Multi-Cloud Engineering Program

Parte 02 — AWS: plataforma esencial

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	02 de 23
Clases	025–036
Evaluaciones	12
Horas estimadas	52

Índice

Alcance de este cuaderno 2

Índice 3

Parte 02 — AWS: plataforma esencial 4

025 — Organizations, cuentas, OU, SCP y landing zone . . . 6
026 — IAM, roles, políticas, STS y federación . . . 14
027 — VPC, subredes, rutas, NAT, endpoints y seguridad . . . 22
028 — EC2, AMI, EBS y selección de capacidad . . . 30
029 — Elastic Load Balancing y Auto Scaling . . . 38
030 — S3: objetos, versionado, lifecycle y replicación . . . 46
031 — RDS, DynamoDB y ElastiCache: decisión de datos . . . 54
032 — Lambda, API Gateway y Step Functions . . . 62
033 — SQS, SNS y EventBridge . . . 70
034 — CloudWatch, CloudTrail, Config y Systems Manager . . . 78
035 — KMS, Secrets Manager, WAF y controles de seguridad . . . 86
036 — Proyecto: aplicación de tres capas en AWS . . . 95

Parte 02 — AWS: plataforma esencial

← Parte 01 · Índice completo · Parte 03 →

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

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

Diseñar y operar una carga completa en AWS con controles explícitos.

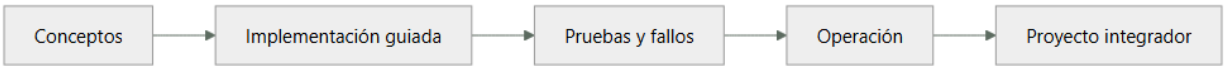
Resultados de la parte

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

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

Secuencia



Clases

ID	Clase	Laboratorio	Horas
025	Organizations, cuentas, OU, SCP y landing zone	governance	4
026	IAM, roles, políticas, STS y federación	iam	4
027	VPC, subredes, rutas, NAT, endpoints y seguridad	network	4
028	EC2, AMI, EBS y selección de capacidad	compute	4
029	Elastic Load Balancing y Auto Scaling	reliability	4
030	S3: objetos, versionado, lifecycle y replicación	storage	4
031	RDS, DynamoDB y ElastiCache: decisión de datos	data	4
032	Lambda, API Gateway y Step Functions	serverless	4
033	SQS, SNS y EventBridge	messaging	4
034	CloudWatch, CloudTrail, Config y Systems Manager	observability	4
035	KMS, Secrets Manager, WAF y controles de seguridad	security	4
036	Proyecto: aplicación de tres capas 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 — John Culkin y Mike Zazon.
- Amazon Web Services in Action — Wittig y Wittig.

025 — Organizations, cuentas, OU, SCP y landing zone

Parte: 02 — AWS: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: governance · Estado: EXECUTABLECORE

Propósito

Construir la estructura de cuentas que sostendrá todo lo que se despliegue en AWS durante el resto del programa. Es la decisión más irreversible de la parte: mover cuentas entre unidades organizativas cambia las políticas que se les aplican, y separar más tarde una cuenta compartida exige migrar recursos. Aquí se aplica a AWS lo que la clase 017 estableció de forma neutral.

Resultados de aprendizaje

Al finalizar podrás:

1. Diseñar una jerarquía de unidades organizativas por régimen de gobierno y no por organigrama.
2. Escribir una SCP que restrinja sin inutilizar la cuenta, exceptuando correctamente los servicios globales.
3. Explicar por qué una SCP no concede permisos y qué implica al depurar un acceso denegado.
4. Desplegar una landing zone con cuentas de auditoría, red y seguridad separadas desde el inicio.
5. Verificar que los guardarraíles funcionan mediante prueba negativa, no por inspección de la configuración.

Conceptos centrales

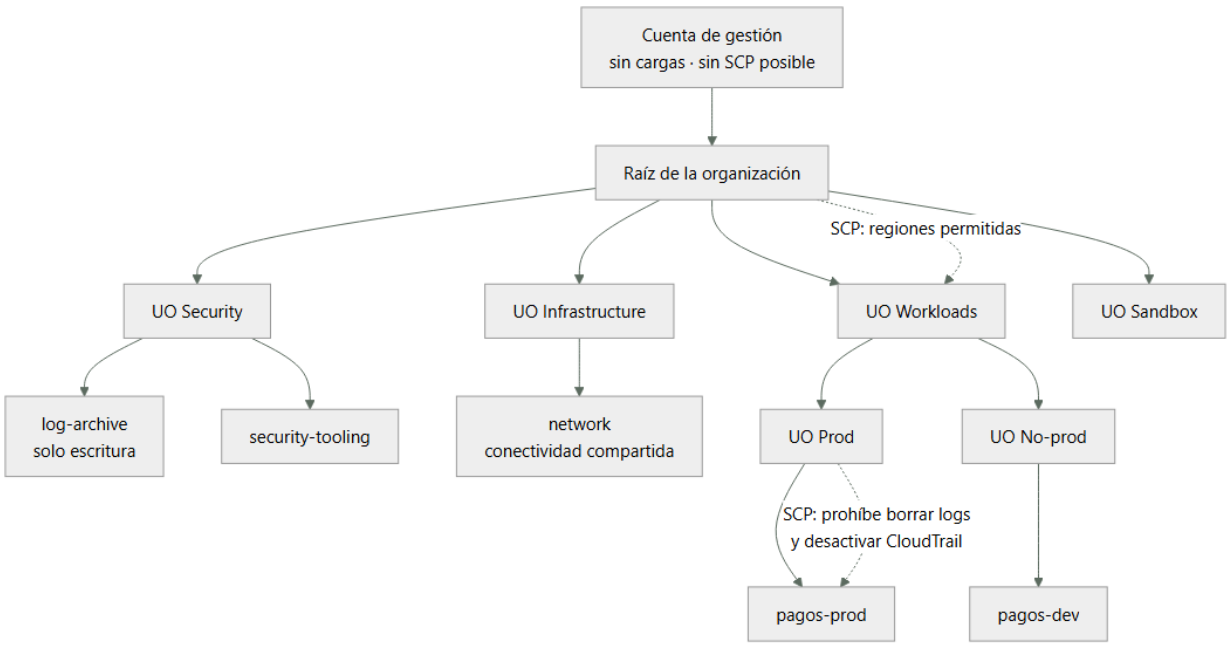
Concepto	Comprensión verificable
SCP	Service Control Policy: política heredada que define el techo máximo de permisos de las cuentas bajo una unidad organizativa. Nunca concede: solo limita lo que otras políticas pueden otorgar.
landing zone	Estructura base de cuentas, red, identidad y registro sobre la que se despliega todo lo demás. Su valor está en existir antes de la primera carga, porque retrofitarla es caro.
cuenta de gestión	La que crea la organización. No admite SCP sobre sí misma, así que no debe alojar cargas: cualquier compromiso ahí no tiene barrera superior que lo contenga.
guardarraíl preventivo	Control que impide la acción antes de que ocurra, normalmente una SCP. Se distingue del detectivo, que avisa después, y exige prueba negativa para demostrar que efectivamente deniega.
cuenta de auditoría	Destino centralizado de registros de todas las demás, con escritura permitida y borrado prohibido. Es lo que evita que quien comprometa una cuenta pueda borrar su propio rastro.

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

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

■ Flujo de razonamiento



Desarrollo

1. La cuenta de gestión no aloja cargas

AWS Organizations tiene una asimetría deliberada: la cuenta de gestión no puede tener SCP aplicadas sobre sí misma. Es la única de toda la organización sin barrera superior.

La consecuencia es directa: cualquier recurso que viva ahí opera sin los guardarraíles que protegen a las demás, y cualquier credencial comprometida en esa cuenta tiene el poder de modificar la organización entera —crear cuentas, mover unidades, desactivar políticas—.

Lo que sí pertenece a la cuenta de gestión:

```
- la propia organización y sus SCP
- la agregación de facturación
- nada más
```

Y lo que hay que hacer con ella el primer día:

```
# Activar MFA en el usuario raíz y no volver a usarlo
$ aws iam get-account-summary --query 'SummaryMap.AccountMFAEnabled'
1
# Cero claves de acceso del usuario raíz
$ aws iam get-account-summary --query 'SummaryMap.AccountAccessKeysPresent'
0
# Alerta ante cualquier uso del usuario raíz
```

La tercera línea es el control más rentable de toda la landing zone: el uso del usuario raíz es un evento que debería ocurrir una o dos veces al año —para tareas que solo él puede hacer— y cualquier otra aparición merece una llamada telefónica.

AWS Control Tower automatiza buena parte de esta estructura, y conviene saber qué hace por debajo antes de usarlo: crea las unidades organizativas Security y Sandbox, las cuentas de archivo de registros y de auditoría, y aplica un conjunto de guardarraíles. Usarlo sin entender la estructura produce dependencia de la herramienta para operaciones que después hay que hacer a mano.

2. SCP: el orden de evaluación decide el diagnóstico

Una SCP no concede nada. Es un filtro sobre lo que las políticas de identidad pueden otorgar dentro de esa rama de la organización:

```
permiso efectivo = SCP ∩ política de identidad ∩ política de recurso ∩ límite de permisos
```

Dos consecuencias operativas:

1. Adjuntar una SCP con Allow: no da acceso a nadie. Si ninguna política de identidad lo concede, no hay permiso.
2. Un Deny en la SCP gana sobre cualquier concesión, incluida la del administrador de la cuenta. No hay escalada posible desde dentro.

Las SCP admiten dos estrategias, y mezclarlas causa confusión:

```
lista de permitidos (allowlist):  quitar FullAWSAccess y enumerar lo permitido
  → muy restrictivo, alto mantenimiento: cada servicio nuevo exige tocar la SCP

lista de denegados (denylist):    mantener FullAWSAccess y denegar lo prohibido
  → lo habitual: menos mantenimiento, protege contra lo que se sabe peligroso
```

El error de diagnóstico más caro es buscar el problema en la política de identidad cuando lo deniega una SCP. El mensaje de error de AWS lo distingue, y conviene leerlo con atención:

```
AccessDenied ... with an explicit deny in a service control policy
AccessDenied ... with an explicit deny in an identity-based policy
AccessDenied ... no identity-based policy allows the action
```

La tercera es denegación implícita —falta conceder—; las dos primeras dicen exactamente qué documento hay que revisar. Añadir permisos ante la primera no cambia nada.

3. Restringir regiones sin romper la cuenta

La SCP más común y la que más cuentas ha inutilizado. Restringir por región parece trivial hasta que se descubre que varios servicios de AWS son globales y sus llamadas se emiten contra us-east-1.

La versión que rompe:

```
{
  "Effect": "Deny",
  "Action": "*",
  "Resource": "*",
  "Condition": {"StringNotEquals": {"aws:RequestedRegion": ["us-east-1", "sa-east-1"]}}
}
```

En cuanto se aplica, deja de funcionar IAM, la facturación, el soporte y Route 53 desde cualquier contexto que no resuelva a esas regiones. Y con Action: "" se bloquea incluso la posibilidad de quitar la política desde la propia cuenta.

La versión correcta excluye los servicios sin región:

```
{
  "Effect": "Deny",
  "NotAction": [
    "iam:*", "organizations:*", "support:*", "sts:*",
    "route53:*", "cloudfront:*", "budgets:*", "ce:*",
    "health:*", "waf:*", "shield:*"
  ],
  "Resource": "*",
  "Condition": {
    "StringNotEquals": {"aws:RequestedRegion": ["us-east-1", "sa-east-1"]}
  }
}
```

Y un matiz sobre sts:: excluirlo es necesario porque el punto de acceso global de STS resuelve a us-east-1; sin la excepción, asumir roles falla desde otras regiones.

Toda SCP debe probarse en una cuenta de sandbox antes de aplicarla a una unidad organizativa productiva. No hay simulador que capture todos los casos, y el coste de equivocarse es una cuenta que no se puede administrar.

4. Cuentas de la base: auditoría, red y seguridad

Tres cuentas que conviene separar desde el primer día, porque retrofitarlas exige mover recursos con estado:

Archivo de registros (log-archive). Recibe CloudTrail, Config y logs de acceso de todas las cuentas. Su política de bucket permite escritura desde la organización y prohíbe el borrado a todo el mundo, incluido su propio administrador:

```
{
  "Effect": "Deny",
  "Principal": "*",
  "Action": ["s3:DeleteObject", "s3:DeleteObjectVersion", "s3:PutBucketPolicy"],
  "Resource": "arn:aws:s3:::org-log-archive/*"
```

```
}
```

Con bloqueo de objetos en modo cumplimiento, ni siquiera la cuenta raíz puede borrarlos antes de que expire la retención. Ese es el punto: un atacante que compromete una cuenta no puede borrar la evidencia de lo que hizo.

Red (network). Aloja el Transit Gateway, las zonas privadas de DNS y los endpoints compartidos. Separarla evita que el ciclo de vida de la conectividad dependa de un producto concreto, y permite que el equipo de red tenga permisos ahí sin tenerlos en las cuentas de carga.

Herramientas de seguridad (security-tooling). Cuenta delegada como administradora de GuardDuty, Security Hub y Config. La delegación es importante: permite operar esos servicios en toda la organización sin usar la cuenta de gestión, que es donde no hay barreras.

Las tres tienen algo en común: su valor aparece durante un incidente, y para entonces ya no se pueden crear.

5. Un guardarraíl sin prueba negativa es una intención

Configurar una SCP no demuestra que deniegue. La configuración puede estar adjuntada a la unidad equivocada, tener una condición que nunca se cumple, o quedar anulada por un NotAction demasiado amplio.

La verificación exige intentar la acción y comprobar que falla:

```
# Prueba negativa 1: región no permitida
$ aws ec2 describe-instances --region eu-west-1
An error occurred (UnauthorizedOperation) ... with an explicit deny in a
service control policy                                        ✓ deniega

# Prueba negativa 2: desactivar el rastro de auditoría
$ aws cloudtrail stop-logging --name org-trail
An error occurred (AccessDeniedException) ... explicit deny in a service
control policy                                                ✓ deniega

# Prueba positiva: lo que debe seguir funcionando
$ aws iam list-roles --max-items 1 >/dev/null && echo "IAM operativo  ✓"
$ aws sts get-caller-identity >/dev/null && echo "STS operativo  ✓"
```

Las dos últimas líneas son tan importantes como las dos primeras: una SCP que deniega lo que debe y también lo que no, es un fallo igual de grave, solo que se descubre más tarde y en peor momento.

El simulador de políticas de IAM ayuda pero no basta: no evalúa SCP en todos los escenarios ni captura las llamadas que los servicios hacen entre sí. La única evidencia válida es el intento real desde una cuenta bajo la unidad organizativa correspondiente.

Estas pruebas deben ser automáticas y periódicas, no de una sola vez. Una SCP correcta hoy puede quedar anulada mañana por otra adjuntada a un nivel distinto de la jerarquía, sin que nadie modifique la primera.

Ejemplo trabajado

CloudShop opera en una sola cuenta de AWS y migra a una landing zone. Se ejecuta y se verifica paso a paso.

Estado inicial y sus tres riesgos concretos:

```
$ aws organizations describe-organization 2>&1 | head -1
AWSOrganizationsNotInUseException
$ aws cloudtrail describe-trails --query 'trailList[0].[Name,S3BucketName,IsMultiRegionTrail]' --
output text
cloudshop-trail   cloudshop-logs   False
```

```
1. Sin organización: no hay SCP posible, ningún guardarraíl preventivo.
2. CloudTrail en la MISMA cuenta: quien la comprometa borra su rastro.
3. Rastro de una sola región: la actividad en otras regiones es invisible.
```

Paso 1 — organización y unidades por régimen de gobierno, no por equipo:

```
$ aws organizations create-organization --feature-set ALL
$ for uo in Security Infrastructure Workloads Sandbox; do
    aws organizations create-organizational-unit --parent-id r-abcd --name $uo
  done
$ aws organizations create-organizational-unit --parent-id ou-work --name Prod
$ aws organizations create-organizational-unit --parent-id ou-work --name NonProd
```

Paso 2 — cuentas de la base:

```
log-archive        Security         destino de logs, borrado prohibido
security-tooling   Security         administrador delegado de GuardDuty y Config
network            Infrastructure   Transit Gateway y DNS privado
cloudshop-prod     Workloads/Prod   la carga actual, migrada aquí
cloudshop-dev      Workloads/NonProd
```

Paso 3 — SCP de regiones, probada antes en Sandbox:

```
$ aws organizations attach-policy --policy-id p-regiones --target-id ou-sandbox
# desde una cuenta de sandbox:
$ aws ec2 describe-instances --region ap-south-1
AccessDenied ... explicit deny in a service control policy      ✓
$ aws iam list-roles --max-items 1 >/dev/null && echo ok
ok                                                              ✓ IAM intacto
$ aws sts get-caller-identity >/dev/null && echo ok
ok                                                              ✓ STS intacto
```

La primera versión de la SCP sí rompió STS: sin sts: en el NotAction, asumir roles desde sa-east-1 fallaba porque el punto de acceso global resuelve a us-east-1. Se detectó en sandbox, que es exactamente para lo que sirve.

Solo entonces se aplica a Workloads.

Paso 4 — SCP de protección de la evidencia sobre Prod:

```
{
  "Effect": "Deny",
  "Action": [
    "cloudtrail:StopLogging", "cloudtrail:DeleteTrail",
    "config:DeleteConfigurationRecorder", "config:StopConfigurationRecorder",
    "guardduty:DeleteDetector", "guardduty:DisassociateFromMasterAccount"
  ],
  "Resource": "*"
}
```

```
$ aws cloudtrail stop-logging --name org-trail   # desde cloudshop-prod
AccessDeniedException ... explicit deny in a service control policy   ✓
```

Paso 5 — verificación del aislamiento de cuotas, que era el fallo de la clase 017:

```
$ for c in cloudshop-prod cloudshop-dev; do
    printf "%-18s " $c
    aws service-quotas get-service-quota --service-code ec2 \
      --quota-code L-1216C47A --profile $c --query 'Quota.Value' --output text
  done
cloudshop-prod     256
cloudshop-dev      128
```

Cuotas independientes: un experimento en desarrollo ya no puede consumir la capacidad de producción.

Resultado, con las pruebas que lo demuestran:

```
cuentas                                        1          5      —
guardarraíles preventivos                      0          2      prueba negativa ✓
logs borrables por quien compromete la cuenta  sí         no     stop-logging denegado ✓
cuota compartida entre entornos                sí         no     256 y 128 separadas ✓
visibilidad multi-región                       no         sí     rastro de organización ✓
```

