encargo y su orden

El encargo. Llevar a producción la plataforma de pedidos de CloudShop en AWS: con usuarios reales, guardia, presupuesto y continuidad comprobada.

El orden, por coste de cambio:

```
1  CUENTAS E IDENTIDAD                              clase 206
   una cuenta por entorno; federación sin secretos
   barreras de organización: nadie puede crear roles ni
   desactivar auditoría desde una canalización
   → si esto llega tarde, hay que migrar todo lo anterior

2  DATOS                                            clase 208
   patrones de acceso escritos ANTES
   claves de partición que reparten
   lo que no encaja, replicado a otro almacén
   → la decisión más cara de cambiar de la parte

3  CÓMPUTO                                 clases 207, 212, 213
   funciones para lo que encaja; contenedores para lo demás
   concurrencia reservada calculada por las conexiones
   memoria y tamaños medidos, no copiados

4  ENTRADA Y SEGURIDAD                      clases 205, 209
   distribución, bucket privado, capa 7 detrás
   testigo de acceso validado con las siete comprobaciones
   filtrado en modo cuenta antes de bloquear

5  ASÍNCRONO                                        clase 210
   bus, cola por consumidor, idempotencia
   cola de fallidos con alerta por antigüedad y reproceso
   probado

6  OBSERVABILIDAD                                   clase 211
   declarada junto al servicio; alertas que llegan a alguien

7  COSTE                                            clase 214
   etiquetas obligatorias en la creación; presupuestos con
   acción; retirada de lo ocioso

8  CONTINUIDAD                                      clase 215
   segunda región, y el ejercicio dirigido que la comprueba
```

Y la regla que resume la parte:

```
en cada uno de los ocho pasos, la primera tarea es
REVISAR LOS VALORES POR DEFECTO
→ porque están elegidos para que la demostración funcione
                                                      ley 26
```

Y el error de método que hunde este proyecto:

```
empezar por el paso 3, que es lo divertido
→ y descubrir en el paso 2 que la clave de partición está
  mal y hay que migrar                          clase 208
```

2. El entregable, las pruebas y la evaluación

El documento, con lo que hay que poder enseñar:

```
1  el problema, con la cifra que lo demuestra
2  patrones de acceso, escritos y fechados antes del modelo
3  arquitectura: cuentas, servicios, datos, entrada, salida
4  la lista de VALORES POR DEFECTO cambiados y por qué
5  concurrencia, plazos y capacidad, con su cálculo
6  seguridad: testigos, autorización, filtrado, perímetro
7  observabilidad: qué se vigila y qué solo se consulta
8  coste: desglose, atribución y coste por unidad de negocio
9  continuidad: modo, plazos medidos y ejercicio ejecutado
10 pruebas negativas, con los fallos publicados
11 lo que NO se hace, y por qué
```

Las pruebas negativas de la parte, que son el criterio de terminado:

- leer un objeto por la URL directa del bucket
- asumir el rol de producción desde otra rama o repositorio
- llamar a la API con testigo de identidad, caducado o alterado
- pedir el recurso de otro usuario con testigo válido
- llamar a la pasarela saltándose la distribución
- registrar un nombre con apóstrofo (que NO se bloquee)

- superar el límite de ritmo y recibir 429
- enviar la misma petición 50 veces en paralelo
- enviar el mismo mensaje 50 veces a la cola
- dejar un mensaje en la cola de fallidos y esperar la alerta
- reprocesar 10.000 mensajes sin tumbar la base
- agotar la concurrencia y comprobar que se rechaza en vez de caer
- desplegar una versión rota y ver la reversión automática
- desplegar durante tráfico y contar peticiones cortadas
- provocar la condición de cada alerta y ver si llega
- crear un recurso sin etiquetas
- borrar la pila y comprobar que los datos sobreviven
- perder la región primaria y cronometrar los cinco tramos
- volver a la primaria sin dos escritores