Lo que hizo útil el ejercicio no fue la estructura sino las pruebas negativas. La primera SCP parecía correcta por inspección y rompía STS; solo intentar la acción lo reveló.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-02-aws-core-platform/025-organizations-cuentas-ou-scp-y-landing-zone/lab.py
```

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

1. Lee exercise.steps y formula una predicción antes de ejecutar.
2. Ejecuta la práctica y verifica todos los elementos de checks.
3. Provoca el caso de negativetest y explica la señal observada.

4. Materializa landing-zone-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 una jerarquía de política, ownership y evidencia. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye landing-zone-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
Tras aplicar una SCP de regiones deja de funcionar IAM y no se puede quitar la política	Los servicios globales emiten contra us-east-1 y no se exceptuaron	Usa NotAction con iam, organizations, sts, route53, cloudfront y facturación; prueba siempre en sandbox primero.
Se añaden permisos a un rol y el acceso sigue denegado	Lo deniega una SCP, y ninguna concesión posterior la revierte	Lee el mensaje: «explicit deny in a service control policy» indica qué documento revisar.
Un atacante borra los registros de la cuenta que comprometió	CloudTrail escribía en la misma cuenta y nada impedía detenerlo	Cuenta de archivo separada, bloqueo de objetos y SCP que deniegue StopLogging y DeleteTrail.
La cuenta de gestión aloja cargas de trabajo	Es la única sin SCP posible: no tiene barrera superior	Deja en ella solo la organización y la facturación; mueve todo lo demás a cuentas bajo unidades organizativas.
Un guardarraíl configurado no impide la acción que debía impedir	Se verificó por inspección de la configuración y no por intento real	Prueba negativa automatizada y periódica, más prueba positiva de lo que debe seguir funcionando.

■ 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 cuenta de gestión no debe alojar cargas de trabajo?
2. Adjuntas una SCP con Allow: a una unidad organizativa. ¿Quién obtiene acceso y por qué?
3. ¿Qué servicios hay que exceptuar al restringir regiones, y qué ocurre concretamente si olvidas sts:?
4. ¿Qué dos propiedades debe tener la cuenta de archivo de registros para que la evidencia sobreviva a un compromiso?

5. ¿Por qué una prueba negativa no basta sin su prueba positiva correspondiente?

Referencias

- AWS (2024). Organizations: service control policies — evaluación, estrategias y límites. <https://docs.aws.amazon.com/organizations/latest/userguide/orgsmanagepoliciesscps.html>
- AWS (2024). Organizing your AWS environment using multiple accounts — cuentas de la base y unidades recomendadas. <https://docs.aws.amazon.com/whitepapers/latest/organizing-your-aws-environment/organizing-your-aws-environment.html>
- AWS (2024). Control Tower: how controls work — guardarraíles preventivos y detectivos. <https://docs.aws.amazon.com/controltower/latest/controlreference/controls.html>
- AWS (2024). CloudTrail: security best practices — rastro de organización, validación de integridad y bloqueo de objetos. <https://docs.aws.amazon.com/awscloudtrail/latest/userguide/best-practices-security.html>
- AWS (2024). AWS services that work with IAM — qué servicios son globales y cómo se comportan con condiciones de región. <https://docs.aws.amazon.com/IAM/latest/UserGuide/referenceaws-services-that-work-with-iam.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 02 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 024 · Proyecto: decisión de migración sustentada con ADR	Parte 02 · Programa	026 · IAM, roles, políticas, STS y federación →

Evaluación — 025 Organizations, cuentas, OU, SCP y landing zone

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 landing-zone-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.

026 — IAM, roles, políticas, STS y federación

Parte: 02 — AWS: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: iam · Estado: EXECUTABLECORE

Propósito

Dominar el modelo de identidad de AWS hasta poder predecir el resultado de una petición sin ejecutarla, y sustituir toda credencial de larga vida por identidad temporal. Es la clase que hace posible que en la parte 17 un pipeline despliegue sin secretos, y la que explica por qué la mayoría de brechas en AWS no son fallos de AWS.

Resultados de aprendizaje

Al finalizar podrás:

1. Reproducir el orden de evaluación de una petición y predecir el resultado ante políticas en conflicto.
2. Escribir una política con condiciones que acoten origen, etiqueta y presencia de MFA.
3. Configurar federación OIDC desde un pipeline con una condición de sujeto que impida la suplantación.
4. Usar límites de permisos para delegar creación de roles sin permitir escalada de privilegios.
5. Diagnosticar un acceso denegado leyendo el mensaje para saber qué documento revisar.

Conceptos centrales

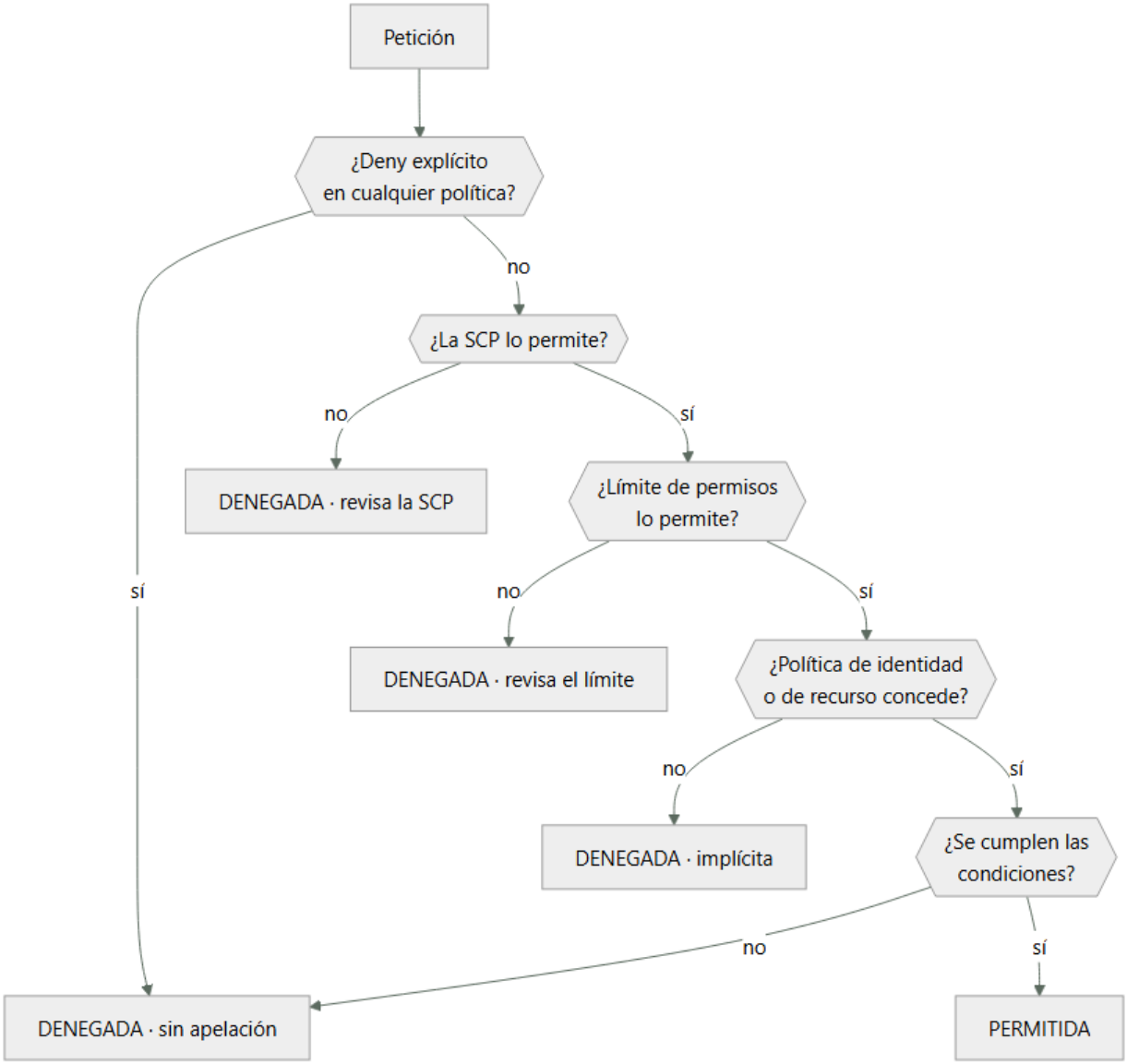
Concepto	Comprensión verificable
rol	Identidad sin credenciales permanentes que se asume temporalmente. Tiene dos documentos: la política de confianza —quién puede asumirlo— y las de permisos —qué puede hacer una vez dentro—.
política de confianza	Documento que declara qué principales pueden asumir el rol y bajo qué condiciones. Es la puerta de entrada, y un error aquí es más grave que uno en los permisos.
límite de permisos	Política adjunta a una identidad que define el techo de lo que puede hacer, aunque otras políticas concedan más. Permite delegar la creación de roles sin permitir escalada.
confused deputy	Ataque en el que un tercero legítimo es inducido a actuar contra un recurso ajeno. Se mitiga con condiciones sobre la cuenta de origen y el identificador externo.
clave de condición	Atributo evaluable de la petición: origen, hora, MFA, etiquetas, VPC. Convierte una política de «puede» en «puede, si además se cumple esto».

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

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 orden de evaluación, y cómo leer el error

AWS evalúa cada petición en una secuencia fija. Conocerla convierte «no tengo permiso» en un diagnóstico de un minuto:

1. Deny explícito en cualquier política → denegada, sin excepción.
2. SCP de la organización → si no lo permite, denegada.
3. Límite de permisos, si existe → si no lo permite, denegada.
4. Política de identidad o de recurso → alguna debe conceder.
5. Condiciones → deben cumplirse todas.

El mensaje de error dice exactamente dónde mirar, y casi nadie lo lee:

```
"...with an explicit deny in a service control policy"
  → revisa las SCP. Añadir permisos a la identidad no hará nada.

"...with an explicit deny in a permissions boundary"
  → revisa el límite de permisos del rol.

"...with an explicit deny in an identity-based policy"
```

```
  → hay un Deny en la política del rol o del usuario.

"...because no identity-based policy allows the action"
  → denegación implícita: falta conceder. Este es el único caso en que
    añadir un permiso resuelve el problema.
```

Solo el cuarto caso se arregla concediendo. Los tres primeros exigen tocar otro documento, y perseguirlos con más permisos es la causa habitual de que un rol acabe con AdministratorAccess.

Hay un matiz sobre acceso entre cuentas: cuando el recurso está en otra cuenta hacen falta las dos políticas —la de identidad en la cuenta que llama y la de recurso en la que responde—. Una sola nunca basta, y el mensaje no siempre distingue cuál falta.

2. Condiciones: donde una política deja de ser un cheque en blanco

Un permiso sin condiciones es válido desde cualquier lugar, a cualquier hora y en cualquier contexto. Las claves de condición acotan eso:

```
{
  "Effect": "Allow",
  "Action": ["s3:GetObject", "s3:PutObject"],
  "Resource": "arn:aws:s3:::cloudshop-facturas/*",
  "Condition": {
    "Bool": {"aws:MultiFactorAuthPresent": "true"},
    "StringEquals": {"aws:PrincipalTag/equipo": "finanzas"},
    "IpAddress": {"aws:SourceIp": ["203.0.113.0/24"]},
    "DateLessThan": {"aws:CurrentTime": "2026-12-31T23:59:59Z"}
  }
}
```

Las cuatro condiciones se combinan con Y lógico: todas deben cumplirse. Dentro de una misma clave con varios valores, la relación es O.

Dos condiciones merecen atención especial:

aws:PrincipalTag habilita control de acceso por atributos: en vez de una política por equipo, una sola política que compara la etiqueta del principal con la del recurso. Escala mucho mejor que enumerar.

aws:SourceIp no funciona como se espera con endpoints de VPC. Cuando la llamada viaja por un endpoint, la IP de origen es privada y la condición falla. Para ese caso hay que usar aws:SourceVpce o aws:SourceVpc, y es un fallo que se descubre justo después de endurecer la red.

Y una condición que evita el confused deputy al conceder acceso a un tercero:

```
"Condition": {"StringEquals": {"sts:ExternalId": "identificador-unico-por-cliente"}}
```

Sin ella, un proveedor que gestiona varias cuentas podría ser inducido a actuar contra la tuya usando su propio rol legítimo.

3. Federación OIDC: desplegar sin ningún secreto

El mecanismo que elimina las claves de acceso de los pipelines, aplicado a AWS. El intercambio, sin secreto compartido en ningún punto:

```
1. El pipeline pide a su plataforma un token OIDC que declara quién es.
2. Llama a sts:AssumeRoleWithWebIdentity presentando ese token.
3. AWS verifica la firma contra las claves públicas del emisor y comprueba
   que las declaraciones cumplen la política de confianza.
4. Devuelve credenciales temporales de 1 hora.
```

La política de confianza es donde se juega todo:

```
{
  "Effect": "Allow",
  "Principal": {"Federated": "arn:aws:iam::123456789012:oidc-provider/token.actions.githubusercontent.
com"},
  "Action": "sts:AssumeRoleWithWebIdentity",
  "Condition": {
    "StringEquals": {
      "token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
      "token.actions.githubusercontent.com:sub": "repo:miorg/cloudshop:environment:produccion"
    }
```

```
  }
}
```

Sin la condición sobre sub, cualquier repositorio de GitHub del mundo puede asumir el rol. La firma será válida porque procede del mismo emisor; lo único que distingue a tu repositorio de otro es esa declaración.

Los errores por grado de peligro:

```
sin condición sobre sub                    → cualquiera, en cualquier parte
"sub": "repo:miorg/*"                      → cualquier repositorio de tu organización
StringLike "repo:miorg/cloudshop:*"        → cualquier rama, incluida una de un fork
StringEquals "...:environment:produccion"  → correcto
```

Usar environment en lugar de ref añade una capa: los entornos de GitHub admiten revisores obligatorios, así que el rol solo es asumible tras una aprobación humana.

4. Límites de permisos: delegar sin permitir escalada

El problema: quieres que los equipos creen sus propios roles sin abrir la puerta a que se concedan AdministratorAccess. Conceder iam:CreateRole y iam:AttachRolePolicy es, de hecho, conceder administrador.

Un límite de permisos define el techo de lo que una identidad puede hacer, con independencia de lo que sus políticas concedan:

```
// Límite: nada fuera de estos servicios, y prohibido tocar IAM salvo lo mínimo
{
  "Version": "2012-10-17",
  "Statement": [
    {"Effect": "Allow", "Action": ["s3:*", "dynamodb:*", "logs:*", "sqs:*"], "Resource": "*"},
    {"Effect": "Deny", "Action": ["iam:*", "organizations:*", "account:*"], "Resource": "*"}
  ]
}
```

Y la política que permite delegar obligando a aplicar ese límite:

```
{
  "Effect": "Allow",
  "Action": ["iam:CreateRole", "iam:AttachRolePolicy"],
  "Resource": "arn:aws:iam::*:role/equipos/*",
  "Condition": {
    "StringEquals": {"iam:PermissionsBoundary": "arn:aws:iam::123456789012:policy/limite-equipos"}
  }
}
```

La condición es la clave: solo se pueden crear roles que lleven el límite adjunto. Sin ella, el equipo crearía roles sin límite y la delegación sería una escalada.

Falta una pieza para cerrarlo del todo:

```
{"Effect": "Deny", "Action": ["iam:DeleteRolePermissionsBoundary", "iam:PutRolePermissionsBoundary"],
"Resource": "*"}
```

Sin esta denegación, el equipo podría crear el rol con límite y quitárselo después. Es el hueco que convierte un diseño correcto en uno inútil, y no es evidente hasta que alguien lo prueba.

5. Privilegio mínimo con las herramientas de AWS

Adivinar la lista de permisos produce o bien exceso o bien bloqueo. AWS ofrece tres herramientas para partir de lo observado:

```
# 1. Qué servicios ha usado realmente un rol
$ aws iam generate-service-last-accessed-details --arn arn:aws:iam::...:role/ci-deploy
$ aws iam get-service-last-accessed-details --job-id ... \
    --query 'ServicesLastAccessed[?TotalAuthenticatedEntities>`0`].[ServiceName,LastAuthenticated]'

# 2. Generar una política a partir de la actividad de CloudTrail
$ aws accessanalyzer start-policy-generation --policy-generation-details \
    '{"principalArn":"arn:aws:iam::...:role/ci-deploy"}' --cloud-trail-details ...

# 3. Validar la política antes de aplicarla
$ aws accessanalyzer validate-policy --policy-document file://politica.json \
    --policy-type IDENTITY_POLICY --query 'findings[?findingType==`SECURITY_WARNING`]'
```

La tercera detecta patrones peligrosos que pasan desapercibidos: comodines en el principal, iam:PassRole sin restricción de recurso, o acciones que permiten escalada.

iam:PassRole merece atención propia. Permite entregar un rol a un servicio, y sin restricción de recurso equivale a conceder cualquier permiso que exista en la cuenta: basta con pasar un rol de administrador a una función o a una instancia que tú controlas.

```
// Peligroso
{"Effect": "Allow", "Action": "iam:PassRole", "Resource": "*"}

// Correcto: solo roles concretos y solo al servicio esperado
{
  "Effect": "Allow", "Action": "iam:PassRole",
  "Resource": "arn:aws:iam::*:role/cloudshop-tarea-*",
  "Condition": {"StringEquals": {"iam:PassedToService": "ecs-tasks.amazonaws.com"}}
}
```

Y el límite del método basado en registros, ya señalado en la clase 018: los caminos de error usan permisos distintos. Un rol que escribe en una cola pero no en su cola de mensajes fallidos funciona hasta el primer fallo.

Ejemplo trabajado

El pipeline de CloudShop despliega en producción con una clave de acceso de 2 años y medio y PowerUserAccess. Se migra a OIDC con privilegio mínimo, verificando cada paso.

Estado inicial:

```
$ aws iam list-access-keys --user-name ci-deploy \
    --query 'AccessKeyMetadata[0].[CreateDate,Status]' --output text
2023-03-14T10:22:00Z   Active
$ aws iam list-attached-user-policies --user-name ci-deploy --query 'AttachedPolicies[].PolicyName'
["PowerUserAccess"]
```

```
antigüedad          2 años 5 meses, 0 rotaciones
copias conocidas    secreto del repositorio, 2 portátiles, 1 runbook
permisos            ~8.000 acciones
ventana si se filtra ilimitada
```

Paso 1 — proveedor OIDC y prueba de la política de confianza insegura, para verla fallar:

```
$ aws iam create-open-id-connect-provider \
    --url https://token.actions.githubusercontent.com --client-id-list sts.amazonaws.com
```

Con la condición solo sobre aud, desde un repositorio de laboratorio ajeno a la organización:

```
$ aws sts assume-role-with-web-identity --role-arn ...:role/ci-deploy-oidc ...
{"Credentials": {...}}          ← ÉXITO desde un repositorio ajeno
```

Se corrige acotando el sujeto al entorno:

```
$ aws iam update-assume-role-policy --role-name ci-deploy-oidc \
    --policy-document file://confianza.json    # sub: repo:miorg/cloudshop:environment:produccion
```

Pruebas negativas y positiva:

```
desde otro repositorio                → AccessDenied  ✓
desde miorg/cloudshop, rama de trabajo → AccessDenied  ✓
desde miorg/cloudshop, entorno prod    → credenciales de 1 h  ✓
```

Paso 2 — recortar permisos con lo observado en 30 días:

```
$ aws accessanalyzer start-policy-generation --policy-generation-details \
    '{"principalArn":"arn:aws:iam::123456789012:user/ci-deploy"}'
$ aws accessanalyzer get-generated-policy --job-id ... \
    --query 'generatedPolicyResult.generatedPolicies[0].policy' | jq -r '.Statement[].Action' | wc -l
34
```

```
permisos concedidos antes    ~8.000 acciones
acciones realmente usadas        34
servicios usados                  5   (s3, cloudfront, ecs, ecr, logs)
```

Se ejercitan también los caminos de error antes de aplicar. Aparecen dos acciones más que ningún despliegue correcto usaba:

```
ecs:StopTask                al abortar un despliegue fallido
logs:PutLogEvents sobre el grupo de errores
```

Paso 3 — validar la política antes de aplicarla:

```
$ aws accessanalyzer validate-policy --policy-document file://ci-deploy.json \
    --policy-type IDENTITY_POLICY \
    --query 'findings[?findingType==`SECURITY_WARNING`].[issueCode,learnMoreLink]' --output text
PASS_ROLE_WITH_STAR_IN_RESOURCE   https://docs.aws.amazon.com/...
```

El validador detecta el fallo que la generación automática no evita: iam:PassRole con Resource: "", que permite pasar cualquier rol —incluido uno de administrador— a un servicio controlado por el pipeline. Se acota:

```
{
  "Effect": "Allow", "Action": "iam:PassRole",
  "Resource": "arn:aws:iam::*:role/cloudshop-tarea-*",
  "Condition": {"StringEquals": {"iam:PassedToService": "ecs-tasks.amazonaws.com"}}
}
```

```
$ aws accessanalyzer validate-policy ... --query 'findings[?findingType==`SECURITY_WARNING`]'
[]                                                              ✓
```

Paso 4 — retirar la credencial, con observación previa:

```
$ aws iam update-access-key --user-name ci-deploy --access-key-id AKIA... --status Inactive
# 7 días sin fallos ni llamadas registradas con esa clave
$ aws iam delete-access-key --user-name ci-deploy --access-key-id AKIA...
$ aws iam delete-user --user-name ci-deploy
```

Resultado:

```
credencial               permanente, 2,5 años   token de 1 h
secretos almacenados     4 copias               0
acciones permitidas      ~8.000                 36
iam:PassRole             sin restricción        1 patrón, 1 servicio
superficie si se filtra  ilimitada              1 h, solo ese repo y entorno
```

El paso que más aportó fue el validador, no la generación automática: la política generada a partir del uso real contenía un PassRole sin acotar que habría permitido escalar a administrador desde el propio pipeline.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-02-aws-core-platform/026-iam-roles-politicas-sts-y-federacion/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 politica-iam-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 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 politica-iam-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
Cualquier repositorio de GitHub puede asumir el rol de despliegue	La política de confianza no condiciona la declaración sub	Condiciona con StringEquals sobre repositorio y entorno; repo:org/ o StringLike con comodín no bastan.
Se conceden más permisos y el acceso sigue denegado	El deny está en una SCP o en un límite de permisos	Lee el mensaje de error: nombra el documento exacto que deniega.
Una política endurecida con aws:SourceIp rompe el acceso desde la VPC	A través de un endpoint de VPC la IP de origen es privada	Usa aws:SourceVpce o aws:SourceVpc para tráfico que viaja por endpoints.
Un equipo con permiso para crear roles se concede administrador	Se delegó iam:CreateRole sin exigir límite de permisos ni impedir retirarlo	Condiciona con iam:PermissionsBoundary y deniega Put/DeleteRolePermissionsBoundary.
Un pipeline con permisos mínimos consigue privilegios de administrador	iam:PassRole con Resource permite pasar cualquier rol a un servicio propio	Acota PassRole por patrón de recurso y por iam:PassedToService.

■ 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. Un error dice «explicit deny in a service control policy». ¿Sirve de algo añadir permisos al rol? ¿Qué documento revisas?
2. ¿Por qué aws:SourceIp deja de funcionar cuando el tráfico pasa por un endpoint de VPC, y qué clave se usa entonces?
3. ¿Qué dos piezas hacen falta para delegar la creación de roles sin permitir escalada de privilegios?
4. ¿Por qué iam:PassRole con Resource: "" equivale a conceder administrador?
5. ¿Qué distingue exactamente tu repositorio del de un atacante en un intercambio OIDC?

Referencias

- AWS (2024). Policy evaluation logic — orden de evaluación completo con SCP y límites de permisos. <https://docs.aws.amazon.com/IAM/latest/UserGuide/referencepoliciesevaluation-logic.html>
- AWS (2024). Global condition context keys — catálogo de claves de condición y su comportamiento. <https://docs.aws.amazon.com/IAM/latest/UserGuide/referencepoliciescondition-keys.html>
- AWS (2024). Permissions boundaries for IAM entities — delegación segura de creación de roles. <https://docs.aws.amazon.com/IAM/latest/UserGuide/accesspoliciesboundaries.html>
- AWS (2024). IAM Access Analyzer policy validation — comprobaciones de seguridad antes de aplicar una política. <https://docs.aws.amazon.com/IAM/latest/UserGuide/access-analyzer-policy-validation.html>
- Hardt, D., ed. (2012). RFC 6749: The OAuth 2.0 Authorization Framework — base del intercambio de token por credenciales. <https://www.rfc-editor.org/rfc/rfc6749>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Evaluación — 026 IAM, roles, políticas, STS y federación

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 politica-iam-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.

027 — VPC, subredes, rutas, NAT, endpoints y seguridad

Parte: 02 — AWS: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: network · Estado: EXECUTABLECORE

Propósito

Diseñar una VPC entendiendo qué componente decide cada cosa: la tabla de rutas determina por dónde sale el tráfico, el grupo de seguridad quién puede hablar, y la elección entre NAT y endpoint decide una partida de la factura que suele sorprender. Es la base de las clases 028 a 036 y el origen de la mayoría de incidentes de conectividad del programa.

Resultados de aprendizaje

Al finalizar podrás:

1. Planificar un rango CIDR que admita crecimiento y no colisione con la red corporativa ni con futuros pares.
2. Determinar si una subred es pública o privada por su tabla de rutas, no por su nombre.
3. Elegir entre grupo de seguridad y ACL de red sabiendo cuál tiene estado y cuál no.
4. Calcular el ahorro de sustituir tráfico por NAT gateway por endpoints de VPC.
5. Diagnosticar un fallo de conectividad recorriendo las capas en orden y con los registros de flujo.

Conceptos centrales

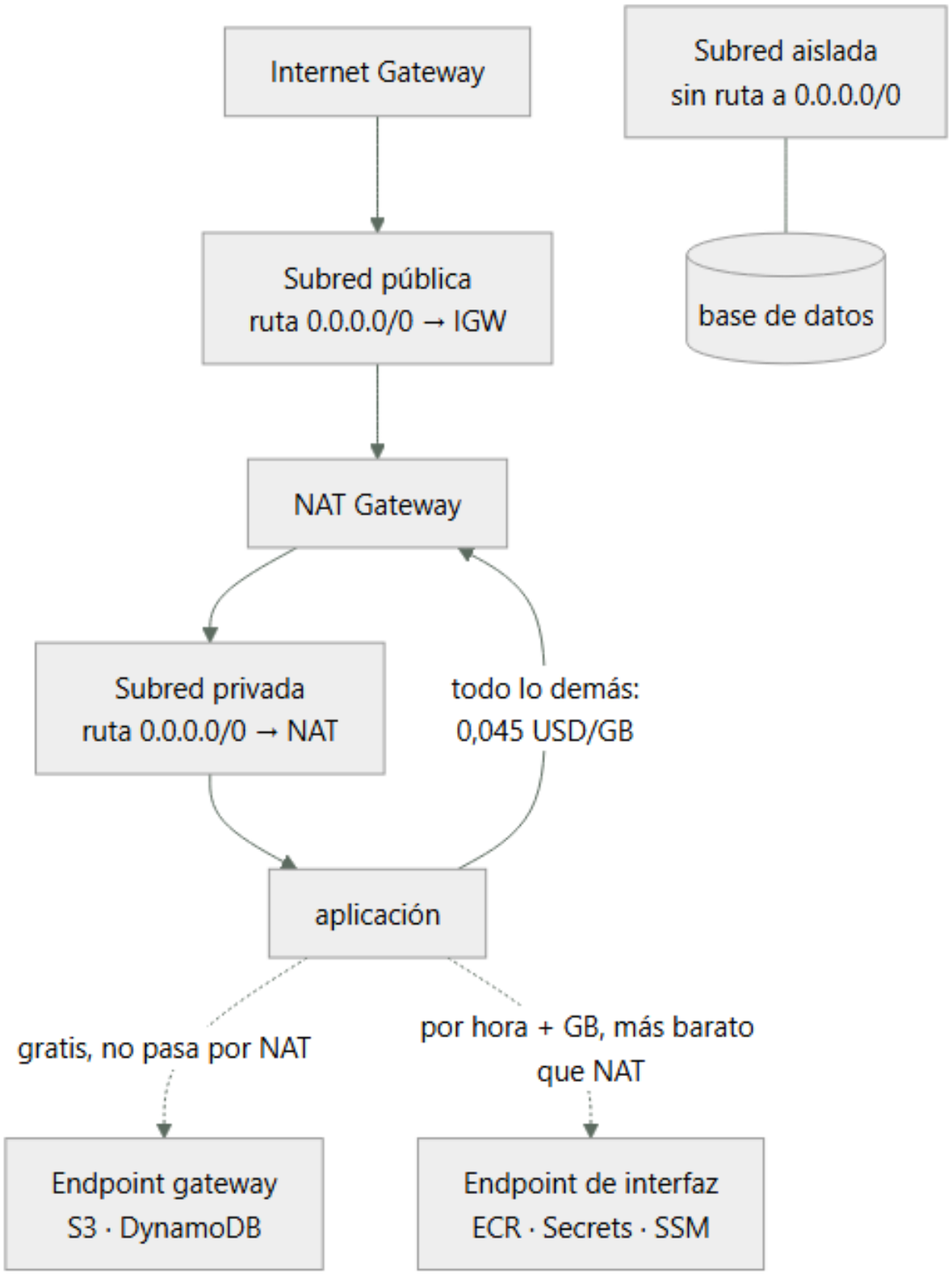
Concepto	Comprensión verificable
tabla de rutas	Conjunto de reglas que decide por dónde sale el tráfico de una subred. Es lo único que hace pública a una subred: una ruta hacia una puerta de enlace de internet.
grupo de seguridad	Cortafuegos con estado a nivel de interfaz de red. Solo tiene reglas de permiso y la respuesta al tráfico permitido vuelve automáticamente, sin regla de salida que la autorice.
ACL de red	Cortafuegos sin estado a nivel de subred. Admite denegaciones explícitas, y al no tener estado exige regla de vuelta para los puertos efímeros.
NAT gateway	Servicio que permite salida a internet desde subredes privadas. Cobra por hora y por GB procesado, lo que lo convierte en una de las partidas más caras y menos vigiladas.
endpoint de VPC	Ruta privada hacia un servicio de AWS sin pasar por internet. Los de tipo gateway —S3 y DynamoDB— son gratuitos; los de interfaz cobran por hora y por GB, pero menos que el NAT.

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

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. Planificar el CIDR antes de crear nada

El rango de una VPC no se puede cambiar; solo se pueden añadir bloques secundarios, con restricciones. Un error aquí se paga durante años.

Tres reglas que evitan los problemas más caros:

1. No solapar con nada con lo que puedas necesitar hablar. Si la red corporativa usa 10.0.0.0/8 y creas la VPC en 10.0.0.0/16, la VPN o el emparejamiento serán imposibles: el enrutamiento no admite solapamiento. Un registro central de rangos asignados es más barato que una migración.

2. Dimensionar con holgura. AWS admite de /16 a /28. Un /16 da 65.536 direcciones y no cuesta nada reservarlo; un /24 se agota en cuanto se añaden zonas o servicios que consumen direcciones.

3. Contar las direcciones que AWS reserva. En cada subred se reservan 5, no 2:

```
Subred 10.20.1.0/24 → 256 direcciones teóricas
  .0    dirección de red
  .1    puerta de enlace de la VPC
  .2    servidor DNS
  .3    reservada para uso futuro
  .255  difusión
  ----------------------------
  251 direcciones utilizables
```

Un esquema que crece sin colisiones:

```
VPC produccion     10.20.0.0/16
  pública    zona a  10.20.0.0/20    4.091 utilizables
  pública    zona b  10.20.16.0/20
  privada    zona a  10.20.32.0/20
  privada    zona b  10.20.48.0/20
  aislada    zona a  10.20.64.0/20
  aislada    zona b  10.20.80.0/20
  libre              10.20.96.0/19   reservado para crecer
```

El bloque libre al final no es desperdicio: es lo que permite añadir una tercera zona sin rehacer el esquema.

2. Pública o privada lo decide la tabla de rutas

Los nombres «pública» y «privada» son convención humana. Lo único que hace pública a una subred es una ruta hacia una puerta de enlace de internet:

```
Subred pública        0.0.0.0/0 → igw-xxxx
Subred privada        0.0.0.0/0 → nat-xxxx
Subred aislada        sin ruta hacia 0.0.0.0/0
```

De ahí un fallo que aparece con regularidad: una subred llamada «privada» con ruta al IGW es pública, y cualquier recurso con IP pública en ella es accesible desde internet. El nombre no protege nada.

La comprobación es directa:

```
$ aws ec2 describe-route-tables \
    --filters Name=association.subnet-id,Values=subnet-0a1b2c \
    --query 'RouteTables[0].Routes[?DestinationCidrBlock==`0.0.0.0/0`].GatewayId' --output text
igw-0f9e8d          # es PÚBLICA, se llame como se llame
```

Y hay una segunda condición para que una instancia sea alcanzable desde internet: necesita IP pública además de la ruta. Una instancia sin IP pública en una subred pública no es accesible desde fuera, aunque sí puede salir si hay NAT.

La tercera categoría —aislada, sin ruta a internet en absoluto— es donde deben vivir las bases de datos. No es lo mismo que privada: una subred privada puede iniciar conexiones salientes a internet a través del NAT, y eso es exactamente lo que usa un atacante para exfiltrar datos o descargar herramientas.

3. Grupo de seguridad y ACL: con estado y sin estado

La diferencia decisiva:

	Grupo de seguridad	ACL de red
Ámbito	Interfaz de red	Subred entera
Estado	Con estado	Sin estado
Reglas	Solo permitir	Permitir y denegar
Evaluación	Todas las reglas	Por orden de número
Referencia a otro grupo	Sí	No, solo CIDR

Con estado significa que la respuesta al tráfico permitido vuelve automáticamente. Si un grupo permite entrada al 443, la respuesta sale sin necesidad de regla de salida.

Sin estado significa que hay que autorizar ambos sentidos. Este es el error clásico de las ACL:

```
ACL con entrada 443 permitida y salida solo 443:
  petición entrante al 443          ✓ permitida
  respuesta desde el 443 hacia el
  puerto efímero del cliente (52341) ✗ BLOQUEADA
```

La conexión se establece y se cuelga. Hay que permitir la salida hacia el rango efímero 1024-65535, y es la causa habitual de que «el firewall parece bien configurado y no funciona».

La capacidad de referenciar otro grupo de seguridad en vez de un CIDR es lo que hace mantenible el diseño:

```
$ aws ec2 authorize-security-group-ingress --group-id sg-db \
    --protocol tcp --port 5432 --source-group sg-app
```

Esto dice «lo que esté en sg-app puede hablar con la base de datos», y sigue siendo cierto cuando las instancias cambian de IP, se escalan o se sustituyen. Un CIDR fijo exige mantenimiento cada vez que la topología cambia.

La recomendación práctica: grupos de seguridad para casi todo, ACL solo para denegaciones amplias —bloquear un rango de IP concreto—, porque su falta de estado las hace fáciles de configurar mal.

4. NAT gateway: la partida cara que nadie mira

Un NAT gateway cobra dos veces: por hora de existencia y por GB procesado. La segunda es la que sorprende.

```
precio por hora            ~0,045 USD  →  32,85 USD/mes
precio por GB procesado    ~0,045 USD
```

Con 8 TB mensuales de tráfico saliente hacia servicios de AWS:

```
8.000 GB × 0,045 = 360 USD/mes solo de procesamiento
más 32,85 de la hora, por cada NAT
con un NAT por zona (3 zonas): 98,55 + 360 = 458,55 USD/mes
```

Y lo importante: buena parte de ese tráfico no necesita salir a internet. Descargar imágenes de ECR, leer secretos, escribir logs o hablar con S3 son llamadas a servicios de AWS que un endpoint resuelve por la red interna.

```
tráfico por NAT gateway            0,045 USD
tráfico por endpoint de interfaz   0,010 USD
tráfico por endpoint gateway       0,000 USD   ← S3 y DynamoDB
```

El endpoint gateway es gratuito y se instala como una ruta en la tabla; no tener uno para S3 es dinero regalado sin ninguna contrapartida. El de interfaz cuesta 0,01 USD por hora por zona más 0,01 por GB, así que compensa a partir de un volumen modesto:

```
umbral del endpoint de interfaz (1 zona):
  7,30 USD/mes de coste fijo / (0,045 − 0,010) USD/GB ≈ 209 GB/mes
```

Por encima de 209 GB mensuales hacia ese servicio, el endpoint sale más barato. Y además del ahorro, aporta seguridad: el tráfico no sale de la red de AWS, lo que permite subredes aisladas que hablan con los servicios que necesitan sin ninguna ruta a internet.

5. Diagnóstico de conectividad, de abajo hacia arriba

El orden ahorra horas porque cada paso descarta una capa completa:

```
# 1. ¿Existe ruta hacia el destino?
$ aws ec2 describe-route-tables --filters Name=association.subnet-id,Values=subnet-xxx \
    --query 'RouteTables[0].Routes[].[DestinationCidrBlock,GatewayId,NatGatewayId]' --output text

# 2. ¿Lo permite la ACL de la subred, en AMBOS sentidos?
$ aws ec2 describe-network-acls --filters Name=association.subnet-id,Values=subnet-xxx \
    --query 'NetworkAcls[0].Entries[?RuleNumber<`32767`]'

# 3. ¿Lo permite el grupo de seguridad de origen y el de destino?
$ aws ec2 describe-security-groups --group-ids sg-app sg-db \
    --query 'SecurityGroups[].[GroupId,IpPermissions[].FromPort]'

# 4. Analizador de accesibilidad: evalúa la ruta completa
$ aws ec2 create-network-insights-path --source i-origen --destination i-destino \
    --protocol tcp --destination-port 5432
```

```
$ aws ec2 start-network-insights-analysis --network-insights-path-id nip-xxx
```

El paso 4 es el más rentable y el menos usado: evalúa la configuración completa y dice qué componente bloquea, sin necesidad de tráfico real.

Y cuando la configuración parece correcta pero algo falla, los registros de flujo dan el veredicto:

```
$ aws logs filter-log-events --log-group-name /aws/vpc/flowlogs \
    --filter-pattern '[version, account, eni, source, destination, srcport,
                       destport=5432, protocol, packets, bytes, start, end,
                       action=REJECT, status]' --max-items 5
```

Un REJECT indica que una regla lo bloqueó y hay que revisar grupos y ACL. La ausencia total de registros es un diagnóstico distinto y más útil: significa que el paquete nunca llegó a la interfaz, así que el problema está en enrutamiento o resolución de nombres, no en el firewall.

Ejemplo trabajado

La factura de red de CloudShop es de 1.180 USD/mes y nadie sabe explicarla. Además, la base de datos está en una subred llamada «privada». Se auditan ambas cosas.

Parte 1 — de dónde viene el gasto:

```
$ aws ce get-cost-and-usage --time-period Start=2026-07-01,End=2026-08-01 \
    --granularity MONTHLY --metrics UnblendedCost \
    --filter '{"Dimensions":{"Key":"USAGE_TYPE_GROUP","Values":["EC2: NAT Gateway"]}}' \
    --query 'ResultsByTime[0].Total.UnblendedCost.Amount' --output text
842.31
```

842 USD de 1.180 son NAT gateway. Se desglosa qué tráfico lo atraviesa con los registros de flujo:

```
destino                        GB/mes    % del tráfico por NAT
S3 (misma región)               9.140          49 %
ECR (descarga de imágenes)      4.620          25 %
Secrets Manager y SSM             310           2 %
CloudWatch Logs                 1.850          10 %
internet real (APIs de terceros) 2.680         14 %
                               ------
total                          18.600 GB
```

El 86 % del tráfico que paga NAT no va a internet: va a servicios de AWS de la misma región.

Cálculo del cambio a endpoints:

```
actual
  3 NAT × 32,85 USD/mes de hora            =    98,55
  18.600 GB × 0,045 USD                    =   837,00
                                              --------
                                               935,55

con endpoints
  gateway S3 (gratis)          9.140 GB   →      0,00
  interfaz ECR   3 zonas × 7,30 + 4.620 × 0,010 = 68,10
  interfaz Secrets+SSM  2 svc × 3 × 7,30 + 310 × 0,010 = 46,90
  interfaz Logs  3 × 7,30 + 1.850 × 0,010  =    40,40
  NAT solo para internet real
    3 × 32,85 + 2.680 × 0,045              =   219,15
                                              --------
                                               374,55

ahorro: 935,55 − 374,55 = 561 USD/mes  (−60 %)
```

Se verifica el umbral antes de crear cada endpoint de interfaz, para no añadir coste fijo sin retorno:

```
Secrets Manager: 310 GB/mes
  coste fijo 3 zonas: 21,90 USD
  ahorro por GB: 310 × (0,045 − 0,010) = 10,85 USD
  → NO compensa por volumen, pero SÍ por seguridad: permite que la subred
    aislada lea secretos sin ninguna ruta a internet. Se crea igualmente,
    con el sobrecoste de 11 USD/mes declarado como decisión de seguridad.
```

Parte 2 — la subred «privada» de la base de datos:

```
$ aws ec2 describe-route-tables \
    --filters Name=association.subnet-id,Values=subnet-0db1 \
```

```
    --query 'RouteTables[0].Routes[].[DestinationCidrBlock,GatewayId,NatGatewayId]' --output text
10.20.0.0/16    local     None
0.0.0.0/0       None      nat-0a7f
```

No es pública —sale por NAT, no por IGW— pero sí puede iniciar conexiones salientes a internet. Para una base de datos eso es un camino de exfiltración que no aporta nada:

```
$ aws ec2 create-route-table --vpc-id vpc-0c1d --tag-specifications \
    'ResourceType=route-table,Tags=[{Key=Name,Value=aislada}]'
# sin ruta 0.0.0.0/0; solo local y los endpoints necesarios
$ aws ec2 associate-route-table --route-table-id rtb-nueva --subnet-id subnet-0db1
```

Prueba negativa desde la instancia de base de datos:

```
$ curl -m 5 https://example.com
curl: (28) Connection timed out                    ✓ sin salida a internet
$ aws secretsmanager get-secret-value --secret-id cloudshop/db --query Name
"cloudshop/db"                                     ✓ sigue accediendo por endpoint
```

Resultado:

```
coste de red                        1.180 USD     619 USD    (−48 %)
tráfico por NAT                    18.600 GB     2.680 GB
salida a internet desde la BD          sí            no
endpoints                               0             5
```

El diagnóstico no fue de red sino de rutas. El 86 % del gasto venía de que todo el tráfico interno salía por un componente pensado para internet, y la base de datos tenía un camino de salida que nadie había decidido darle: lo heredó de la tabla de rutas por defecto.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-02-aws-core-platform/027-vpc-subredes-rutas-nat-endpoints-y-seguridad/lab.py
```

El laboratorio selecciona el motor de práctica network 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 vpc-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 una topología con rutas, puertos y controles. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye vpc-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
Una subred llamada «privada» resulta accesible desde internet	Su tabla de rutas apunta a la puerta de enlace de internet: el nombre no decide nada	Verifica la ruta 0.0.0.0/0; pública, privada y aislada se distinguen solo por ahí.
Una conexión se establece y se cuelga pese a que el firewall permite el puerto	La ACL de red no tiene estado y falta permitir la vuelta por el rango efímero	Autoriza 1024-65535 en el sentido de retorno, o usa grupos de seguridad, que sí tienen estado.
La factura de NAT gateway es la mayor partida de red	El tráfico hacia servicios de AWS de la misma región atraviesa el NAT y paga 0,045 USD/GB	Endpoint gateway para S3 y DynamoDB (gratis) y de interfaz por encima de 209 GB/mes.
No se puede ampliar el rango de la VPC ni emparejar con la red corporativa	El CIDR se solapa y no se puede cambiar tras la creación	Planifica el rango contra un registro central antes de crear nada; deja bloques libres para crecer.
Una base de datos puede iniciar conexiones salientes a internet	Está en subred privada con ruta al NAT, heredada de la tabla por defecto	Usa subred aislada sin ruta a 0.0.0.0/0 y endpoints para lo que necesite; verifica con prueba negativa.

■ 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. ¿Cuántas direcciones utilizables tiene una subred /24 en AWS y por qué no son 254?
2. ¿Qué convierte a una subred en pública, y basta con eso para que una instancia sea alcanzable desde internet?
3. Una ACL permite entrada al 443 y salida solo por el 443. ¿Qué ocurre y por qué?
4. A partir de cuántos GB mensuales compensa un endpoint de interfaz frente al NAT, y de dónde sale ese número?
5. Los registros de flujo no muestran ninguna entrada para una conexión fallida. ¿Qué te dice eso frente a un REJECT?

Referencias

- AWS (2024). VPC user guide: subnets and routing — direcciones reservadas y tablas de rutas. <https://docs.aws.amazon.com/vpc/latest/userguide/configure-subnets.html>
- AWS (2024). Compare security groups and network ACLs — con estado frente a sin estado. <https://docs.aws.amazon.com/vpc/latest/userguide/infrastructure-security.html>
- AWS (2024). AWS PrivateLink and VPC endpoints — tipos de endpoint, precios y restricciones. <https://docs.aws.amazon.com/vpc/latest/privatelink/what-is-privatelink.html>
- AWS (2024). VPC Flow Logs — formato de registro y campos para diagnóstico. <https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html>
- AWS (2024). Reachability Analyzer — análisis estático de la ruta entre dos recursos. <https://docs.aws.amazon.com/vpc/latest/reachability/what-is-reachability-analyzer.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 02 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 026 · IAM, roles, políticas, STS y federación	Parte 02 · Programa	028 · EC2, AMI, EBS y selección de capacidad →

Evaluación — 027 VPC, subredes, rutas, NAT, endpoints y seguridad

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 vpc-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.

028 — EC2, AMI, EBS y selección de capacidad

Parte: 02 — AWS: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: compute · Estado: EXECUTABLECORE

Propósito

Elegir capacidad de cómputo con criterio medible en vez de por costumbre. La familia de instancia, el tipo de volumen y el modelo de compra son tres decisiones independientes, y equivocarse en cualquiera produce o bien un cuello de botella invisible o bien una factura que triplica lo necesario.

Resultados de aprendizaje

Al finalizar podrás:

1. Descifrar el nombre de una instancia y deducir familia, generación, procesador y capacidades adicionales.
2. Elegir tipo de volumen a partir de IOPS, throughput y patrón de acceso, no por tamaño.
3. Detectar el agotamiento de créditos de una instancia ráfaga y decidir entre modo ilimitado o cambiar de familia.
4. Calcular el umbral de utilización a partir del cual compensa un plan de ahorro frente a bajo demanda.
5. Construir una AMI reproducible y explicar por qué el estado local es el enemigo de la elasticidad.