Y los criterios de evaluación, publicados antes:

```
1  los patrones de acceso están fechados antes del modelo  3
2  la lista de valores por defecto cambiados es explícita  3
3  no queda ninguna credencial de larga duración           3
4  la concurrencia está calculada, no puesta a ojo         2
5  la autorización comprueba la propiedad del recurso      3
6  toda operación con efecto es idempotente                3
7  cada alerta se ha probado provocando la condición       2
8  el coste atribuido supera el 90 %                       2
9  hay coste por unidad de negocio                         2
10 la continuidad se midió ejecutándola                    3
11 las pruebas negativas se ejecutaron y hay fallos
   publicados                                              3
12 está escrito lo que no se hace                          1
```

Y el 11 pesa como los que más, con la evidencia acumulada:

```
proporción de pruebas negativas que han fallado la primera
vez en este programa
  clase 144    3 de 11      27 %
  clase 168    5 de 11      45 %
  clase 179    9 de 31      29 %
  clase 189    3 de 14      21 %
  clase 204    6 de 15      40 %
→ un proyecto con cero fallos no las ejecutó       ley 22
```

3. Cierre de la parte 17: corrección de las cinco predicciones

Las cinco predicciones de la clase 204, corregidas con la evidencia de las clases 205 a 215.

1. «bajar a un proveedor concreto revelará que buena parte de lo que hemos tratado como decisiones de arquitectura son valores por defecto; y estarán mal elegidos para producción en más de la mitad de los casos»

   CORRECTA, y con margen. De los valores por defecto que hubo que revisar en las once clases, la cuenta salió así: bucket con alojamiento estático y política pública, error 404 convertido en 200, política de caché heredada, invalidación con comodín, memoria de función en el mínimo, plazo en 30 s, sin concurrencia reservada, sin alias, visibilidad de cola en 30 s, sin informe de fallo parcial, grupos de registros sin caducidad, nivel de depuración en producción, etiquetas de imagen mutables, registro sin caducidad, drenaje corto, porcentaje mínimo sano al 50 %, periodo de gracia menor que el arranque, circuito de despliegue desactivado y escalado por CPU. Diecinueve, y los diecinueve había que cambiarlos.

2. «el diseño de la base de datos por patrones de acceso será la decisión más cara de cambiar de la parte, y la que más veces se toma sin haber escrito los patrones»

   CORRECTA en las dos mitades. Fue la más cara: seis semanas de dos personas para cambiar una clave de partición que se había elegido en una tarde, por la primera consulta que pidió negocio. Y se tomó sin escribir los patrones, que habrían costado tres horas. Pero incompleta en algo: el

error más CARO fue ese, y el más FRECUENTE fue otro —los valores por defecto, que aparecieron en casi todas las clases—.

3. «la eliminación de secretos mediante federación resultará sencilla técnicamente y lenta organizativamente: el obstáculo será quién tiene permiso para cambiarla»

   FALLADA, y en las dos mitades. La parte técnica NO era sencilla: tenía una trampa que dejó el sistema PEOR que con la clave estática, porque una condición de sujeto con comodín permitía a cualquiera de doscientos catorce miembros desplegar en producción creando un repositorio. Y la lentitud no vino de los permisos: vino de que el inventario decía cuarenta y una canalizaciones y había cuarenta y nueve. Predijimos el obstáculo equivocado, y lo más grave es que dimos por resuelta la parte que tenía el fallo.

4. «más de la mitad de los problemas del proyecto productivo no serán de AWS sino de lo mismo de siempre: leyes 25, 15 y 22»

   CORRECTA. Una política pública añadida «para probar» y no retirada; ocho canalizaciones fuera del inventario, una con credenciales de administrador para un sistema retirado en 2023; once alertas a canales sin suscriptores, entre ellas la de la cola de fallidos que ocho meses después dejó cuarenta y un mil mensajes a un día de caducar; dos bases de datos de pruebas encendidas catorce meses; treinta y un entornos efímeros que nunca caducaron; y en el ejercicio de continuidad, seis de ocho hallazgos que no eran de infraestructura.

5. «el coste real será entre dos y cuatro veces la estimación inicial, y la diferencia estará casi toda en partidas que no son cómputo»

   CORRECTA EN EL MECANISMO Y CORTA EN LA CIFRA. La segunda mitad se cumplió por completo: el cómputo resultó ser el 17 % de la factura de la API y el 20 % de la de la plataforma, y las partidas sorpresa fueron pasarela, registros, índices, transferencia entre zonas y capas huérfanas del registro de imágenes. Pero el factor no fue de dos a cuatro: fue de SIETE, porque la estimación inicial solo contaba invocaciones y duración.

Marcador: tres correctas, una correcta y subestimada, una fallada. Y la fallada enseña más que las otras cuatro: dimos por resuelta la parte técnica y pusimos el problema en las personas, y resultó que la parte técnica tenía una trampa que empeoraba la seguridad mientras parecía mejorarla.

4. Recuento de leyes, ley 26 e hipótesis de la parte 18

El recuento de leyes, cerrada la parte 17.

```
ley 13  lo que no se mira deja de funcionar en silencio       45
ley 15  la señal existe y nadie la mira                       34
ley 22  un procedimiento nunca ejecutado no funciona          29
ley 14  el coste se decide al crear, no al pagar              28
ley 16  un control que estorba se rodea                       26
ley 20  lo que no tiene dueño se filtra y se desperdicia      26
ley 21  el acoplamiento vive en quién escribe                 21
ley 23  la capacidad la limita lo que ya se mantiene          13
ley 25  lo provisional sobrevive a su motivo                  12
ley 24  lo que no está en el diagrama no se analiza           11
ley 19  la compensación hace invisible el fallo               10
ley 17  se optimiza la medida, no el objetivo                 10
ley 18  lo asíncrono traslada la garantía, no la elimina       8
```

Y la parte 17 obliga a escribir una ley nueva, que es la más específica de todas y la que más dinero y latencia ha movido en esta parte:

```
LEY 26
```

```
  el valor por defecto está elegido para que la
  demostración funcione, no para que el sistema aguante

apariciones en esta parte                                  5
  clase 205   alojamiento estático del bucket, 404 a 200,
              política de caché heredada, invalidación con
              comodín
  clase 207   memoria mínima, plazo de 30 s, sin
              concurrencia reservada, sin alias
  clase 210   visibilidad de 30 s, sin informe de fallo
              parcial de lote
  clase 211   grupos de registros sin caducidad, nivel de
              depuración en producción
  clase 212   drenaje corto, mínimo sano al 50 %, gracia
              menor que el arranque, escalado por CPU

y lo que la distingue
  no habla de descuido ni de falta de dueño
  habla de que el valor inicial está OPTIMIZADO PARA OTRA
  COSA: para que funcione en cinco minutos en un tutorial
  → el remedio no es vigilar: es revisar la lista de
    valores por defecto como primera tarea de cada servicio
```

La hipótesis de la parte 18 (clases 217 a 228, Azure en producción), escrita antes de estudiarla para que la clase 228 la corrija:

```
1. los valores por defecto de Azure estarán mal elegidos para
   producción en una proporción parecida a los de AWS, pero
   fallarán en otro sitio: los de AWS tienden a ser
   permisivos en red y almacenamiento; los de Azure lo serán
   en ámbito de identidad y de asignación de permisos
                                                  ley 26

2. la jerarquía de grupos de administración y suscripciones
   resultará ser el equivalente del plan de direcciones: se
   decide en una tarde, condiciona una década y renumerarla
   costará meses                                   ley 14

3. la identidad será el eje de toda la parte: lo que en AWS
   se resuelve con roles y políticas se resolverá aquí con
   el directorio, y el error más frecuente será un ÁMBITO DE
   ASIGNACIÓN demasiado amplio —el equivalente exacto de la
   condición de sujeto con comodín de la clase 206

4. la mayoría de los conceptos técnicos se corresponderán uno
   a uno entre las dos nubes; lo que NO se corresponderá es
   el modelo operativo, y trasladar «la misma arquitectura»
   costará más en operación que en código        clase 158

5. los problemas del proyecto productivo volverán a ser, en
   mayoría, de las leyes 25, 15 y 22. Y aquí añadimos algo
   incómodo: si esta predicción vuelve a acertar por cuarta
   parte consecutiva, dejará de tener mérito y pasará a ser
   una descripción, no una hipótesis. Lo que habrá que
   preguntarse entonces no es si acertamos, sino por qué,
   sabiéndolo, sigue ocurriendo
```