Conceptos centrales

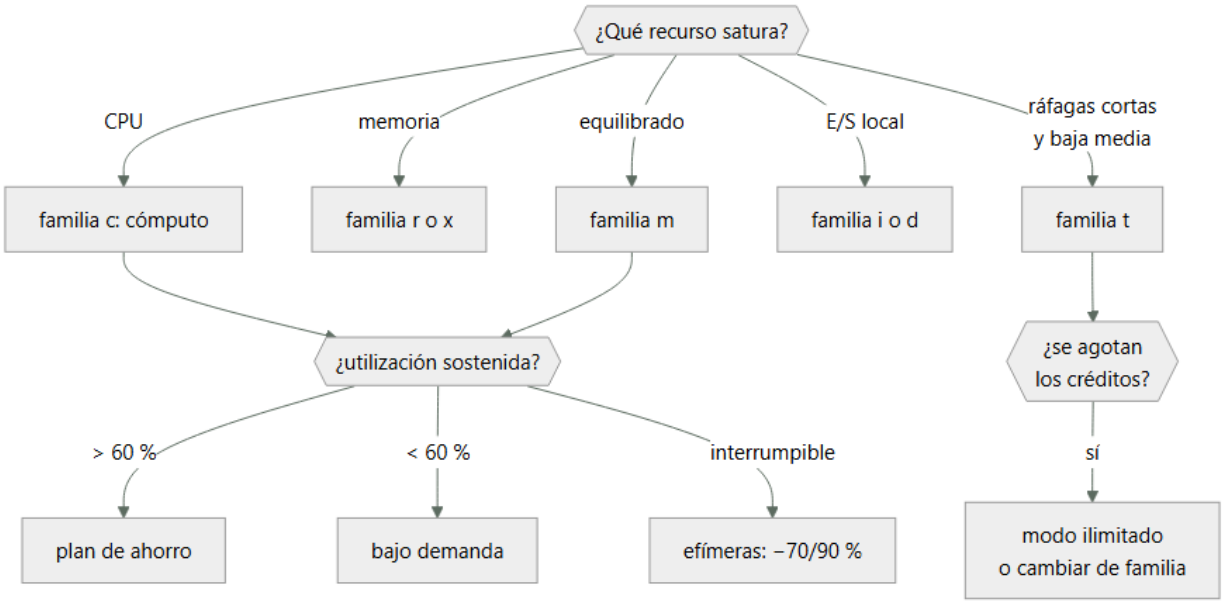
Concepto	Comprensión verificable
familia de instancia	Perfil de recursos optimizado para un uso: propósito general, cómputo, memoria, almacenamiento o aceleración. Elegir mal la familia produce que se pague de más en la dimensión que no se usa.
crédito de CPU	Saldo que acumulan las instancias ráfaga mientras están ociosas y gastan al trabajar. Al agotarse, el rendimiento cae a la línea base sin que ninguna métrica de la aplicación lo explique.
IOPS	Operaciones de entrada/salida por segundo. Junto con el throughput y el tamaño de bloque determina el rendimiento real de un volumen: 16.000 IOPS de 4 KB no equivalen a 16.000 de 256 KB.
AMI	Imagen de máquina que define el estado inicial de una instancia. Construirla de forma reproducible es lo que permite sustituir instancias en lugar de repararlas.
instancia efímera	Instancia que puede reclamarse con dos minutos de preaviso a cambio de un descuento de hasta el 90 %. Correcta para trabajo interrumpible; nunca para el camino crítico de una petición.

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

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 nombre de la instancia es una especificación

m7g.2xlarge no es un código arbitrario: cada carácter dice algo, y leerlo evita elegir a ciegas.

```
m      familia: propósito general (c=cómputo, r=memoria, i=E/S, t=ráfaga)
7      generación: cuanto mayor, mejor relación precio-rendimiento
g      procesador: g=Graviton (ARM), i=Intel, a=AMD; sin letra = Intel
.2xlarge  tamaño: 8 vCPU y 32 GB
```

Sufijos adicionales que cambian el precio y el comportamiento:

Sufijo	Significa
d	Almacenamiento NVMe local, efímero
n	Ancho de banda de red mejorado
e	Memoria o almacenamiento ampliados
flex	Rendimiento sostenido reducido, más barato

La generación es la palanca más rentable y la más ignorada: pasar de la 5 a la 7 suele dar mejor rendimiento por menos dinero, y el cambio no requiere tocar la aplicación salvo por la arquitectura del procesador.

Sobre Graviton: es ARM, así que exige binarios para arm64. Para lenguajes interpretados o con máquina virtual —Python, Java, Node— el cambio suele ser transparente; para binarios compilados o imágenes de contenedor hay que reconstruir. La relación precio-rendimiento típicamente mejora entre un 20 % y un 40 %, así que la reconstrucción se amortiza rápido en flotas grandes.

El tamaño escala linealmente dentro de la familia: 2xlarge es exactamente el doble de xlarge en vCPU, memoria, red y precio. Eso hace que la aritmética de dimensionado sea directa.

2. Créditos de CPU: el desplome que no aparece en las métricas

Las instancias de la familia t no ofrecen su vCPU completa de forma continua. Tienen una línea base —un porcentaje del rendimiento nominal— y acumulan créditos mientras consumen por debajo de ella.

```
t3.medium: línea base 20 % por vCPU
  ociosa al 5 %   → acumula créditos
  al 80 % de CPU  → gasta créditos
  saldo a cero    → el rendimiento se limita al 20 %
```

El síntoma es característico y desconcierta: la aplicación se vuelve cuatro o cinco veces más lenta sin ningún cambio en el código, la carga ni la configuración. Ninguna métrica del sistema operativo lo explica, porque desde dentro la CPU parece ocupada y disponible.

El diagnóstico exige una métrica del proveedor, no del sistema:

```
$ aws cloudwatch get-metric-statistics --namespace AWS/EC2 \
    --metric-name CPUCreditBalance --dimensions Name=InstanceId,Value=i-0a1b2c \
    --start-time 2026-08-01T00:00:00Z --end-time 2026-08-01T12:00:00Z \
    --period 300 --statistics Minimum --query 'Datapoints[?Minimum<`10`]|length(@)'
17
```

Diecisiete intervalos con el saldo por debajo de 10 créditos: la instancia lleva horas limitada.

Dos salidas, con consecuencias distintas:

```
modo ilimitado   permite superar la línea base pagando el exceso.
                 Correcto para picos ocasionales; ruinoso si la carga es
                 sostenida, porque el sobrecargo puede superar el precio
                 de una instancia normal.
cambiar familia  si la utilización media supera la línea base de forma
                 sostenida, la familia t es la elección equivocada.
```

La regla: la familia t es para cargas con media baja y picos cortos. Si la media supera la línea base, el descuento desaparece y el riesgo permanece.

3. Volúmenes: IOPS, throughput y tamaño de bloque

Elegir volumen por tamaño es el error más común. Lo que decide el rendimiento son tres números y su interacción:

Tipo	IOPS máx	Throughput máx	Precio relativo	Para
gp3	16.000	1.000 MB/s	1,0	Propósito general; IOPS y throughput independientes del tamaño
gp2	3 por GB, tope 16.000	250 MB/s	1,25	Generación anterior: el rendimiento depende del tamaño
io2	256.000	4.000 MB/s	3-5	Bases de datos exigentes, con durabilidad mayor
st1	500	500 MB/s	0,3	Lectura secuencial grande: logs, big data

La diferencia entre gp2 y gp3 es económicamente relevante: en gp2 hay que sobredimensionar el tamaño para obtener IOPS. Para 6.000 IOPS hacen falta 2.000 GB aunque solo se usen 200. En gp3 se configuran por separado:

```
gp2 para 6.000 IOPS:  2.000 GB × 0,10 USD/GB          = 200,00 USD/mes
gp3 equivalente:       200 GB × 0,08 + 3.000 IOPS extra × 0,005
                    =  16,00 + 15,00                  =  31,00 USD/mes
```

Un factor de 6,5 por la misma capacidad efectiva. Migrar de gp2 a gp3 es de las optimizaciones de mayor retorno y menor riesgo que existen.

Y el detalle que descoloca al medir: una operación de E/S se cuenta hasta 256 KB. Una lectura de 1 MB consume 4 IOPS, no 1. Por eso una carga secuencial agota antes el throughput que las IOPS, y una aleatoria de bloques pequeños agota antes las IOPS:

```
16.000 IOPS × 4 KB   =   64 MB/s  ← limita el IOPS
16.000 IOPS × 256 KB = 4.096 MB/s ← limita el throughput (tope 1.000)
```

Hay un segundo límite que se olvida: la instancia también tiene un tope de ancho de banda hacia EBS, independiente del volumen. Un gp3 de 1.000 MB/s conectado a una instancia con 600 MB/s de ancho de banda entrega 600.

4. Modelos de compra y el umbral que los separa

Cuatro formas de pagar el mismo cómputo:

Modelo	Descuento	Compromiso	Riesgo
Bajo demanda	0 %	Ninguno	Ninguno
Savings Plan 1 año	28 %	Gasto por hora, no instancia concreta	Sobredimensionar
Savings Plan 3 años	45 %	Ídem	Cambio tecnológico

Modelo	Descuento	Compromiso	Riesgo
Efímeras	70-90 %	Ninguno	Reclamación con 2 min de aviso

El umbral de utilización a partir del cual compensa comprometerse sale de igualar costes, como en la clase 015:

```
u × 730 h × P = 730 h × 0,72 P   →   u = 0,72
```

Con más del 72 % de uso sostenido, el plan de un año sale a cuenta. Y hay un matiz que lo hace más atractivo de lo que parece: los Savings Plans de cómputo comprometen gasto por hora, no un tipo de instancia. Se puede cambiar de familia, de tamaño y de región sin perder el descuento, lo que elimina buena parte del riesgo de sobredimensionar.

Las instancias efímeras merecen su propio criterio. El descuento es enorme y el riesgo concreto: dos minutos de preaviso. Son correctas para trabajo por lotes reanudable, transcodificación, entrenamiento con puntos de control y entornos de prueba. No lo son para nada que sostenga una petición de usuario, salvo que la arquitectura absorba la pérdida sin degradación visible.

La estrategia habitual en una flota:

```
base estable (60 % de la capacidad)  → Savings Plan
variación previsible (25 %)          → bajo demanda
picos y lotes (15 %)                 → efímeras
```

5. El estado local es el enemigo de la elasticidad

Una instancia que guarda estado en su disco no se puede sustituir: hay que repararla. Y reparar es lo contrario de operar en cloud.

La distinción práctica, con lo que hay que hacer con cada cosa:

```
logs        → fuera del host, a un recolector (clase 007)
sesiones    → almacén compartido, no memoria del proceso
ficheros    → almacenamiento de objetos o sistema de ficheros compartido
caché       → externa; si es local, debe poder perderse sin consecuencias
datos       → base de datos gestionada
configuración → parámetros o secretos, inyectados al arrancar
```

Si todo eso está fuera, una instancia es desechable: se puede terminar y crear otra sin ceremonia. Esa propiedad es la que hacen posibles el autoescalado, los despliegues sin interrupción y el parcheo por sustitución.

Y exige una AMI reproducible. Construirla a mano —arrancar, instalar, guardar imagen— produce una imagen que nadie sabe recrear:

```
# Fragmento de plantilla declarativa
source "amazon-ebs" "cloudshop" {
  source_ami_filter {
    filters = { name = "al2023-ami-*-arm64" }
    owners  = ["amazon"]
    most_recent = true
  }
  instance_type = "t4g.small"
  ami_name      = "cloudshop-{{timestamp}}"
}
```

Dos propiedades que hay que exigir a la imagen:

1. Reproducible: la misma plantilla produce una imagen funcionalmente equivalente.
2. Fechada e inmutable: nunca se sobrescribe una AMI; se crea otra y se cambia la referencia. Es la misma lógica de la clase 003 sobre etiquetas mutables.

Y una advertencia de seguridad: una AMI compartida por error es pública para siempre a efectos prácticos. Antes de compartir hay que comprobar que no contiene claves, historial de shell ni datos residuales.

Ejemplo trabajado

El servicio de catálogo de CloudShop se degrada cada día entre las 11:00 y las 14:00. No hay despliegues ni aumento de tráfico en esa franja.

Las métricas de la aplicación no explican nada:

```
peticiones/s        estables en 340
tasa de error       0,02 %, sin cambio
CPU del sistema     94 % durante la degradación
latencia p95        88 ms → 410 ms
```

La CPU al 94 % sugiere saturación, pero el tráfico no subió. Se mira la métrica del proveedor:

```
$ aws cloudwatch get-metric-statistics --namespace AWS/EC2 \
    --metric-name CPUCreditBalance --dimensions Name=InstanceId,Value=i-0a1b2c \
    --start-time 2026-08-01T08:00:00Z --end-time 2026-08-01T15:00:00Z \
    --period 3600 --statistics Average --query 'Datapoints[].[Timestamp,Average]' --output text | sort
2026-08-01T08:00:00Z    142.0
2026-08-01T10:00:00Z     38.0
2026-08-01T11:00:00Z      0.0        ← agotados
2026-08-01T14:00:00Z      0.0
```

Créditos agotados a las 11:00. Las instancias son t3.large con línea base del 30 %; al agotar el saldo, el rendimiento se limita a esa fracción:

```
rendimiento nominal      2 vCPU
línea base 30 %          0,6 vCPU efectivas
factor de degradación    3,3×
latencia observada       88 × 3,3 = 290 ms teórico; medido 410 con encolamiento
```

Se comprueba la utilización media real para elegir la salida:

```
utilización media diaria    47 %
línea base de t3.large      30 %
```

La media supera la línea base, así que la familia t es la elección equivocada: el modo ilimitado cobraría el exceso todo el día. Se dimensiona una alternativa por la ley de Little:

```
λ = 340 peticiones/s, W = 0,088 s → L = 30 concurrentes
a ρ = 0,70 → 43 concurrentes → con 16 por instancia, 3 instancias
```

Comparación de tres opciones para 3 instancias:

```
3 × t3.large (actual)              6    182,50 USD   no: se limita
3 × t3.large modo ilimitado        6    182,50 + ~95 sobrecargo = 277,50
3 × m7g.large (Graviton)           6    139,00 USD   sí, sostenido
```

m7g.large cuesta menos que la opción actual degradada y sostiene el rendimiento. Requiere reconstruir la imagen para arm64; la aplicación es Python y la reconstrucción resultó transparente salvo por una dependencia con extensión nativa.

Se aprovecha para revisar los volúmenes:

```
$ aws ec2 describe-volumes --filters Name=attachment.instance-id,Values=i-0a1b2c \
    --query 'Volumes[].[VolumeType,Size,Iops]' --output text
gp2     500     1500
```

```
gp2 de 500 GB: 1.500 IOPS, uso real 92 GB → se pagaba tamaño para obtener IOPS
gp3 de 120 GB con 3.000 IOPS configurados:
  gp2: 500 × 0,10                       = 50,00 USD/mes
  gp3: 120 × 0,08                       =  9,60 USD/mes
  (3.000 IOPS entran en la base gratuita)
  ahorro por instancia                   = 40,40 USD/mes
```

Resultado combinado:

```
instancias              3 × t3.large   3 × m7g.large
cómputo                   182,50 USD     139,00 USD
almacenamiento            150,00 USD      28,80 USD
total                     332,50 USD     167,80 USD   (−50 %)
latencia p95 11:00-14:00      410 ms          91 ms
degradación diaria          3 horas        ninguna
```

El diagnóstico no estaba en la aplicación ni en el sistema operativo: estaba en una métrica que solo el proveedor publica. Y la corrección salió más barata que el problema.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-02-aws-core-platform/028-ec2-ami-ebs-y-seleccion-de-capacidad/lab.py
```

El laboratorio selecciona el motor de práctica compute 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 instancia-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 una selección de capacidad justificada y observable. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye instancia-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
El rendimiento se desploma a diario sin cambios de código ni de carga	Agotamiento de créditos de CPU en una instancia ráfaga	Vigila CPUCreditBalance; si la media supera la línea base, cambia de familia en vez de activar modo ilimitado.
Se paga un volumen enorme del que se usa una fracción	En gp2 las IOPS dependen del tamaño, así que hay que sobredimensionar	Migra a gp3 y configura IOPS y throughput por separado; el ahorro suele superar el 60 %.
Un volumen con 1.000 MB/s entrega bastante menos	La instancia tiene su propio tope de ancho de banda hacia EBS	Comprueba el límite de la instancia además del volumen; el menor de los dos manda.
Una instancia no se puede sustituir sin perder datos	Guarda estado local: sesiones, ficheros o logs	Externaliza todo el estado; una instancia que no es desechable impide el autoescalado y el parcheo por sustitución.
Un trabajo en instancias efímeras pierde horas de progreso	Se usó capacidad interrumpible sin puntos de control	Reserva las efímeras para trabajo reanudable con checkpoints; el preaviso es de dos minutos.

■ 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. Descompón c7gn.4xlarge: ¿qué dice cada parte del nombre?
2. Una instancia t3 tiene utilización media del 47 % y línea base del 30 %. ¿Conviene modo ilimitado? Justifica.
3. ¿Cuántas IOPS consume una lectura secuencial de 1 MB, y por qué eso cambia qué límite alcanzas antes?
4. ¿Por qué gp3 con 200 GB puede rendir más que gp2 con 2.000 GB y costar seis veces menos?
5. ¿A partir de qué utilización sostenida compensa un Savings Plan de un año, y qué riesgo elimina que comprometa gasto y no tipo de instancia?

Referencias

• AWS (2024). Amazon EC2 instance types — nomenclatura, familias y capacidades adicionales. <https://docs.aws.amazon.com/ec2/latest/instancetypes/instance-types.html>

• AWS (2024). Burstable performance instances — línea base, créditos y modo ilimitado. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances.html>

• AWS (2024). Amazon EBS volume types — IOPS, throughput y tamaño de operación de E/S. <https://docs.aws.amazon.com/ebs/latest/userguide/ebs-volume-types.html>

• AWS (2024). Savings Plans — compromiso por gasto horario frente a instancias reservadas. <https://docs.aws.amazon.com/savingsplans/latest/userguide/what-is-savings-plans.html>

• AWS (2024). Spot Instances: interruption notices — preaviso de dos minutos y patrones de uso seguro. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-interruptions.html>

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

Descargar: Parte 02 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 027 · VPC, subredes, rutas, NAT, endpoints y seguridad	Parte 02 · Programa	029 · Elastic Load Balancing y Auto Scaling →

Evaluación — 028 EC2, AMI, EBS y selección de capacidad

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 instancia-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.

029 — Elastic Load Balancing y Auto Scaling

Parte: 02 — AWS: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: reliability · Estado: EXECUTABLECORE

Propósito

Construir una capa elástica que reparta tráfico y ajuste capacidad sin oscilar ni tumbar el servicio al desplegar. Aquí se aplican a AWS los cálculos de la clase 016 —histéresis, margen de arranque, rechazo de carga— y se añade lo que ninguna fórmula anticipa: que el drenado de conexiones y el tipo de comprobación de estado deciden si un despliegue pierde peticiones.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir entre balanceador de aplicación y de red según protocolo, latencia y necesidad de terminar TLS.
2. Configurar comprobaciones de estado que distingan una instancia arrancando de una averiada.
3. Dimensionar un grupo de autoescalado con umbrales, enfriamiento y periodo de gracia coherentes.
4. Evitar la pérdida de peticiones al reducir capacidad, mediante drenado de conexiones.
5. Diagnosticar un 502 y un 504 del balanceador sabiendo qué componente señala cada uno.

Conceptos centrales

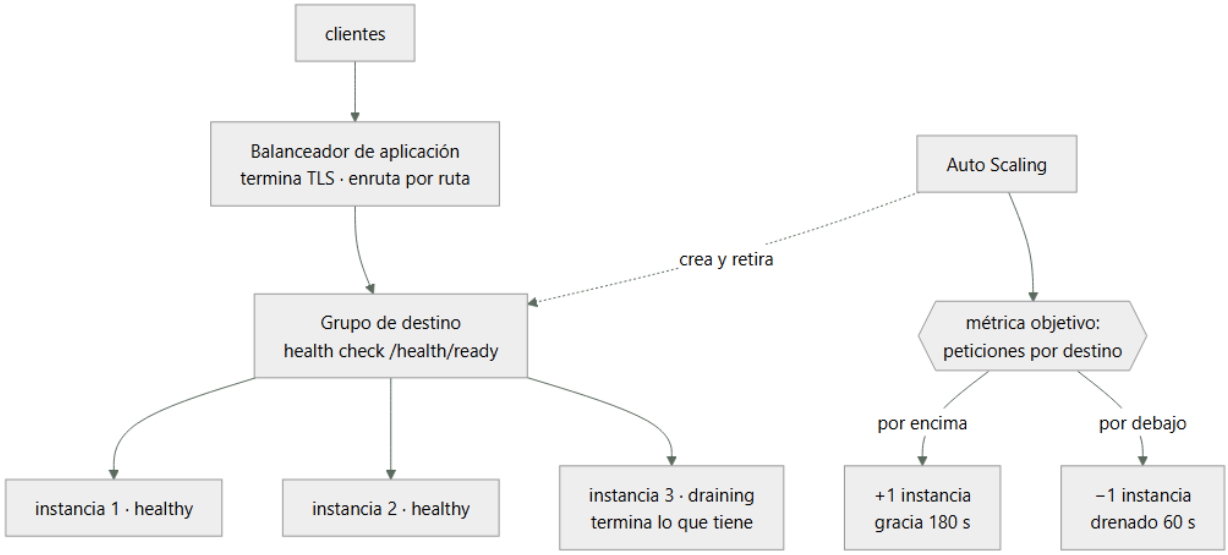
Concepto	Comprensión verificable
grupo de destino	Conjunto de destinos que reciben tráfico, con su propia comprobación de estado y política de enrutamiento. Es la unidad sobre la que actúan el balanceador y el autoescalado.
comprobación de estado	Sondeo periódico que decide si un destino recibe tráfico. Debe reflejar readiness —¿puede servir?— y no liveness, por lo visto en la clase 012.
drenado de conexiones	Periodo durante el cual un destino que se retira deja de recibir peticiones nuevas pero termina las que tiene en curso. Sin él, reducir capacidad corta transacciones a medias.
periodo de gracia	Tiempo que el autoescalado espera antes de evaluar la salud de una instancia recién creada. Corto de más, mata instancias que aún arrancan y entra en bucle.
seguimiento de objetivo	Política de escalado que ajusta capacidad para mantener una métrica en un valor deseado. Gestiona la histéresis automáticamente, a diferencia de las políticas por umbral.

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

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

■ Flujo de razonamiento



Desarrollo

1. Cuatro balanceadores, tres decisiones

	Aplicación (ALB)	Red (NLB)	Gateway (GWLB)
Capa	7 (HTTP)	4 (TCP/UDP)	3 (GENEVE)
Enruta por	Ruta, host, cabecera, método	Puerto	Todo el paquete
Termina TLS	Sí	Sí, opcionalmente	No
IP fija	No	Sí, por zona	No
Latencia añadida	5-10 ms	1 ms	Depende
Preserva IP de origen	En cabecera X-Forwarded-For	En el paquete	Sí

Tres criterios deciden:

Protocolo. Si es HTTP y hace falta enrutar por ruta o host, ALB. Si es TCP, UDP o un protocolo propio, NLB.

IP fija. El NLB tiene una IP estática por zona; el ALB resuelve por DNS y sus IP cambian. Esto importa cuando un cliente externo debe incluir tu servicio en una lista de permitidos: con ALB no hay IP estable que dar.

IP de origen. El NLB preserva la IP del cliente en el propio paquete; el ALB la pone en X-Forwarded-For. Una aplicación que aplica límites de tasa por IP y lee la IP de la conexión verá la del balanceador —y limitará a todos como si fueran uno— salvo que lea la cabecera. Es un fallo que solo aparece bajo ataque.

Y un detalle de coste: ambos cobran por hora y por unidad de capacidad consumida, una métrica compuesta de conexiones nuevas, conexiones activas, ancho de banda y reglas evaluadas. La dimensión que domine determina la factura, y suele ser conexiones nuevas por segundo en servicios que no reutilizan conexión —lo de la clase 005—.

2. La comprobación de estado debe reflejar readiness

Un balanceador retira del reparto lo que falla la comprobación. Por tanto la comprobación debe responder «¿puede servir ahora?» y no «¿está el proceso vivo?».