Y el cierre de la parte 17: de once clases, lo que más dinero y latencia movió no fue ninguna decisión de arquitectura, sino diecinueve valores por defecto que había que cambiar y que funcionaban perfectamente sin cambiarlos. La parte 18 hace el mismo recorrido en Azure, empezando por lo que allí condiciona todo lo demás: la jerarquía de suscripciones y grupos de administración. Es la clase 217.

Ejemplo trabajado

El sistema de CloudShop en producción en AWS, montado con el método de la parte. Lo que sigue es el resumen del entregable, la lista de valores por defecto cambiados, y el resultado de las diecinueve pruebas negativas —de las que fallaron siete.

La arquitectura, en una página:

```
CUENTAS       organización con 4 cuentas: producción,
              preproducción, desarrollo, seguridad
```

```
              barreras: sin crear roles, sin desactivar
              auditoría, sin borrar copias

IDENTIDAD     federación desde el repositorio, 4 roles por
              propósito, producción atada a entorno
              protegido con 2 aprobaciones
              credenciales de larga duración: 2, ambas de
              terceros, con rotación y fecha de revisión

DATOS         DynamoDB con tabla única para pedidos y
              clientes; 9 patrones escritos y fechados
              búsqueda y analítica alimentadas por el flujo
              de cambios
              PostgreSQL solo para facturación heredada

CÓMPUTO       funciones para la API de pedidos
              contenedores para búsqueda y para el
              procesador de eventos
              Kubernetes solo para las cargas del equipo de
              datos

ENTRADA       CloudFront → capa 7 → API HTTP → funciones
              bucket privado con control de acceso al origen
              filtrado con 4 grupos, ajustados tras 3
              semanas en modo cuenta

ASÍNCRONO     bus → 4 colas → 4 consumidores
              todos idempotentes; todas con cola de fallidos

OBSERVABILIDAD declarada en la plantilla de cada servicio
              584 series de métricas, 63 alertas, 1 canal

COSTE         5 etiquetas obligatorias, barrera de creación
              presupuestos con acción en no producción
              retirada automática de lo ocioso

CONTINUIDAD   activo-pasivo en caliente en eu-central-1
              ejercicio ejecutado 2 veces
```

La lista de valores por defecto cambiados, que fue el entregable más útil:

servicio	por defecto	cambiado a
S3	alojamiento estático	desactivado
S3	acceso público	bloqueado
S3	sin versionado	activado
CloudFront	caché heredada	política propia por ruta
CloudFront	sin cabeceras seg.	política de cabeceras
CloudFront	404 → 200 con index	función de borde por extensión
Lambda	128 MB	1.024 MB (medido)
Lambda	plazo 30 s	8 s
Lambda	sin concurrencia reservada	reservada, por conexiones
Lambda	sin alias	alias + escalonado
API	REST	HTTP
SQS	visibilidad 30 s	300 s (por lote)
SQS	sin fallo parcial	activado
SQS	retención 4 días	14 días
CloudWatch	sin caducidad	21 días
CloudWatch	nivel depuración	información
ECR	etiquetas mutables	inmutables
ECR	sin caducidad	10 últimas por rama
ECS	drenaje 30 s	60 s
ECS	mínimo sano 50 %	100 %
ECS	gracia 30 s	120 s
ECS	sin circuito	activado
ECS	escalado por CPU	peticiones por tarea
DynamoDB	lectura fuerte	eventual salvo 2 patrones

```
total cambiados                                          24
que funcionaban sin cambiarlos                           24
```