```
camino          /health/ready       ← no /  ni /health/live
intervalo       10 s
plazo           5 s                 ← menor que el intervalo
umbral sano     2 comprobaciones    ← vuelve rápido
umbral insano   3 comprobaciones    ← no se va por un fallo aislado
códigos válidos 200
```

La asimetría entre los dos umbrales es deliberada: entrar rápido y salir despacio. Un fallo aislado no debe retirar un destino sano, pero un destino que se recupera debe volver a recibir tráfico cuanto antes.

El tiempo de detección se calcula:

```
detección de fallo = intervalo × umbral insano = 10 × 3 = 30 s
```

Treinta segundos durante los cuales una fracción del tráfico va a un destino averiado. Bajarlo tiene coste: intervalos muy cortos generan carga de sondeo y falsos positivos bajo saturación. Con nueve destinos y 10 segundos son 54 sondeos por minuto, despreciable; con 200 destinos y 5 segundos son 2.400, que ya se nota.

El error que convierte una degradación en caída total, ya visto en la clase 012: si /health/ready comprueba la base de datos y esta cae, los nueve destinos fallan a la vez y el balanceador se queda sin ninguno sano. AWS mitiga esto con el modo fail open —si ningún destino está sano, envía tráfico a todos—, pero apoyarse en eso es apoyarse en un comportamiento de emergencia. Lo correcto es que la comprobación distinga dependencias esenciales de opcionales.

3. Autoescalado: los cuatro parámetros y su interacción

El seguimiento de objetivo es preferible a los umbrales manuales porque gestiona la histéresis solo: se declara el valor deseado de una métrica y el servicio calcula los ajustes.

```
$ aws autoscaling put-scaling-policy --auto-scaling-group-name cloudshop-asg \
    --policy-name seguimiento-peticiones --policy-type TargetTrackingScaling \
    --target-tracking-configuration '{
      "PredefinedMetricSpecification": {
        "PredefinedMetricType": "ALBRequestCountPerTarget",
        "ResourceLabel": "app/cloudshop-alb/50dc.../targetgroup/cloudshop-tg/6d0e..."
      },
      "TargetValue": 420.0,
      "ScaleInCooldown": 300,
      "ScaleOutCooldown": 60
    }'
```

Cuatro decisiones en ese documento:

La métrica. ALBRequestCountPerTarget es mejor que la CPU para servicios web: mide demanda directamente y no depende de que el trabajo sea de cómputo. Una carga limitada por E/S nunca dispara un escalado basado en CPU aunque esté saturada.

El valor objetivo. Se deriva de la capacidad medida por destino y del objetivo de utilización de la clase 016:

```
capacidad por destino a saturación   600 peticiones/s
objetivo de utilización              0,70
valor objetivo                       420 peticiones/s
```

Los enfriamientos asimétricos. Subir rápido (60 s) y bajar despacio (300 s). Retirar capacidad demasiado pronto es lo que produce oscilación, y el coste de sobrar una instancia unos minutos es mucho menor que el de faltar.

El periodo de gracia, que se configura en el grupo:

```
periodo de gracia > tiempo de arranque hasta servir tráfico
medido: 118 s  →  se fija en 180 s
```

Si la gracia es menor que el arranque, el grupo marca insanas a las instancias nuevas, las termina y crea otras: un bucle que consume dinero y nunca converge.

4. Drenado: por qué reducir capacidad pierde peticiones

Cuando el autoescalado retira una instancia, el balanceador deja de enviarle peticiones nuevas pero puede tener transacciones en curso. Sin drenado, esas se cortan.

```
drenado 0 s    la instancia se termina de inmediato
               → peticiones en vuelo cortadas → errores para el usuario
drenado 60 s   deja de recibir nuevas, termina las que tiene
               → cero errores si ninguna dura más de 60 s
```

El valor debe ser mayor que la petición más larga:

```
$ aws elbv2 modify-target-group-attributes --target-group-arn arn:...:targetgroup/... \
    --attributes Key=deregistration_delay.timeout_seconds,Value=60
```

Y hay una segunda pieza que casi siempre falta: la aplicación debe cooperar. El balanceador deja de enviar tráfico, pero el orquestador envía SIGTERM a la instancia. Si el proceso no lo atrapa y cierra de inmediato, el drenado del balanceador no sirve de nada —lo de la clase 002—:

```
def apagar(signum, frame):
```

```
    servidor.stop_accepting()      # no aceptar conexiones nuevas
    servidor.wait_for_inflight(55) # terminar las en curso, con margen
    sys.exit(0)

signal.signal(signal.SIGTERM, apagar)
```

Hay un orden que importa y que se equivoca a menudo: el proceso debe primero fallar la comprobación de readiness y solo después dejar de aceptar conexiones. Si deja de aceptar antes de que el balanceador se entere, hay una ventana de segundos en la que el balanceador sigue enviando tráfico a un puerto cerrado, y el usuario recibe un 502.

La secuencia correcta, con sus tiempos:

```
t+0   llega SIGTERM
t+0   /health/ready empieza a devolver 503
t+30  el balanceador ha detectado el fallo (intervalo × umbral) y deja de enviar
t+30  el proceso deja de aceptar conexiones nuevas
t+85  terminan las peticiones en curso
t+85  el proceso sale con 0
```

5. 502 y 504 del balanceador señalan cosas distintas

Los errores del balanceador se distinguen de los de la aplicación en las métricas y en los logs, y confundirlos alarga los incidentes:

Código	Métrica de CloudWatch	Qué ocurrió	Dónde mirar
502	HTTPCodeELB502Count	El destino cerró la conexión o devolvió algo inválido	Logs del destino
503	HTTPCodeELB503Count	No hay destinos sanos	Comprobación de estado
504	HTTPCodeELB504Count	El destino no respondió dentro del plazo	Latencia y saturación del destino

La causa más frecuente de 502 sin ningún error en la aplicación es una discordancia de tiempos de espera:

```
tiempo de espera del balanceador   60 s
tiempo de espera keep-alive del destino  5 s     ← MENOR
```

El destino cierra la conexión inactiva a los 5 segundos; el balanceador, que la creía viva, envía una petición por ella y recibe un cierre. El resultado es un 502 intermitente, con más frecuencia cuanto menor sea el tráfico —porque las conexiones se quedan inactivas—.

La regla: el keep-alive del destino debe ser mayor que el tiempo de espera de inactividad del balanceador. Con 60 s en el balanceador, 75 s en el destino.

Y los logs del balanceador dan el detalle que las métricas no:

```
request_processing_time  target_processing_time  response_processing_time
     0.001                       -1                        -1
```

Un -1 en targetprocessingtime significa que el destino nunca respondió: el problema no está en la aplicación sino en la conexión hacia ella. Distinguir eso de un 59.998 —el destino tardó y agotó el plazo— cambia por completo dónde hay que buscar.

Ejemplo trabajado

Cada despliegue de CloudShop produce entre 200 y 400 errores 502, y el autoescalado oscila creando y destruyendo instancias. Se atacan los dos problemas.

Problema 1 — errores en cada despliegue.

```
$ aws elbv2 describe-target-group-attributes --target-group-arn arn:...:targetgroup/cloudshop-tg \
    --query 'Attributes[?Key==`deregistration_delay.timeout_seconds`].Value' --output text
0
```

Drenado a cero: las instancias se terminan con peticiones en vuelo. Se mide la duración de las peticiones para elegir el valor:

```
p50   38 ms
```

```
p95   91 ms
p99  185 ms
máx  4,2 s   (exportación de informes)
```

Se fija en 60 s, con margen amplio sobre los 4,2 s del peor caso.

Pero al repetir el despliegue siguen apareciendo 502, solo que menos:

```
antes del cambio   340 errores
después            96 errores
```

La aplicación no atrapaba SIGTERM. Se añade el manejador y se comprueba el orden:

```
versión inicial:  SIGTERM → cierra el socket inmediatamente
                  el balanceador tarda 30 s en enterarse → 502 durante 30 s
versión correcta: SIGTERM → /health/ready devuelve 503
                            espera 35 s (intervalo 10 × umbral 3 + margen)
                            deja de aceptar y drena lo en curso

tras el arreglo completo:  0 errores en 5 despliegues consecutivos
```

Problema 2 — oscilación del autoescalado.

```
$ aws autoscaling describe-scaling-activities --auto-scaling-group-name cloudshop-asg \
    --max-items 8 --query 'Activities[].[StartTime,Description]' --output text
2026-08-01T11:42  Terminating EC2 instance: i-0f3a
2026-08-01T11:39  Launching a new EC2 instance: i-0f3a
2026-08-01T11:36  Terminating EC2 instance: i-0c8b
2026-08-01T11:33  Launching a new EC2 instance: i-0c8b
```

Crea y destruye cada 3 minutos. La configuración:

```
política       umbral simple: sube al 70 % de CPU, baja al 65 %
enfriamiento   120 s
gracia         60 s
tiempo de arranque medido  118 s
```

Dos fallos independientes:

```
1. Histéresis insuficiente: con 8 instancias al 71 %, añadir una baja
   la utilización a 63 % → dispara la reducción → vuelve a 71 %.
2. Gracia (60 s) < arranque (118 s): las instancias nuevas se marcan
   insanas antes de poder servir y se terminan.
```

Se sustituye por seguimiento de objetivo sobre peticiones por destino:

```
capacidad medida por destino a saturación   600 peticiones/s
objetivo 0,70                               420 peticiones/s
enfriamiento de subida                       60 s
enfriamiento de bajada                      300 s
periodo de gracia                           180 s   (> 118 medidos)
```

Se elige peticiones por destino y no CPU porque el servicio pasa el 60 % del tiempo esperando a la base de datos: la CPU nunca refleja la saturación real.

Resultado en 7 días:

```
errores 502 por despliegue             340          0
actividades de escalado/día             48          6
instancias en régimen              6-11 osc.      7-9
p95 en hora punta                   410 ms       94 ms
coste mensual de cómputo            412 USD     338 USD
```

El coste bajó un 18 % al dejar de oscilar: cada instancia creada y destruida a los 3 minutos se facturaba igual y no llegaba a servir tráfico útil.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-02-aws-core-platform/029-elastic-load-balancing-y-auto-scaling/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 servicio-elastico-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 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 servicio-elastico-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
Cada despliegue produce cientos de 502	Drenado a cero: las instancias se terminan con peticiones en vuelo	Fija el drenado por encima de la petición más larga y atrapa SIGTERM en la aplicación.
Se configura el drenado y siguen apareciendo 502, aunque menos	La aplicación cierra el socket antes de que el balanceador detecte el cambio de estado	Al recibir SIGTERM, falla readiness primero y espera intervalo × umbral antes de dejar de aceptar.
El autoescalado crea y destruye instancias cada pocos minutos	Umbrales de subida y bajada demasiado próximos, o gracia menor que el arranque	Usa seguimiento de objetivo y fija la gracia por encima del arranque medido.
El servicio se satura y el autoescalado no reacciona	La métrica es la CPU y la carga está limitada por E/S	Escala por peticiones por destino o por una métrica que refleje la demanda real.
Aparecen 502 intermitentes, más frecuentes con poco tráfico	El keep-alive del destino es menor que el tiempo de espera del balanceador	Configura el keep-alive del destino por encima del plazo de inactividad del balanceador.

■ 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 dos casos concretos necesitas un balanceador de red en vez de uno de aplicación?
2. Con intervalo de 10 s y umbral insano de 3, ¿cuánto tarda en detectarse un fallo y qué ocurre entretanto?
3. ¿Por qué los umbrales sano e insano deben ser asimétricos, y en qué dirección?
4. Describe la secuencia correcta desde que llega SIGTERM hasta que el proceso sale sin perder peticiones.

5. Un log del balanceador muestra targetprocessingtime = -1. ¿Qué significa y dónde buscas?

Referencias

- AWS (2024). Application Load Balancer: target groups and health checks — parámetros y semántica. <https://docs.aws.amazon.com/elasticloadbalancing/latest/application/target-group-health-checks.html>
- AWS (2024). Deregistration delay — drenado de conexiones y su interacción con el ciclo de vida. <https://docs.aws.amazon.com/elasticloadbalancing/latest/application/edit-target-group-attributes.html>
- AWS (2024). Target tracking scaling policies — métricas predefinidas y enfriamientos. <https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-target-tracking.html>
- AWS (2024). Troubleshoot your Application Load Balancer — causas de 502, 503 y 504 y campos del log de acceso. <https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-troubleshooting.html>
- Beyer, B. et al., eds. (2016). Site Reliability Engineering, cap. 20 — reparto de carga y comprobaciones de salud. <https://sre.google/sre-book/load-balancing-datacenter/>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 02 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 028 · EC2, AMI, EBS y selección de capacidad	Parte 02 · Programa	030 · S3: objetos, versionado, lifecycle y replicación →

Evaluación — 029 Elastic Load Balancing y Auto Scaling

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 servicio-elastico-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.

030 — S3: objetos, versionado, lifecycle y replicación

Parte: 02 — AWS: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: storage · Estado: EXECUTABLECORE

Propósito

Operar almacenamiento de objetos entendiendo qué garantiza y qué no. Los once nueves de durabilidad no protegen de un borrado autorizado —lo estableció la clase 010— y el modelo de acceso de S3 es el origen de una parte desproporcionada de las brechas públicas atribuidas a la nube. Aquí se convierte todo eso en configuración verificable.

Resultados de aprendizaje

Al finalizar podrás:

1. Explicar por qué el versionado y el bloqueo de objetos son controles distintos y qué protege cada uno.
2. Diseñar una política de ciclo de vida que reduzca coste sin romper la recuperación ni el cumplimiento.
3. Calcular si una transición a una clase más fría ahorra, contando el mínimo de días y el coste de recuperación.
4. Bloquear el acceso público en todos los niveles y verificarlo con prueba negativa.
5. Distinguir replicación de copia de seguridad, y por qué la primera propaga los borrados.

Conceptos centrales

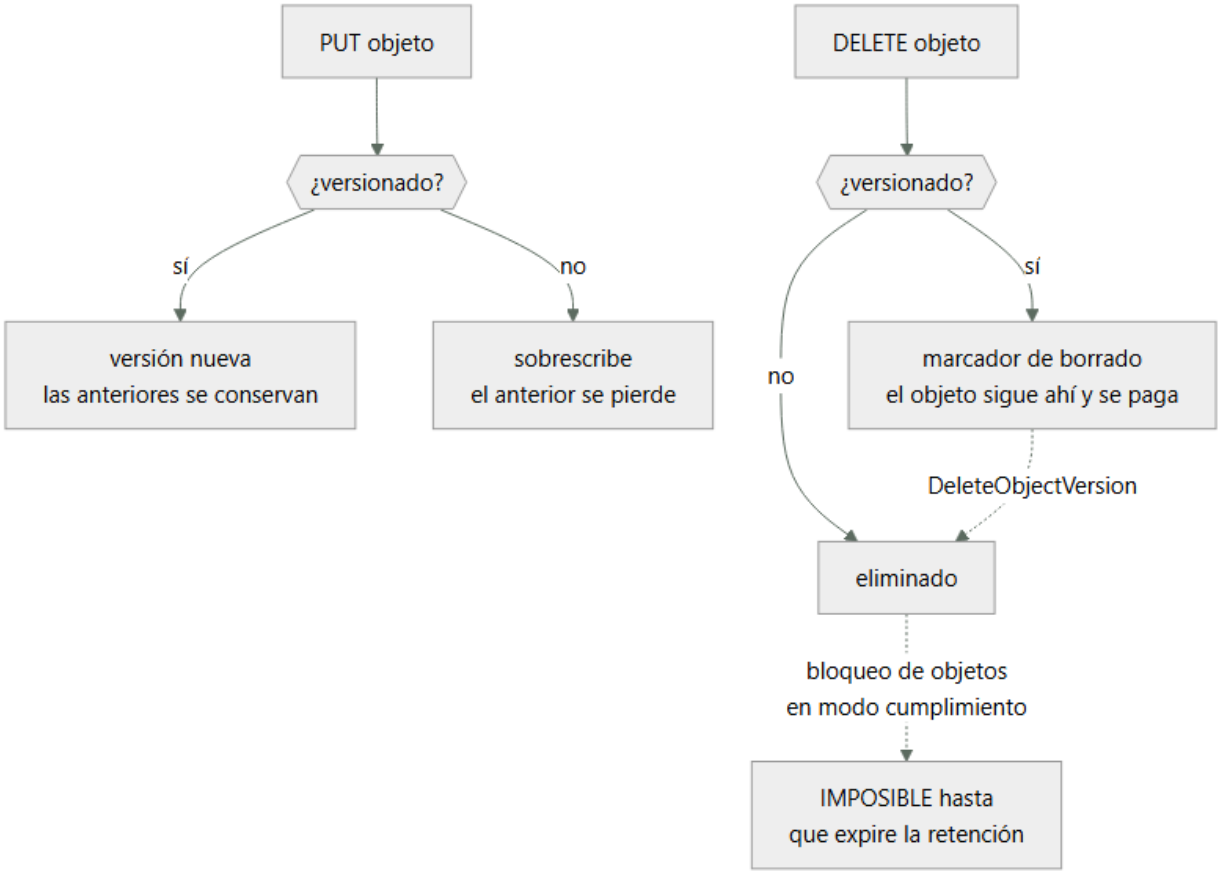
Concepto	Comprensión verificable
versionado	Conservar todas las versiones de un objeto, incluidas las borradas mediante un marcador. Protege del error humano, pero no de quien tenga permiso para borrar versiones.
bloqueo de objetos	Retención inmutable que impide borrar o sobrescribir durante un periodo. En modo cumplimiento ni siquiera la cuenta raíz puede saltárselo, que es lo que lo hace útil frente a ransomware.
clase de almacenamiento	Perfil de coste y acceso: estándar, infrecuente, archivo. Las frías cobran menos por GB y añaden coste de recuperación, mínimos de duración y a veces latencia de horas.
marcador de borrado	Versión especial que oculta un objeto sin eliminarlo. Explica por qué un bucket versionado sigue creciendo y costando después de «vaciarlo».
punto de acceso	Endpoint con su propia política, asociado a un bucket. Permite dar permisos acotados por aplicación sin que la política del bucket crezca sin control.

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

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. Versionado y bloqueo protegen de cosas distintas

Se confunden y cubren riesgos que no se solapan:

Riesgo	Versionado	Bloqueo de objetos
Sobrescritura accidental	✓	✓
Borrado accidental	✓	✓
Borrado deliberado con permisos	✗	✓
Ransomware con credenciales válidas	✗	✓
Corrupción por fallo de disco	✓ (11 nueves)	✓

Las dos filas centrales son la diferencia. Con versionado, quien tenga s3:DeleteObjectVersion borra todas las versiones y el objeto desaparece de verdad. El bloqueo en modo cumplimiento lo impide hasta que expire la retención, y no admite excepción para nadie —incluida la cuenta raíz—.

```
$ aws s3api put-object-lock-configuration --bucket cloudshop-facturas \
    --object-lock-configuration '{"ObjectLockEnabled":"Enabled",
      "Rule":{"DefaultRetention":{"Mode":"COMPLIANCE","Days":2555}}}'
```

Los dos modos no son intercambiables:

```
GOVERNANCE  quien tenga s3:BypassGovernanceRetention puede saltárselo.
            Útil para protección operativa con escape controlado.
COMPLIANCE  nadie puede. Ni el administrador, ni la raíz, ni AWS.
            Es el único que protege frente a un atacante con permisos.
```

Y una advertencia que hay que entender antes de activarlo: el modo cumplimiento no se puede reducir ni desactivar. Un objeto con 7 años de retención ocupará y costará 7 años. Es exactamente lo que se quiere para evidencia legal y una trampa cara si se aplica por defecto a todo un bucket de datos operativos.

El bloqueo exige versionado activado y solo puede habilitarse al crear el bucket, o mediante una solicitud a soporte. Otra razón para decidirlo al principio.

2. El marcador de borrado explica la factura que no baja

En un bucket versionado, DELETE no borra: crea un marcador que oculta el objeto. Las versiones anteriores siguen ahí y siguen facturándose.