Y la observación que el equipo escribió:

```
los veinticuatro funcionaban
ninguno daba error
ninguno aparecía en ninguna alerta
y los veinticuatro habrían causado un problema en
producción, entre coste, latencia, pérdida de datos o
seguridad                                                ley 26
```

Las diecinueve pruebas negativas: siete fallaron.

```
✓  URL directa del bucket                          denegada
✓  rol de producción desde otra rama               denegado
✓  testigo de identidad, caducado o alterado       rechazado
✗  recurso de otro usuario con testigo válido
   → 2 de 11 rutas no comprobaban la propiedad; las 2 eran
     rutas nuevas añadidas después de la revisión
                                                   clase 209
✓  llamada a la pasarela saltándose la distribución denegada
✓  registro con apóstrofo                          aceptado
✓  límite de ritmo                                 429
✓  misma petición 50 veces en paralelo             1 efecto
✗  mismo mensaje 50 veces a la cola
   → el consumidor de notificaciones no era idempotente:
     se había añadido en el último mes sin la comprobación
✓  mensaje en cola de fallidos → alerta             41 s
✗  reprocesar 10.000 mensajes
   → tumbó la base la primera vez: el procedimiento tenía
     límite de ritmo y quien lo ejecutó no lo usó porque
     no estaba en el primer paso                   clase 210
✓  agotar la concurrencia                          429, sin
                                                   caída
✓  versión rota                                    revertida
                                                   en 2 min
✗  desplegar durante tráfico
   → 14 peticiones cortadas en el servicio de búsqueda:
     su aplicación no atendía la señal de terminación
                                                   clase 212
✗  provocar la condición de cada alerta
   → 5 de 63 no llegaron; 3 por umbral mal puesto y 2 por
     destino equivocado
✓  recurso sin etiquetas                           rechazado
✗  borrar la pila y comprobar los datos
   → la tabla de idempotencia NO tenía política de
     retención y se borró
   → el sistema siguió funcionando, pero perdió la memoria
     de lo procesado: 1.100 mensajes en vuelo se habrían
     duplicado al reintentarse
✗  perder la región primaria
   → 12 min 40 s frente a los 10 min 40 s del último
     ejercicio: había 2 servicios nuevos sin réplica de
     imagen en la secundaria
✓  volver sin dos escritores                       correcto
```

Y el análisis de las siete:

```
dos por servicios AÑADIDOS después de la última revisión
  (rutas sin comprobación de propiedad, consumidor sin
   idempotencia)
dos por procedimientos con pasos en mal orden o umbrales
  mal puestos
dos por omisiones al crecer (retención de una tabla nueva,
  réplica de imágenes de servicios nuevos)
una por una aplicación que no atendía la señal

→ SEIS de las siete se deben a lo mismo: el sistema creció
  y las comprobaciones no crecieron con él
→ y por eso las pruebas negativas se ejecutan
  PERIÓDICAMENTE, no una vez                       ley 22
```

Las cifras del sistema en producción, tras tres meses:

```
p99 del flujo de compra                  < 500 ms    412 ms
disponibilidad observada                   99,8 %    99,86 %
techo por dependencias                     99,84 %   —
coste mensual                            12.000 €   16.700 €
coste por pedido                           0,050 €    0,046 €
tiempo de conmutación de región            15 min    12 min 40 s
pérdida de datos en conmutación              1 min     22 s
coste atribuido                               90 %    94,1 %
alertas por turno                            < 2       0,7
proporción de alertas accionables           > 80 %      89 %
despliegues al mes                             —        94
  revertidos automáticamente                   —         5
  que causaron incidente                       —         0
```