```
$ aws s3 rm s3://cloudshop-datos/informe.csv
delete: s3://cloudshop-datos/informe.csv
$ aws s3 ls s3://cloudshop-datos/informe.csv
# vacío: parece borrado
$ aws s3api list-object-versions --bucket cloudshop-datos --prefix informe.csv \
    --query '[Versions[].[Size,VersionId],DeleteMarkers[].VersionId]'
[[[4823901, "3HL4kqt..."]], ["Ktr5nQ..."]]
```

El objeto de 4,8 MB sigue existiendo y pagándose. En un bucket con años de operación, las versiones no vigentes pueden ser la mayor parte del almacenamiento:

```
$ aws s3api list-object-versions --bucket cloudshop-datos --output json \
  | jq '{vigentes: ([.Versions[]|select(.IsLatest)|.Size]|add),
         antiguas: ([.Versions[]|select(.IsLatest|not)|.Size]|add)}'
{"vigentes": 412000000000, "antiguas": 1840000000000}
```

1,84 TB de versiones antiguas frente a 412 GB vigentes: el 82 % del coste es historial.

La solución no es desactivar el versionado —se perdería la protección— sino caducar lo antiguo con ciclo de vida:

```
{
  "Rules": [{
    "ID": "caducar-versiones-antiguas",
    "Status": "Enabled",
    "Filter": {"Prefix": ""},
    "NoncurrentVersionExpiration": {"NoncurrentDays": 90, "NewerNoncurrentVersions": 3},
    "Expiration": {"ExpiredObjectDeleteMarker": true},
    "AbortIncompleteMultipartUpload": {"DaysAfterInitiation": 7}
  }]
}
```

La última regla merece mención aparte: las subidas multiparte abandonadas no aparecen en ningún listado normal y se facturan indefinidamente. Es una partida invisible que en buckets con mucha escritura fallida puede alcanzar cientos de gigabytes.

3. Clases de almacenamiento: el ahorro que a veces no lo es

Clase	USD/GB/mes	Recuperación	Mínimo	Latencia
Standard	0,023	0	—	ms
Intelligent-Tiering	0,023 → 0,0125	0	—	ms
Standard-IA	0,0125	0,01 USD/GB	30 días	ms
Glacier Instant	0,004	0,03 USD/GB	90 días	ms
Glacier Flexible	0,0036	0,01-0,03	90 días	min-horas
Deep Archive	0,00099	0,02 USD/GB	180 días	12 h

Las dos columnas de la derecha son las que invalidan la mitad de las transiciones que se configuran:

El mínimo de duración se factura completo. Un objeto movido a Standard-IA y borrado a los 10 días paga 30. Para datos con vida corta, la transición cuesta más de lo que ahorra.

La recuperación se cobra. Si los datos se leen con frecuencia, el coste de acceso supera el ahorro de almacenamiento:

```
1 TB en Standard:      1.024 × 0,023 = 23,55 USD/mes
1 TB en Standard-IA:   1.024 × 0,0125 = 12,80 USD/mes
ahorro bruto                         = 10,75 USD/mes

umbral de lecturas: 10,75 / 0,01 USD/GB = 1.075 GB/mes
→ si se lee más del 105 % del volumen al mes, Standard-IA sale MÁS CARO
```

Y hay un coste que se olvida: la propia transición cuesta unos 0,01 USD por cada 1.000 objetos. Con 40 millones de objetos pequeños, mover a otra clase cuesta 400 USD de una vez, y si los objetos son de 20 KB el ahorro mensual puede no compensarlo nunca.

De ahí que Intelligent-Tiering sea razonable cuando el patrón de acceso es desconocido: mueve automáticamente según el uso real, sin coste de recuperación, a cambio de una pequeña tarifa de monitorización por objeto que solo compensa con objetos mayores de 128 KB.

4. Bloquear el acceso público en todos los niveles

Las brechas de almacenamiento expuesto siguen ocurriendo porque hay cuatro mecanismos independientes que pueden conceder acceso público, y bloquear uno no bloquea los demás:

```
1. ACL del bucket
2. ACL del objeto
3. Política del bucket
4. Política del punto de acceso
```

El bloqueo de acceso público los cubre todos, y debe aplicarse a nivel de cuenta, no solo de bucket:

```
$ aws s3control put-public-access-block --account-id 123456789012 \
    --public-access-block-configuration \
    'BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBuckets=true'
```

Los cuatro parámetros hacen cosas distintas y hay que entender por qué son cuatro:

```
BlockPublicAcls        impide CREAR ACL públicas nuevas
IgnorePublicAcls       IGNORA las ACL públicas que ya existan
BlockPublicPolicy      impide crear políticas públicas nuevas
RestrictPublicBuckets  bloquea el acceso a través de políticas ya existentes
```

Las de «crear» no arreglan lo que ya está mal; las de «ignorar» y «restringir» sí. Aplicar solo las dos primeras deja abierto lo que ya estaba abierto, que es exactamente el caso de la brecha de la clase 010.

La verificación no puede ser por inspección:

```
$ curl -s -o /dev/null -w '%{http_code}\n' https://cloudshop-facturas.s3.amazonaws.com/f-1042.pdf
403                                                  ✓ sin credenciales
$ aws s3api put-bucket-acl --bucket cloudshop-facturas --acl public-read
AccessDenied ... blocked by the public access block    ✓ no se puede abrir
$ aws s3api get-object --bucket cloudshop-facturas --key f-1042.pdf /dev/null
{"ContentLength": 48219}                             ✓ con credenciales sí
```

Las tres pruebas juntas: niega sin credenciales, impide abrirlo, y permite el acceso legítimo. Solo la primera no demuestra nada — el objeto podría estar simplemente mal nombrado.

5. Replicación no es copia de seguridad

La confusión es cara. La replicación copia cambios a otro bucket, y un borrado es un cambio:

```
replicación   protege de: pérdida de una región, latencia de lectura lejana
              NO protege de: borrado, cifrado por ransomware, corrupción lógica
              porque propaga esas operaciones al destino

copia de seguridad  protege de: todo lo anterior
              porque es un punto en el tiempo aislado del origen
```

Hay un matiz importante: por defecto la replicación no replica los borrados —los marcadores de borrado no se replican salvo que se active DeleteMarkerReplication—. Eso da una protección parcial, pero no cubre el caso peligroso: si un atacante borra versiones con DeleteObjectVersion, esa operación sí se propaga.

La configuración que sí resiste:

```
origen    versionado + bloqueo de objetos + política que deniega DeleteObjectVersion
destino   cuenta DISTINTA, con sus propias credenciales y su propio bloqueo
```

La cuenta distinta es la pieza decisiva, por lo visto en la clase 017: si el destino está en la misma cuenta, unas credenciales comprometidas alcanzan ambos. Con cuentas separadas, el atacante necesita comprometer dos.

Y como estableció la clase 019, nada de esto vale sin la prueba:

```
$ aws s3api list-object-versions --bucket cloudshop-backup --prefix f-1042.pdf \
    --query 'Versions[0].[VersionId,LastModified]' --output text
3sL9mQt...   2026-08-01T07:12:44Z
```

```
$ aws s3api get-object --bucket cloudshop-backup --key f-1042.pdf \
    --version-id 3sL9mQt... /tmp/r.pdf && sha256sum /tmp/r.pdf
```

Restaurar y comparar la suma de comprobación con el original. Que exista la copia no demuestra que sea recuperable ni que sea correcta.

Ejemplo trabajado

Auditoría del almacenamiento de CloudShop: 2,25 TB facturados, un bucket con facturas de clientes y ninguna prueba de recuperación.

Hallazgo 1 — el 82 % del coste es historial invisible:

```
$ aws s3api list-object-versions --bucket cloudshop-datos --output json \
  | jq '{vigentes:([.Versions[]|select(.IsLatest)|.Size]|add),
         antiguas:([.Versions[]|select(.IsLatest|not)|.Size]|add),
         marcadores:(.DeleteMarkers|length)}'
{"vigentes": 412000000000, "antiguas": 1840000000000, "marcadores": 28417}
```

```
vigentes     412 GB ×  0,023 =  9,48 USD/mes
antiguas   1.840 GB ×  0,023 = 42,32 USD/mes   ← el 82 %
```

Se añaden subidas multiparte abandonadas, que no salían en ningún listado:

```
$ aws s3api list-multipart-uploads --bucket cloudshop-datos --query 'length(Uploads)'
1247
```

1.247 subidas incompletas, algunas de 2024, facturándose desde entonces.

Ciclo de vida aplicado:

```
{"Rules":[{"ID":"limpieza","Status":"Enabled","Filter":{"Prefix":""},
  "NoncurrentVersionExpiration":{"NoncurrentDays":90,"NewerNoncurrentVersions":3},
  "Expiration":{"ExpiredObjectDeleteMarker":true},
  "AbortIncompleteMultipartUpload":{"DaysAfterInitiation":7}}]}
```

```
almacenamiento tras 90 días:  412 GB vigentes + ~180 GB de historial reciente
coste            51,80 → 13,62 USD/mes   (−74 %)
```

Hallazgo 2 — una transición configurada que costaba dinero. El equipo había movido las miniaturas a Standard-IA:

```
objetos           38.400.000 miniaturas de ~18 KB
volumen           691 GB
lecturas/mes      1.240 GB   (las miniaturas se leen constantemente)

Standard:     691 × 0,023                    = 15,89 USD/mes
Standard-IA:  691 × 0,0125 + 1.240 × 0,01    =  8,64 + 12,40 = 21,04 USD/mes
```

La transición encarecía un 32 %. Además, los objetos de 18 KB están por debajo del umbral de 128 KB, así que ni Intelligent-Tiering compensaría. Se devuelven a Standard.

Hallazgo 3 — el bucket de facturas.

```
$ aws s3api get-public-access-block --bucket cloudshop-facturas \
    --query 'PublicAccessBlockConfiguration' --output json
{"BlockPublicAcls": true, "IgnorePublicAcls": false,
 "BlockPublicPolicy": true, "RestrictPublicBuckets": false}
```

Dos de cuatro. Impide crear accesos públicos nuevos y no restringe los que ya existen. La política del bucket tenía un Principal: "" de 2024, exactamente el caso de la clase 010.

```
$ curl -s -o /dev/null -w '%{http_code}\n' https://cloudshop-facturas.s3.amazonaws.com/f-1042.pdf
200                                              ← accesible sin credenciales
```

Corrección en los cuatro parámetros y a nivel de cuenta:

```
$ aws s3control put-public-access-block --account-id 123456789012 \
    --public-access-block-configuration \
    'BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBuckets=true'
$ curl -s -o /dev/null -w '%{http_code}\n' https://cloudshop-facturas.s3.amazonaws.com/f-1042.pdf
403                                              ✓
$ aws s3api put-bucket-acl --bucket cloudshop-facturas --acl public-read
AccessDenied                                     ✓ no se puede reabrir
$ aws s3api get-object --bucket cloudshop-facturas --key f-1042.pdf /dev/null >/dev/null && echo ok
ok                                               ✓ acceso legítimo intacto
```

Y se añade retención inmutable, que el versionado por sí solo no daba:

```
bloqueo de objetos en modo cumplimiento, 2.555 días (7 años de retención legal)
réplica a bucket en cuenta distinta, con su propio bloqueo
restauración probada: objeto recuperado y sha256 idéntico al original  ✓
```

Resultado:

```
coste de almacenamiento            67,69 USD    29,51 USD   (−56 %)
facturas accesibles sin credencial     sí           no
borrado de versiones posible           sí           no (cumplimiento)
restauración probada                   nunca      sí, con hash verificado
subidas abandonadas                  1.247          0
```

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-02-aws-core-platform/030-s3-objetos-versionado-lifecycle-y-replicacion/lab.py
```

El laboratorio selecciona el motor de práctica storage 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 bucket-gobernado-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 una política de durabilidad, acceso, retención y costo. Además, la entrega debe incluir el comando ejecutado, la salida estructurada y una conclusión que no exceda lo observado.

Reto verificable

Construye bucket-gobernado-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
La factura de almacenamiento no baja después de borrar objetos	En un bucket versionado, DELETE crea un marcador y las versiones siguen facturándose	Ciclo de vida con NoncurrentVersionExpiration y limpieza de marcadores caducados.
Se factura almacenamiento que no aparece en ningún listado	Subidas multiparte abandonadas: invisibles en s3 ls y facturadas indefinidamente	Añade AbortIncompleteMultipartUpload al ciclo de vida.
Mover datos a una clase más fría encarece la factura	El coste de recuperación y el mínimo de duración superan el ahorro por GB	Calcula el umbral de lecturas; con acceso frecuente u objetos pequeños, la clase fría pierde.
Se activa el bloqueo de acceso público y el bucket sigue expuesto	Solo se activaron los parámetros de «crear», no los de «ignorar» y «restringir»	Activa los cuatro y a nivel de cuenta; verifica con petición sin credenciales.
Un borrado malicioso se propaga al bucket réplica	La replicación copia cambios, y borrar versiones es un cambio	Réplica en cuenta distinta con bloqueo de objetos propio; la replicación no sustituye a la copia de seguridad.

■ 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 riesgo cubre el bloqueo de objetos que el versionado no cubre, y por qué el modo cumplimiento es el único que sirve ahí?
2. Tras aws s3 rm, el objeto no aparece en el listado. ¿Se está pagando? Justifica.
3. 1 TB en Standard-IA se lee 1,5 TB al mes. ¿Ahorra frente a Standard? Haz el cálculo.
4. ¿Cuál de los cuatro parámetros de bloqueo público cierra un bucket que YA estaba expuesto?
5. ¿Por qué una réplica en la misma cuenta no protege de unas credenciales comprometidas?

Referencias

- AWS (2024). Using versioning in S3 buckets — versiones, marcadores de borrado y facturación. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Versioning.html>
- AWS (2024). S3 Object Lock — modos gobernanza y cumplimiento, y sus límites. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/object-lock.html>
- AWS (2024). Blocking public access to your Amazon S3 storage — semántica de los cuatro parámetros. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-control-block-public-access.html>
- AWS (2024). Managing your storage lifecycle — transiciones, mínimos de duración y limpieza de multiparte. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/object-lifecycle-mgmt.html>
- AWS (2024). S3 storage classes — precios, latencias de recuperación y umbrales de tamaño. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/storage-class-intro.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 02 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 029 · Elastic Load Balancing y Auto Scaling	Parte 02 · Programa	031 · RDS, DynamoDB y ElastiCache: decisión de datos →

Evaluación — 030 S3: objetos, versionado, lifecycle y replicación

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 bucket-gobernado-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.

031 — RDS, DynamoDB y ElastiCache: decisión de datos

Parte: 02 — AWS: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: data · Estado: EXECUTABLECORE

Propósito

Elegir motor de datos a partir de los patrones de acceso, no del catálogo del proveedor. Es la decisión más irreversible de la parte —cambiar de motor con terabytes en producción es una migración, no un ajuste— y la que más determina el coste y la latencia del sistema durante años.

Resultados de aprendizaje

Al finalizar podrás:

1. Derivar el modelo de datos desde los patrones de acceso en lugar de normalizar primero y consultar después.
2. Distinguir cuándo una carga necesita transacciones multiclave y cuándo no, con consecuencias de elección.
3. Dimensionar capacidad en DynamoDB y detectar una clave de partición mal elegida antes de que sature.
4. Justificar una caché por el patrón de lectura y explicar sus dos modos de invalidación.
5. Calcular el coste de las tres opciones para una carga concreta y no solo su idoneidad técnica.

Conceptos centrales

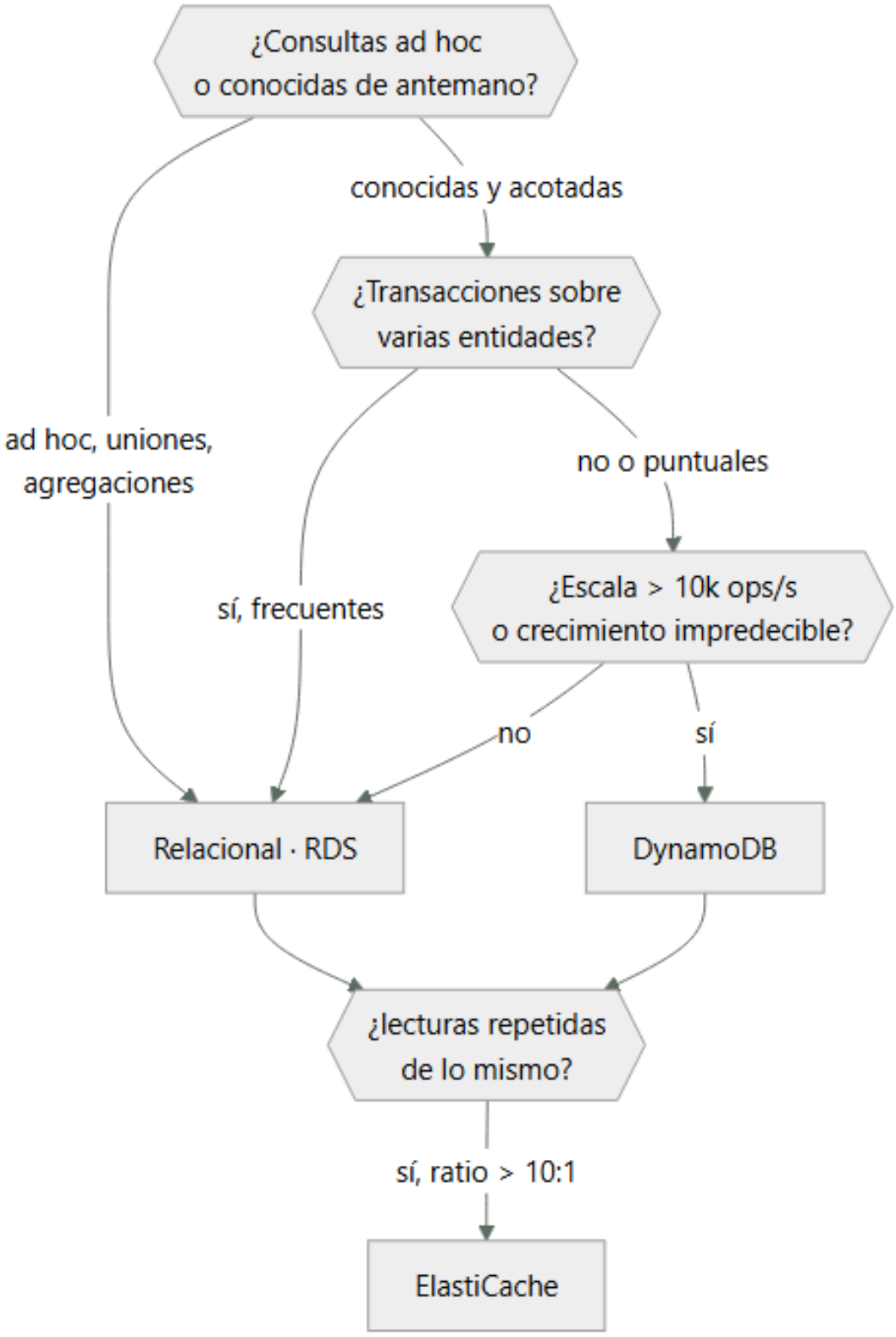
Concepto	Comprensión verificable
patrón de acceso	Consulta concreta que la aplicación hará, con su frecuencia y su latencia exigida. En almacenes no relacionales es la entrada del diseño; enumerarlos mal produce un modelo que no se puede consultar.
clave de partición	Atributo que determina en qué partición física vive un elemento. Una clave con pocos valores distintos concentra el tráfico en pocas particiones y produce una partición caliente.
partición caliente	Partición que recibe una fracción desproporcionada del tráfico y satura antes que el resto. Se manifiesta como limitación de tasa mientras la capacidad agregada parece sobrada.
índice secundario global	Proyección de una tabla con otra clave, que permite consultas por atributos distintos. Tiene su propia capacidad y se actualiza de forma asíncrona: es coherente en último término.
invalidación de caché	Mecanismo para que la copia deje de servirse cuando el origen cambia. Los dos modos —caducidad y escritura activa— tienen fallos distintos: datos obsoletos frente a complejidad y acoplamiento.

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

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 patrones de acceso van primero

En un motor relacional se normaliza el modelo y después se consulta como haga falta: el optimizador se encarga. En un almacén de clave-valor no hay optimizador, así que el modelo debe derivarse de las consultas.

La disciplina es enumerar los patrones antes de diseñar nada:

```
P1  obtener un pedido por su id                        12.000/min   < 10 ms
P2  listar los pedidos de un cliente, más recientes     3.400/min   < 30 ms
P3  listar los pedidos de un estado en una fecha           40/min   < 2 s
P4  total facturado por mes                                2/día    < 30 s
```

Con esa lista, la elección se decide sola:

- P1 y P2 son consultas por clave conocida: un almacén de clave-valor las sirve con latencia de un dígito de milisegundos y escala sin límite práctico.
- P3 necesita un índice secundario.
- P4 es una agregación analítica: no pertenece al almacén operativo, y forzarla ahí degrada P1 y P2.

El error clásico es diseñar el modelo relacional primero, migrarlo tal cual a un almacén no relacional y descubrir que P2 requiere escanear la tabla entera. Un escaneo completo en un almacén de clave-valor no es una consulta lenta: es una consulta que cuesta dinero proporcional al tamaño y que empeora cada mes.

Y el criterio inverso también aplica: si los patrones de acceso no se pueden enumerar —porque el negocio hace preguntas nuevas cada semana—, un motor relacional es la elección correcta, y no una concesión.

2. Transacciones: qué se pierde y qué cuesta recuperarlo

Un motor relacional da atomicidad sobre varias tablas con una sola sentencia. En un almacén de clave-valor eso existe pero con límites duros que hay que conocer antes de diseñar:

```
DynamoDB TransactWriteItems:
  hasta 100 elementos por transacción
  máximo 4 MB en total
  consume el DOBLE de capacidad de escritura
  no puede abarcar regiones
```

La consecuencia de diseño: si la operación central del negocio toca cinco entidades y ocurre miles de veces por segundo, el coste doble y el límite de 100 elementos importan. Si ocurre decenas de veces por minuto, no.

La alternativa cuando la transacción no cabe es el patrón de saga: descomponer en pasos con compensación. No es gratis:

```
transacción     todo o nada, garantizado por el motor
saga            pasos independientes con compensación explícita
                → hay estados intermedios visibles
                → la compensación puede fallar y hay que reintentarla
                → el código de compensación se ejercita poco y suele estar mal probado
```

La regla práctica: elegir saga por diseño, no por descubrir tarde que la transacción no cabía. Un sistema que necesita compensación en su camino principal está pagando complejidad permanente por una decisión de motor.

Y una precisión sobre consistencia: DynamoDB ofrece lectura fuertemente consistente en la tabla base —cuesta el doble de capacidad de lectura— pero los índices secundarios globales son siempre coherentes en último término. Una escritura seguida de una consulta por índice puede no ver el cambio. Ese retraso es de milisegundos en régimen normal y puede crecer bajo carga.

3. La clave de partición decide si escala

DynamoDB reparte los datos por el hash de la clave de partición. Una clave con pocos valores distintos concentra el tráfico:

```
clave = estado_pedido    (5 valores: nuevo, pagado, enviado, entregado, cancelado)
  → 5 particiones lógicas para toda la tabla
  → el 80 % del tráfico va a "nuevo"
  → esa partición satura mientras la capacidad agregada parece sobrada
```

El síntoma es característico y desconcierta: ProvisionedThroughputExceededException con la capacidad global muy por debajo del límite. La partición está limitada a 3.000 unidades de lectura y 1.000 de escritura por segundo, sea cual sea la capacidad total de la tabla.

Una clave adecuada tiene alta cardinalidad y reparto uniforme:

```
clave = pedido_id (UUID)          → millones de valores, reparto uniforme  ✓
clave = cliente_id                → miles de valores, pero un cliente grande
                                    puede concentrar tráfico              ~
```

```
clave = fecha (2026-08-01)        → todo el tráfico del día en una        ✗
```

El segundo caso es el más traicionero: funciona en pruebas y falla cuando un cliente crece. La mitigación es añadir sufijo de dispersión:

```
clave = f"{cliente_id}#{hash(pedido_id) % 10}"
  → reparte cada cliente en 10 particiones
  → coste: consultar todos los pedidos de un cliente exige 10 consultas
```

Es un intercambio explícito: se gana reparto y se pierde simplicidad de consulta. Como todo en esta clase, la decisión debe estar escrita.

Y el modo bajo demanda no elimina el problema: escala automáticamente la capacidad total, pero el límite por partición sigue existiendo. Una clave mal elegida satura igual, solo que la factura crece antes de que se note.

4. Caché: cuándo aporta y cómo se invalida

Una caché solo aporta si el mismo dato se lee muchas más veces de las que se escribe. El indicador es la relación lectura-escritura:

```
ratio < 3:1     la caché añade complejidad y poco beneficio
ratio 10:1      empieza a compensar
ratio > 100:1   claramente rentable
```

Y el ahorro se calcula, no se supone:

```
lecturas/mes                    840.000.000
tasa de acierto medida                 92 %
lecturas evitadas al origen     772.800.000
coste evitado (0,25 USD por millón de lecturas fuertes) = 193 USD/mes
coste de la caché (cache.r7g.large × 2)                 = 208 USD/mes
```

No compensa por coste con esos números; compensa por latencia, que es otra justificación válida y que hay que declarar como tal:

```
latencia desde el origen    8-12 ms
latencia desde la caché    0,3-0,8 ms
```

Los dos modos de invalidación tienen fallos distintos:

Modo	Cómo	Fallo característico
Caducidad	TTL fijo; se relee al expirar	Datos obsoletos hasta el TTL
Escritura activa	La escritura actualiza caché y origen	Se desincroniza si una de las dos falla

El segundo parece mejor y acopla la escritura a la disponibilidad de la caché. Si la caché no responde, ¿falla la escritura o se queda desincronizada? Ninguna respuesta es buena, y por eso la caducidad suele ser preferible salvo que la obsolescencia sea inaceptable.

Hay un tercer fallo que afecta a ambos, la estampida: cuando una clave muy consultada caduca, todas las peticiones simultáneas van al origen a la vez. Se mitiga con bloqueo de recálculo o con caducidad aleatorizada —el mismo jitter de la clase 004—.

5. El coste también decide, y a veces al revés

Comparar motores solo por idoneidad técnica lleva a decisiones caras. Los tres modelos de precio son estructuralmente distintos:

```
RDS         por hora de instancia + almacenamiento + E/S
            → coste fijo, predecible, independiente del tráfico

DynamoDB    por unidades de lectura y escritura consumidas + almacenamiento
            → coste proporcional al tráfico, cero si no hay uso

ElastiCache por hora de nodo
            → coste fijo
```

Para una carga con tráfico muy variable, DynamoDB bajo demanda puede ser mucho más barato:

```
carga con 2 horas de pico al día y el resto casi inactiva
RDS db.r6g.xlarge Multi-AZ:   730 h × 0,96 USD          = 700,80 USD/mes
DynamoDB bajo demanda:
```

```
escrituras 42 M × 1,25 USD/millón                    =  52,50
lecturas  310 M × 0,25 USD/millón                    =  77,50
almacenamiento 180 GB × 0,25                         =  45,00
                                                       --------
                                                       175,00 USD/mes
```

Pero con tráfico alto y sostenido se invierte:

```
tráfico sostenido de 4.000 escrituras/s
DynamoDB bajo demanda: 10.368 M escrituras × 1,25/millón = 12.960 USD/mes
DynamoDB aprovisionado con autoescalado                ≈  2.600 USD/mes
RDS db.r6g.4xlarge Multi-AZ                            ≈  2.800 USD/mes
```

Bajo demanda cuesta cinco veces más que aprovisionado con tráfico predecible. La regla: bajo demanda para tráfico variable o desconocido; aprovisionado con autoescalado cuando el patrón se estabiliza.

Y una partida que se olvida: en RDS, la E/S se factura aparte en algunas clases de almacenamiento. Una consulta sin índice que escanea millones de filas no solo es lenta: aparece en la factura.

Ejemplo trabajado

CloudShop tiene los pedidos en RDS PostgreSQL. La tabla ha llegado a 180 GB, el p95 de la consulta principal es de 340 ms y la factura de base de datos es de 4.900 USD/mes. Se evalúa el cambio.

Primero, los patrones de acceso reales, medidos en 30 días:

```
SELECT queryid, calls, mean_exec_time, rows
FROM pg_stat_statements ORDER BY calls DESC LIMIT 4;
```

```
P1  SELECT * FROM pedidos WHERE id = $1          17.280.000    2,1 ms      1
P2  SELECT * FROM pedidos WHERE cliente_id = $1
      ORDER BY creado DESC LIMIT 20               4.896.000  312,0 ms     20
P3  SELECT * FROM pedidos WHERE estado=$1
      AND creado > $2                                57.600  890,0 ms  ~4000
P4  agregación mensual por mes                           60  8.400 ms      1
```

P2 domina el problema: 4,9 millones de llamadas diarias a 312 ms. Antes de cambiar de motor se comprueba si es un problema de índice:

```
EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM pedidos
  WHERE cliente_id = 8241 ORDER BY creado DESC LIMIT 20;
```

```
Index Scan using idx_pedidos_cliente on pedidos  (cost=0.56..18402.11)
  Rows Removed by Filter: 0
  Buffers: shared hit=142 read=8891          ← 8.891 bloques de disco
  Execution Time: 309.442 ms
```

El índice existe pero es solo sobre clienteid: PostgreSQL lo usa para filtrar y después ordena 12.400 filas para devolver 20. Se prueba un índice compuesto:

```
CREATE INDEX CONCURRENTLY idx_pedidos_cliente_creado
  ON pedidos (cliente_id, creado DESC);
```

```
Execution Time: 1.284 ms      ← de 312 ms a 1,3 ms
Buffers: shared hit=24 read=0
```

Un índice compuesto resolvió el 96 % del problema. El motor no era el problema: el modelo de índices sí.

Se reevalúa la situación con ese dato:

```
p95 global                340 ms          38 ms
E/S facturada          1,2 TB/mes      0,18 TB/mes
coste                 4.900 USD       4.130 USD
```

Ahora sí se compara con la alternativa, con los patrones ya optimizados:

```
P1 por id                              2,1 ms           3 ms
P2 por cliente, ordenado               1,3 ms           4 ms   (clave compuesta)
P3 por estado y fecha                  890 ms          GSI, ~40 ms
P4 agregación mensual                 8.400 ms         NO SOPORTADO
transacciones multiclave                 sí             límite 100 elementos
coste mensual estimado              4.130 USD        1.980 USD
migración                                —           ~3 meses + reescritura
```

P4 es el bloqueo: la agregación mensual no existe en DynamoDB sin exportar a otro sistema. Y la migración cuesta tres meses para ahorrar 2.150 USD al mes, con recuperación en 4 meses de trabajo de dos personas.

Decisión: quedarse en RDS. Se añaden dos mejoras acotadas:

```
1. Caché para P1, que es el 78 % del tráfico
   ratio lectura/escritura medido: 47:1  → compensa
   tasa de acierto esperada 90 %
   latencia 2,1 ms → 0,4 ms
   coste: +104 USD/mes

2. P4 a una réplica de lectura, para que la agregación de 8,4 s
   deje de competir con el tráfico operativo
   coste: +380 USD/mes
```

```
p95                          340 ms     31 ms
coste                      4.900 USD  4.614 USD
esfuerzo                      —       2 días
```

La lección: se estaba evaluando un cambio de motor de tres meses para resolver un problema que era un índice mal definido. El análisis de patrones de acceso —que se hizo para elegir motor— acabó descartando el cambio.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-02-aws-core-platform/031-rds-dynamodb-y-elasticache-decision-de-datos/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 matriz-datos-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 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 matriz-datos-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
Una consulta usa el índice y aun así tarda cientos de milisegundos	El índice filtra pero no cubre el orden, así que el motor ordena miles de filas	Índice compuesto que incluya las columnas de ordenación; comprueba con EXPLAIN ANALYZE y los bloques leídos.
DynamoDB limita la tasa con la capacidad global muy por debajo del límite	Partición caliente: la clave tiene poca cardinalidad o un valor concentra el tráfico	Elige una clave de alta cardinalidad o añade sufijo de dispersión, asumiendo el coste de consulta múltiple.
Una escritura seguida de consulta por índice secundario no ve el cambio	Los índices secundarios globales son coherentes en último término	Consulta la tabla base cuando necesites leer lo que acabas de escribir.
El coste de DynamoDB se dispara al estabilizarse el tráfico	Bajo demanda cuesta varias veces más que aprovisionado con carga predecible	Cambia a aprovisionado con autoescalado en cuanto el patrón sea estable.
Al caducar una clave muy consultada, el origen recibe una avalancha	Estampida de caché: todas las peticiones recalculan a la vez	Caducidad aleatorizada o bloqueo de recálculo, igual que el jitter en los reintentos.

■ 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 un almacén de clave-valor los patrones de acceso van antes que el modelo, y qué pasa si se invierte el orden?
2. Una tabla tiene la clave de partición fecha. ¿Qué ocurrirá y por qué el modo bajo demanda no lo resuelve?
3. ¿Qué le cuesta a una transacción de DynamoDB en capacidad, y cuál es su límite de elementos?
4. Con ratio lectura/escritura de 3:1, ¿conviene una caché? ¿Y qué otra justificación podría hacerla válida igualmente?
5. ¿En qué caso bajo demanda es cinco veces más caro que aprovisionado, y por qué?

Referencias

- AWS (2024). DynamoDB best practices: partition key design — cardinalidad, particiones calientes y dispersión. <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-partition-key-design.html>
- AWS (2024). DynamoDB transactions — límites de elementos, tamaño y consumo de capacidad. <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/transaction-apis.html>
- Kleppmann, M. (2017). Designing Data-Intensive Applications, caps. 2-3 — modelos de datos, índices y motores de almacenamiento.
- PostgreSQL (2024). Using EXPLAIN — lectura de planes y coste de E/S. <https://www.postgresql.org/docs/current/using-explain.html>
- AWS (2024). ElastiCache caching strategies — caducidad, escritura activa y mitigación de estampidas. <https://docs.aws.amazon.com/AmazonElastiCache/latest/dg/Strategies.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 02 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 030 · S3: objetos, versionado, lifecycle y replicación	Parte 02 · Programa	032 · Lambda, API Gateway y Step Functions →

Evaluación — 031 RDS, DynamoDB y ElastiCache: decisión de datos

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 matriz-datos-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.

032 — Lambda, API Gateway y Step Functions

Parte: 02 — AWS: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: serverless · Estado: EXECUTABLECORE

Propósito

Construir una API serverless sabiendo dónde están sus límites duros y cuáles son negociables. La clase 015 dio el criterio para elegir el modelo; aquí se aplica a AWS y se añade lo que decide si funciona en producción: la concurrencia como recurso compartido de la cuenta, el arranque en frío frente al percentil del SLO, y por qué una función que llama a una base relacional agota sus conexiones.

Resultados de aprendizaje

Al finalizar podrás:

1. Calcular la concurrencia que consumirá una función a partir de su tasa de invocación y su duración.
2. Anticipar el efecto de una función que agota la concurrencia de la cuenta sobre las demás.
3. Decidir entre concurrencia aprovisionada y optimización del paquete según el percentil del SLO.
4. Resolver el agotamiento de conexiones al llamar a una base de datos relacional desde funciones.
5. Elegir entre orquestación con máquina de estados y coreografía por eventos según trazabilidad y acoplamiento.

Conceptos centrales

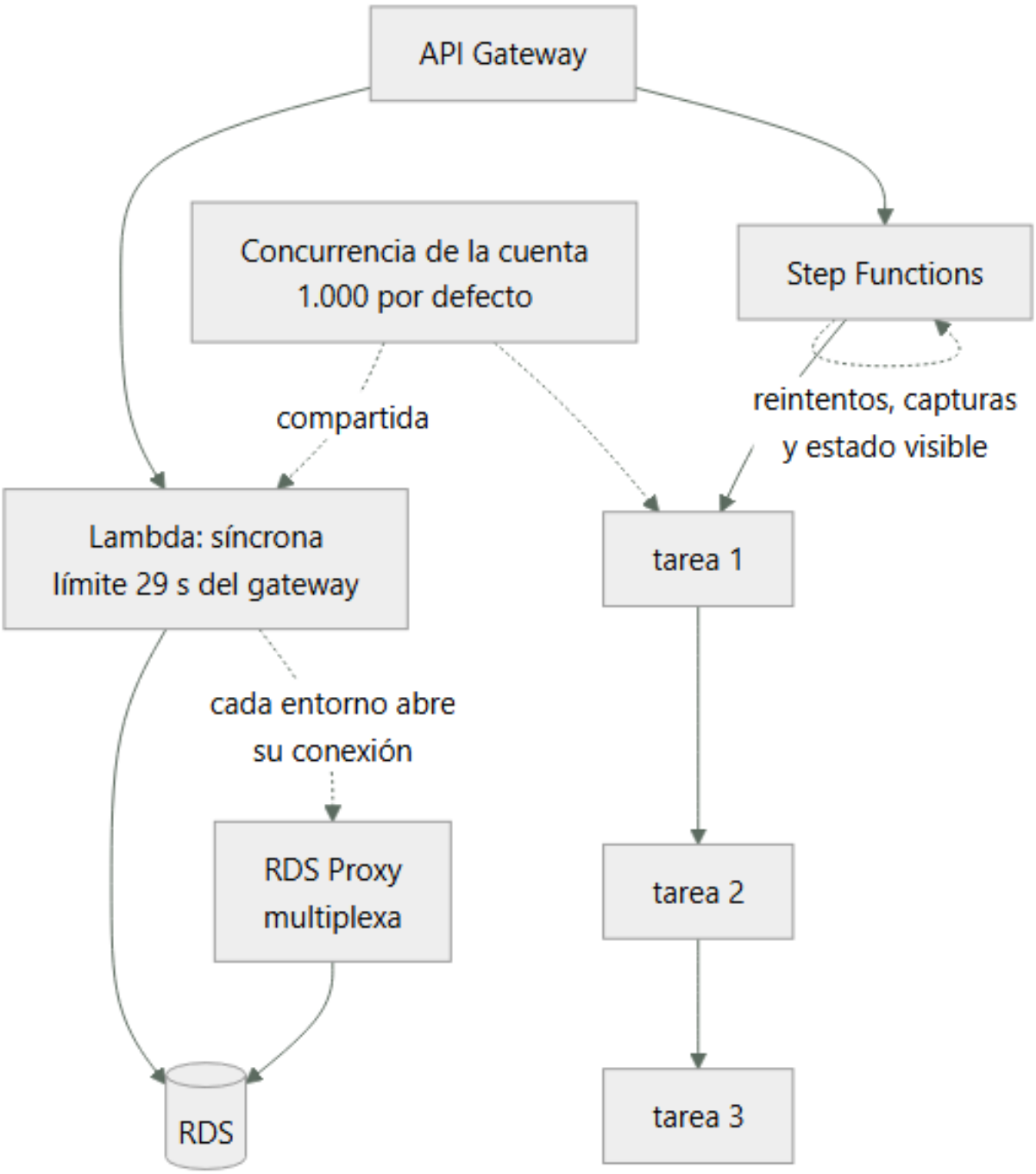
Concepto	Comprensión verificable
concurrencia	Invocaciones ejecutándose simultáneamente. Es el recurso limitado de Lambda y se comparte entre todas las funciones de la cuenta y región: 1.000 por defecto.
concurrencia reservada	Porción de la concurrencia de la cuenta apartada para una función. Actúa a la vez como garantía —nadie se la quita— y como techo —no puede superarla—.
concurrencia aprovisionada	Entornos inicializados y mantenidos calientes. Elimina el arranque en frío y elimina también el ahorro de escalar a cero: se paga aunque no se invoque.
proxy de conexiones	Capa que multiplexa conexiones hacia una base de datos relacional. Resuelve que cada entorno de ejecución abra la suya y agote el límite del motor.
máquina de estados	Orquestador que define el flujo de forma explícita, con reintentos, capturas y estado visible. Se opone a la coreografía, donde cada componente reacciona a eventos sin visión global.

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

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

■ Flujo de razonamiento



Desarrollo

1. La concurrencia es el recurso que se agota, no la CPU

Lambda no escala por CPU ni por memoria: escala creando entornos de ejecución. El número de entornos simultáneos es la concurrencia, y se calcula con la ley de Little de la clase 011:

```
concurrencia = invocaciones por segundo × duración en segundos
```

```
800 inv/s × 0,250 s = 200 concurrentes
```

El límite por defecto es 1.000 por cuenta y región, compartido entre todas las funciones. De ahí el modo de fallo más desconcertante del modelo:

```
función A: procesamiento por lotes, 900 concurrentes durante 10 minutos
función B: API de pagos, necesita 120
→ B recibe TooManyRequestsException y devuelve 429 a los usuarios
→ B no cambió, no tiene errores, y su código está perfecto
```

Es exactamente el problema de vecino ruidoso de la clase 017, dentro de la misma cuenta. Y la solución es la misma: acotar.

```
# Techo para el proceso por lotes: no puede consumir más de 200
$ aws lambda put-function-concurrency --function-name lotes --reserved-concurrent-executions 200
# Garantía para la API crítica: 300 apartados, nadie se los quita
$ aws lambda put-function-concurrency --function-name pagos-api --reserved-concurrent-executions 300
```

La concurrencia reservada tiene un doble filo que hay que entender: es garantía y es techo. La función con 300 reservados nunca bajará de 300 disponibles y nunca pasará de 300, aunque el resto de la cuenta esté ocioso. Reservar de menos convierte la protección en una limitación.

Y hay un límite adicional sobre el ritmo de crecimiento: la concurrencia sube en incrementos de 1.000 por minuto por región tras la ráfaga inicial. Un pico que necesite 5.000 concurrentes en 30 segundos no se puede absorber escalando, igual que en la clase 016.

2. Arranque en frío: comparar con el percentil del SLO

La pregunta útil no es «¿hay arranque en frío?» sino «¿qué fracción de invocaciones lo sufre y dónde cae el percentil de mi SLO?».

```
invocaciones/mes         2.000.000
arranques en frío           12.000   → 0,6 %
latencia p50 en caliente        45 ms
latencia en frío               820 ms

SLO al p95  → el 0,6 % no lo alcanza; el p95 sigue siendo ~90 ms   ✓
SLO al p99  → el 0,6 % está dentro del 1 % peor; el p99 ES el frío  ✗
```

Factores que determinan la latencia de arranque, por impacto:

```
tamaño del paquete       250 MB descomprimido tarda mucho más que 5 MB
runtime                  JVM y .NET inicializan máquina virtual
inicialización propia    conexiones, carga de configuración, SDK
```

La inicialización propia es la más controlable y la peor aprovechada. El código fuera del manejador se ejecuta una vez por entorno, no por invocación:

```
# Fuera del manejador: se paga una vez por entorno
import boto3
cliente = boto3.client("dynamodb")      # reutilizado en invocaciones siguientes
config = cargar_configuracion()

def handler(event, context):
    return cliente.get_item(...)         # sin coste de inicialización
```

Ponerlo dentro del manejador multiplica el coste por cada invocación, no solo por cada arranque.

La concurrencia aprovisionada elimina el frío y cambia la economía por completo:

```
10 entornos aprovisionados de 512 MB:
  10 × 0,5 GB × 730 h × 0,0000097222 USD/GB-s × 3600 = 127,80 USD/mes
  se paga estén o no invocándose
```

Con eso, el cálculo de la clase 015 cambia: ya no se escala a cero, así que el umbral frente a un contenedor permanente se desplaza a favor del contenedor.

3. Funciones y bases de datos relacionales: el choque de modelos

Una función escala a cientos de entornos y cada entorno abre su propia conexión. Un motor relacional tiene un límite de conexiones muy inferior:

```
concurrencia de la función       400 entornos
conexiones por entorno             1
max_connections de la instancia  200      ← se agota al 50 % de la concurrencia
```

El síntoma: FATAL: sorry, too many clients already. Y no se arregla subiendo maxconnections, porque cada conexión de PostgreSQL consume memoria del servidor y a partir de cierto punto degrada el motor entero.

Las tres respuestas, de peor a mejor:

```
1. Reservar concurrencia por debajo del límite de conexiones
   → funciona y desperdicia la elasticidad que motivó usar funciones
```

```
2. Cerrar la conexión al final de cada invocación
   → añade el coste de establecerla a cada llamada (decenas de ms)

3. Proxy de conexiones
   → multiplexa: 400 entornos comparten un pool de 50 conexiones reales
```

La tercera es la correcta, y tiene un matiz que decide si funciona: el proxy solo puede reutilizar conexiones si la sesión no tiene estado. Una transacción abierta, una tabla temporal o una variable de sesión fuerzan a fijar la conexión a ese cliente hasta que termine, y con suficientes clientes fijados el pool se agota igual.

```
sin estado de sesión   reutilización alta, el proxy cumple su función
con transacciones largas  fijación de conexión, el proxy no ayuda
```

Y una alternativa estructural: si la carga es de funciones, un almacén de clave-valor encaja mejor que uno relacional, porque su modelo de acceso es sin conexión persistente. Es la decisión de la clase 031 vista desde el otro lado.

4. Los límites del gateway y cómo se rodean

API Gateway impone restricciones que no dependen de Lambda y que hay que conocer antes de diseñar:

Límite	Valor	Negociable
Tiempo de espera de integración	29 s	No
Tamaño de carga útil	10 MB	No
Peticiones por segundo	10.000 por región	Sí, con solicitud
Ráfaga	5.000	Sí

Los dos primeros son duros y determinan la arquitectura:

29 segundos es menos que los 15 minutos de Lambda. Una función que tarda 2 minutos funciona invocada directamente y falla siempre a través del gateway. El patrón correcto es asíncrono:

```
POST /informes  → 202 Accepted, devuelve id de trabajo
                  encola el trabajo
GET /informes/{id} → 200 con estado, o 303 al resultado cuando termina
```

10 MB de carga útil hace inviable subir ficheros grandes por la API. La solución es no pasarlos por ella:

```
POST /subidas   → devuelve una URL prefirmada de S3
cliente         → sube directamente a S3, sin límite del gateway
S3              → emite evento que dispara el procesamiento
```

Esto además ahorra dinero: los bytes no atraviesan el gateway ni la función.