Y las tres cosas que se decidió no hacer, registradas:

```
no adoptar activo-activo: 7.700 €/mes frente a 410 € de
  pérdida evitada; se revisa si el negocio de empresa
  supera el 15 % de los ingresos
no migrar los servicios web a Kubernetes: funcionan
no unificar todo en contenedores: las funciones cubren su
  caso con menos operación
```

La lección que este proyecto deja: el entregable que más valor tuvo no fue el diagrama ni el cálculo de capacidad: fue la tabla de veinticuatro valores por defecto cambiados, todos los cuales funcionaban sin cambiarlos. Y de las diecinueve pruebas negativas, seis de las siete que fallaron se debieron a que el sistema creció y las comprobaciones no crecieron con él: rutas nuevas sin comprobar propiedad, un consumidor nuevo sin idempotencia, una tabla nueva sin retención y dos servicios nuevos sin réplica de imagen.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-17-aws-production-architecture/216-proyecto-cloudshop-productivo-en-aws/lab.py
```

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

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

Evidencia esperada

El artefacto principal es un incremento integrado, demostrable y documentado. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

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

■ Criterio de aceptación

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

- [ ] Otra persona puede repetir el recorrido sin conocimiento tácito.

■■ Errores frecuentes

Síntoma	Causa probable	Corrección
Hay que migrar datos a mitad del proyecto	Se empezó por el cómputo y la clave de partición se eligió por la primera consulta que pidió negocio	Escribe los patrones de acceso con su frecuencia y latencia antes de modelar, y fecha ese documento.
Todo funciona en pruebas y falla el primer día de carga real	Los valores por defecto están elegidos para que la demostración funcione	Revisa la lista de valores por defecto como primera tarea de cada servicio y documenta cuáles cambiaste y por qué.
Las comprobaciones pasan pero el sistema tiene huecos nuevos	Se añadieron servicios y rutas después de la última revisión	Ejecuta las pruebas negativas periódicamente y añade una por cada capacidad nueva, no solo al terminar.
Borrar una pila se lleva datos por delante	Los recursos con datos no tienen política de retención y viven en la misma pila que el código	Separa las pilas por ciclo de vida y pon retención a todo lo que guarde estado, incluidas las tablas auxiliares.
El procedimiento de recuperación se ejecuta mal bajo presión	Los pasos críticos no están al principio ni son obligatorios	Pon el límite de ritmo y los pasos de seguridad como primer paso, y haz que lo ejecute alguien que no lo escribió.
La estimación de coste se queda muy corta	Solo se contó el cómputo	Estima pasarela, registros, índices, transferencia entre zonas y almacenamiento de imágenes; suelen sumar más que el cómputo.

■ 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 primera tarea de cada servicio es revisar sus valores por defecto?
2. ¿Cuál de las cinco predicciones de la clase 204 falló y qué enseña su fallo?
3. ¿Qué dice la ley 26 y en qué se distingue de la 25?
4. ¿Qué proporción de pruebas negativas ha fallado la primera vez en este programa?
5. ¿Por qué seis de las siete pruebas fallidas del proyecto tenían la misma causa de fondo?

Referencias

- AWS (2025). Well-Architected Framework. <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html>
- AWS (2025). Serverless Application Lens. <https://docs.aws.amazon.com/wellarchitected/latest/serverless-applications-lens/welcome.html>
- AWS (2025). Security Reference Architecture. <https://docs.aws.amazon.com/prescriptive-guidance/latest/security-reference-architecture/welcome.html>
- Beyer, B. y otros (2018). The Site Reliability Workbook. <https://sre.google/workbook/table-of-contents/>
- Basiri, A. y otros (2016). Chaos engineering. <https://ieeexplore.ieee.org/document/7503833>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Evaluación — 216 Proyecto: CloudShop productivo en AWS

Preguntas

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

Reto verificable

Entrega cloudshop-aws 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.