Y una decisión que se toma al principio y cuesta cambiar: HTTP API frente a REST API. La primera cuesta aproximadamente un 70 % menos y tiene menos funciones —sin planes de uso, sin caché, sin validación de solicitud—. Empezar por REST API «por si acaso» es una de las formas más comunes de pagar de más sin usar nada de lo que se paga.

5. Orquestación frente a coreografía

Un flujo de varios pasos se puede construir de dos formas, y la elección determina cuánto cuesta diagnosticar un fallo:

	Orquestación (máquina de estados)	Coreografía (eventos)
Flujo	Explícito en un sitio	Emergente, repartido
Estado	Visible y consultable	Hay que reconstruirlo
Reintentos y compensación	Declarativos	En cada componente
Acoplamiento	Mayor: el orquestador conoce los pasos	Menor
Diagnóstico	Se ve dónde falló	Hay que correlacionar trazas
Coste	Por transición de estado	Por evento

La orquestación se declara y el motor se encarga de reintentar:

```
{
```

```
  "Type": "Task",
  "Resource": "arn:aws:lambda:...:function:cobrar",
  "Retry": [{
    "ErrorEquals": ["States.TaskFailed"],
    "IntervalSeconds": 2, "MaxAttempts": 3, "BackoffRate": 2.0
  }],
  "Catch": [{"ErrorEquals": ["States.ALL"], "Next": "CompensarReserva"}]
}
```

Esas nueve líneas sustituyen a código de reintento con retroceso exponencial repartido por varios servicios, y —más importante— hacen visible la compensación, que en coreografía suele estar implícita y sin probar.

El criterio práctico:

```
orquestación   flujos de negocio con compensación, pasos con orden,
               necesidad de auditar dónde quedó cada ejecución
coreografía    reacciones independientes, muchos consumidores del mismo hecho,
               equipos que no deben coordinarse para añadir un consumidor
```

Y sobre coste: Step Functions Standard cobra por transición de estado, lo que en flujos muy repetitivos se acumula. El modo Express cuesta por duración y es mucho más barato para flujos cortos de alto volumen, a cambio de garantía «al menos una vez» en vez de «exactamente una vez» —lo que exige idempotencia en los pasos, como en la clase 009—.

Ejemplo trabajado

La API de pagos de CloudShop empieza a devolver 429 sin que su tráfico haya cambiado, y en el mismo día aparecen errores de conexión contra la base de datos.

Síntoma 1 — 429 sin cambio de tráfico:

```
$ aws cloudwatch get-metric-statistics --namespace AWS/Lambda \
    --metric-name Throttles --dimensions Name=FunctionName,Value=pagos-api \
    --start-time 2026-08-01T09:00:00Z --end-time 2026-08-01T12:00:00Z \
    --period 300 --statistics Sum --query 'sum(Datapoints[].Sum)'
8412.0
$ aws lambda get-account-settings --query 'AccountLimit.ConcurrentExecutions'
1000
```

Se busca quién consume la concurrencia:

```
$ for f in pagos-api catalogo informes-lotes notificaciones; do
    printf "%-18s " $f
    aws cloudwatch get-metric-statistics --namespace AWS/Lambda \
      --metric-name ConcurrentExecutions --dimensions Name=FunctionName,Value=$f \
      --start-time 2026-08-01T10:00:00Z --end-time 2026-08-01T11:00:00Z \
      --period 3600 --statistics Maximum --query 'Datapoints[0].Maximum'
  done
pagos-api          118.0
catalogo            94.0
informes-lotes     742.0        ←
notificaciones      46.0
```

informes-lotes consume 742 de los 1.000. Se lanzó un reproceso histórico esa mañana. La API de pagos no cambió: se quedó sin sitio.

Verificación con la ley de Little de lo que cada una necesita:

```
pagos-api      340 inv/s × 0,32 s = 109 concurrentes → observado 118  ✓
informes-lotes  62 inv/s × 12,0 s = 744              → observado 742  ✓
```

Corrección con techo y garantía:

```
$ aws lambda put-function-concurrency --function-name informes-lotes \
    --reserved-concurrent-executions 200
$ aws lambda put-function-concurrency --function-name pagos-api \
    --reserved-concurrent-executions 250      # 118 observados + margen
```

```
con 200 reservados, informes-lotes tarda 3,7× más (62 → 16,7 inv/s efectivas)
y es trabajo por lotes sin plazo: aceptable y declarado
```

Síntoma 2 — conexiones agotadas. Aparece al aumentar la concurrencia:

```
FATAL: sorry, too many clients already
```

```
$ aws rds describe-db-parameters --db-parameter-group-name cloudshop-pg \
    --query 'Parameters[?ParameterName==`max_connections`].ParameterValue' --output text
```

```
200

concurrencia combinada de las funciones que tocan la base:  118 + 94 + 200 = 412
conexiones disponibles                                          200
→ se agotan al 48 % de la concurrencia
```

Se evalúan las tres salidas:

```
1. bajar la concurrencia por debajo de 200 → renuncia a la elasticidad
2. cerrar conexión por invocación         → +18 ms medidos por llamada
3. proxy de conexiones                    → multiplexa
```

Se elige el proxy y se comprueba la condición que decide si sirve:

```
$ aws rds create-db-proxy --db-proxy-name cloudshop-proxy \
    --engine-family POSTGRESQL --require-tls ...
$ aws cloudwatch get-metric-statistics --namespace AWS/RDS \
    --metric-name DatabaseConnectionsCurrentlySessionPinned \
    --dimensions Name=ProxyName,Value=cloudshop-proxy \
    --period 300 --statistics Maximum --query 'Datapoints[0].Maximum'
3.0
```

Solo 3 conexiones fijadas de 412 clientes: el código no usa transacciones largas ni estado de sesión, así que la multiplexación funciona.

```
conexiones reales al motor          412 (falla)   47
concurrencia sostenible             ~190         412
latencia p95 de pagos-api           340 ms      112 ms
throttles/hora                      2.804          0
```

Los dos síntomas tenían la misma raíz: recursos compartidos sin acotar —concurrencia de la cuenta y conexiones del motor—. Ninguno era un problema de la función que fallaba.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-02-aws-core-platform/032-lambda-api-gateway-y-step-functions/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 api-serverless-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 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 api-serverless-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
Una función devuelve 429 sin que su tráfico haya cambiado	Otra función consumió la concurrencia compartida de la cuenta	Reserva concurrencia como techo para lo interrumpible y como garantía para lo crítico.
Reservar concurrencia protege la función pero la limita en el pico	La concurrencia reservada es garantía y también techo	Dimensiona el reservado por encima del máximo observado más margen; reservar de menos es limitarse.
Errores de conexión contra la base de datos al escalar la función	Cada entorno abre su conexión y el motor tiene un límite muy inferior	Usa un proxy de conexiones y comprueba la métrica de conexiones fijadas para saber si multiplexa.
Una función de 2 minutos falla siempre a través del gateway	El tiempo de espera de integración es de 29 s y no es negociable	Patrón asíncrono: 202 con identificador de trabajo y consulta posterior del estado.
El p99 se dispara aunque el p50 sea excelente	La fracción de arranques en frío cae dentro del percentil del SLO	Compara la fracción medida con el percentil exigido; reduce el paquete o aprovisiona solo la ruta crítica.

■ 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. Una función se invoca 500 veces por segundo y dura 400 ms. ¿Cuánta concurrencia consume y qué fracción del límite por defecto?
2. ¿Por qué la concurrencia reservada puede empeorar el rendimiento de la función que pretende proteger?
3. ¿Por qué subir maxconnections no resuelve el agotamiento de conexiones desde funciones?
4. ¿Qué límite del gateway hace imposible una integración síncrona de 2 minutos, y qué patrón lo rodea?
5. ¿Qué garantiza el modo Express de una máquina de estados y qué exige a cambio a los pasos?

Referencias

- AWS (2024). Lambda function scaling and concurrency — límites de cuenta, reserva y ritmo de crecimiento. <https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html>
- AWS (2024). Provisioned concurrency — coste y efecto sobre el arranque en frío. <https://docs.aws.amazon.com/lambda/latest/dg/provisioned-concurrency.html>
- AWS (2024). Using Amazon RDS Proxy — multiplexación, fijación de sesión y sus causas. <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/rds-proxy.html>
- AWS (2024). API Gateway quotas and important notes — tiempo de espera de 29 s y tamaño de carga útil. <https://docs.aws.amazon.com/apigateway/latest/developerguide/limits.html>
- AWS (2024). Step Functions: Standard vs Express workflows — garantías de ejecución y modelo de precio. <https://docs.aws.amazon.com/step-functions/latest/dg/concepts-standard-vs-express.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 02 en PDF · Recorrido de AWS en PDF · Manual integral

Evaluación — 032 Lambda, API Gateway y Step Functions

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 api-serverless-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.

033 — SQS, SNS y EventBridge

Parte: 02 — AWS: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: messaging · Estado: EXECUTABLECORE

Propósito

Diseñar comunicación asíncrona sabiendo qué garantiza cada servicio y qué hay que construir encima. La entrega «exactamente una vez» no existe en la red —lo estableció la clase 009— así que el receptor debe ser idempotente. Aquí se elige entre cola, tema y bus por acoplamiento y garantías, y se construye la cola de mensajes fallidos que evita perder trabajo en silencio.

Resultados de aprendizaje

Al finalizar podrás:

1. Elegir entre cola, tema y bus a partir del número de consumidores y del acoplamiento deseado.
2. Configurar el tiempo de invisibilidad coherente con la duración del procesamiento y el número de reintentos.
3. Construir una cola de mensajes fallidos con alerta y procedimiento de reproceso.
4. Explicar por qué una cola FIFO limita el rendimiento y cuándo ese coste se justifica.
5. Diagnosticar mensajes procesados varias veces distinguiendo duplicado de entrega de fallo de idempotencia.

Conceptos centrales

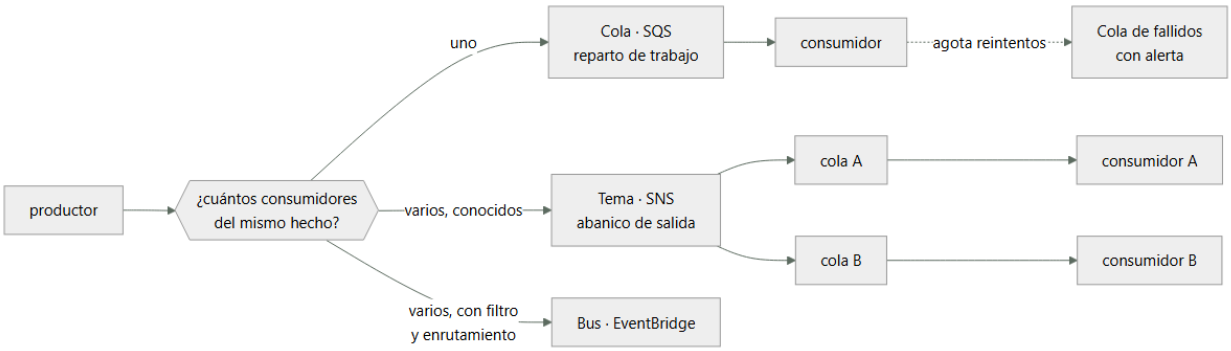
Concepto	Comprensión verificable
tiempo de invisibilidad	Periodo durante el cual un mensaje leído no es visible para otros consumidores. Si expira antes de que el consumidor termine, otro lo recibe y el mensaje se procesa dos veces.
cola de mensajes fallidos	Destino de los mensajes que agotaron sus reintentos. Sin ella, un mensaje envenenado se reintenta indefinidamente y bloquea la cola; con ella, el fallo queda aislado y visible.
abanico de salida	Patrón en el que un mensaje llega a varios consumidores independientes. Un tema lo hace en el momento; una cola, no: el mensaje lo consume uno solo.
entrega al menos una vez	Garantía de que un mensaje llegará, posiblemente más de una vez. Es lo que ofrecen las colas estándar, y obliga a que el receptor sea idempotente.
mensaje envenenado	Mensaje que provoca un fallo determinista en el consumidor. Sin cola de fallidos, se reintenta para siempre y consume capacidad que otros mensajes necesitan.

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

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. Cola, tema y bus: tres formas de desacoplar

	Cola (SQS)	Tema (SNS)	Bus (EventBridge)
Consumidores del mismo mensaje	Uno	Todos los suscritos	Los que casen con la regla
Persistencia	Hasta 14 días	Ninguna: si no hay suscriptor, se pierde	Reproducible con archivo
Filtrado	No	Por atributos	Por contenido completo
Reintentos	Del consumidor	Del suscriptor	Con cola de fallidos por regla
Latencia típica	ms	ms	0,5 s

La primera fila es la distinción fundamental: una cola reparte trabajo, un tema difunde un hecho. Si diez consumidores leen de la misma cola, cada mensaje lo procesa uno solo; eso es correcto para repartir carga y erróneo si los diez deben enterarse.

La segunda fila es la que produce pérdidas silenciosas: SNS no persiste. Si un suscriptor está caído cuando llega el mensaje, lo pierde. Por eso el patrón robusto es siempre tema seguido de cola por consumidor:

```
SNS tema "pedido-creado"
  ■■■ cola facturacion   → consumidor de facturación
  ■■■ cola inventario    → consumidor de inventario
  ■■■ cola analitica     → consumidor de analítica
```

Cada consumidor tiene su propia cola con su propio ritmo, sus propios reintentos y su propia cola de fallidos. Si el de analítica cae dos horas, sus mensajes esperan en su cola sin afectar a los demás.

EventBridge añade filtrado por contenido y enrutamiento declarativo, a cambio de más latencia:

```
{"detail-type": ["pedido-creado"], "detail": {"importe": [{"numeric": [">", 1000]}]}}
```

Esa regla entrega solo los pedidos por encima de 1.000. Con SNS habría que filtrar en el consumidor y pagar por procesar lo que se descarta.

2. El tiempo de invisibilidad es la causa de los duplicados

Cuando un consumidor lee un mensaje, este se vuelve invisible durante un tiempo configurable. Si el consumidor no lo borra antes de que expire, el mensaje vuelve a estar disponible y otro lo procesa.

```
tiempo de invisibilidad   30 s
duración del procesado    45 s      ← MAYOR

t+0   consumidor A lee el mensaje
t+30  expira la invisibilidad; el mensaje reaparece
t+30  consumidor B lo lee y empieza a procesarlo
t+45  A termina y lo borra
t+75  B termina: el efecto se aplicó DOS VECES
```

La regla de dimensionado:

```
invisibilidad ≥ duración del p99 del procesamiento × margen
```

Con un p99 de 45 s, un valor de 120 s es razonable. Y cuando la duración es muy variable, el consumidor puede extenderla mientras trabaja:

```
$ aws sqs change-message-visibility --queue-url $URL \
    --receipt-handle $RH --visibility-timeout 300
```

El efecto compuesto con el número de reintentos es el que sorprende:

```
invisibilidad 120 s, maxReceiveCount 5
→ un mensaje envenenado tarda 120 × 5 = 10 minutos en llegar a la cola de fallidos
→ durante ese tiempo ocupa capacidad de consumo 5 veces
```

Y el límite superior: la invisibilidad máxima es de 12 horas. Un procesamiento que pueda tardar más no cabe en el modelo y necesita otro diseño —un trabajo asíncrono con estado propio—.

Un último detalle que confunde al depurar: si el consumidor falla y no borra el mensaje, no hace falta esperar a la invisibilidad para reintentarlo si el consumidor la reduce a cero explícitamente al capturar el error. Eso acelera el reintento de fallos transitorios sin tocar la configuración.

3. La cola de mensajes fallidos no es opcional

Sin ella, un mensaje que provoca un fallo determinista se reintenta indefinidamente. Consume capacidad de consumo, llena los logs de errores repetidos y retrasa a los mensajes correctos que están detrás.

```
$ aws sqs set-queue-attributes --queue-url $URL --attributes '{
    "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:...:pedidos-dlq\",
                      \"maxReceiveCount\":\"5\"}",
    "VisibilityTimeout": "120",
    "MessageRetentionPeriod": "1209600"
  }'
```

Tres decisiones en esa configuración:

maxReceiveCount. Demasiado bajo (2) envía a fallidos por errores transitorios de red; demasiado alto (20) tarda horas en aislar un envenenado. Entre 3 y 5 cubre los fallos transitorios sin retrasar el aislamiento.

La retención de la cola de fallidos debe ser mayor que la de origen. Si ambas son de 14 días, un mensaje que llega a fallidos el día 13 solo se conserva un día más. La cola de fallidos debe tener 14 días contados desde su llegada, y por eso conviene revisar que el mensaje no envejezca antes de que alguien lo mire.

La alerta. Una cola de fallidos sin alerta es un cubo donde el trabajo desaparece en silencio, que es peor que no tenerla porque da falsa sensación de control:

```
$ aws cloudwatch put-metric-alarm --alarm-name pedidos-dlq-no-vacia \
    --metric-name ApproximateNumberOfMessagesVisible --namespace AWS/SQS \
    --dimensions Name=QueueName,Value=pedidos-dlq \
    --statistic Maximum --period 300 --threshold 0 \
    --comparison-operator GreaterThanThreshold --evaluation-periods 1
```

El umbral es cero: cualquier mensaje en fallidos merece atención. Y hace falta un procedimiento de reproceso escrito, porque el momento de improvisarlo no puede ser durante un incidente.

4. FIFO: qué garantiza y qué cuesta

Las colas FIFO añaden orden estricto y deduplicación, y ambas cosas se pagan:

	Estándar	FIFO
Rendimiento	Prácticamente ilimitado	300 msg/s, 3.000 con lotes
Orden	Mejor esfuerzo	Estricto dentro del grupo
Entrega	Al menos una vez	Exactamente una vez, ventana de 5 min
Alto rendimiento	—	Hasta 70.000/s con particionado por grupo

La frase clave está en la fila del orden: el orden es estricto dentro de un grupo de mensajes, no en toda la cola. El identificador de grupo es lo que decide el paralelismo:

```
grupo = "global"        → orden total, un solo consumidor efectivo, 300 msg/s
grupo = pedido_id       → orden por pedido, miles de grupos en paralelo
```

La segunda opción es casi siempre la correcta: rara vez se necesita orden global. Lo que se necesita es que los eventos de un mismo pedido lleguen en orden, y para eso el grupo debe ser el identificador del pedido.

La deduplicación tiene un límite temporal que hay que conocer: 5 minutos. Dos mensajes idénticos separados por 6 minutos se entregan ambos. Por eso la deduplicación de FIFO no sustituye a la idempotencia del receptor, solo reduce su frecuencia de activación.

La pregunta antes de elegir FIFO:

```
¿el orden importa de verdad, o solo lo parece?
  "el pago debe ir después de la reserva"  → sí, importa
  "prefiero que lleguen en orden"          → no justifica el coste
```

Muchos sistemas que creen necesitar orden lo que necesitan es idempotencia y detección de eventos fuera de orden —descartar una actualización con marca de tiempo anterior a la ya aplicada—, que escala sin límite.

5. Duplicado de entrega frente a fallo de idempotencia

Cuando aparece un efecto duplicado hay dos causas posibles y se corrigen en sitios distintos:

```
duplicado de ENTREGA      el mismo mensaje se entregó dos veces
  → normal en entrega al menos una vez
  → se corrige con idempotencia en el consumidor

fallo de IDEMPOTENCIA     el consumidor no reconoció que ya lo había procesado
  → es un defecto
  → se corrige en la lógica del consumidor
```

Distinguirlas exige registrar el identificador del mensaje:

```
def handler(event, context):
    for registro in event["Records"]:
        mid = registro["messageId"]
        veces = int(registro["attributes"]["ApproximateReceiveCount"])
        log.info("procesando", extra={"message_id": mid, "recepcion": veces})
        if ya_procesado(mid):
            log.info("duplicado descartado", extra={"message_id": mid})
            continue
        procesar(registro)
        marcar_procesado(mid)     # en la MISMA transacción que el efecto
```

El comentario de la última línea es el mismo punto de la clase 009: si marcar y aplicar el efecto no son atómicos, existe una ventana en la que el efecto ocurrió y la marca no, y el reintento duplica.

ApproximateReceiveCount es la métrica que separa los diagnósticos:

```
contador = 1 y efecto duplicado   → fallo de idempotencia: el consumidor falló
contador > 1 y efecto duplicado   → duplicado de entrega no manejado
contador > 1 y sin duplicado      → funcionando como debe
```

Y un consejo operativo: alertar cuando ApproximateReceiveCount supere 2 de forma sostenida. No es un fallo por sí mismo, pero indica que algo está reintentando más de lo normal —invisibilidad corta, consumidor lento o errores transitorios frecuentes— y precede a problemas mayores.

Ejemplo trabajado

Durante una promoción, 1.847 pedidos de CloudShop se facturan dos veces y 312 no se facturan en absoluto. Dos síntomas opuestos, causas distintas.

Síntoma 1 — facturación duplicada.

```
$ aws sqs get-queue-attributes --queue-url $URL_FACTURACION \
    --attribute-names VisibilityTimeout ApproximateNumberOfMessagesNotVisible
{"VisibilityTimeout": "30", "ApproximateNumberOfMessagesNotVisible": "412"}
```

Se mide cuánto tarda el consumidor:

```
p50   8,2 s
p95  28,4 s
p99  51,7 s      ← mayor que la invisibilidad de 30 s
```

```
invocaciones en el pico          42.000
fracción por encima de 30 s        4,4 %  → 1.848 mensajes
duplicados observados                     1.847
```

Cuadra exactamente. El 4,4 % de los mensajes tarda más que la invisibilidad, reaparece y lo procesa otro consumidor. No es un fallo del código: es un parámetro mal dimensionado.

Se comprueba si además había fallo de idempotencia:

```
$ aws logs filter-log-events --log-group-name /aws/lambda/facturacion \
    --filter-pattern '"duplicado descartado"' --start-time 1785540000000 \
    --query 'length(events)'
0
```

Cero descartes: el consumidor no comprobaba duplicados en absoluto. Los dos arreglos son necesarios y ninguno basta solo.

```
$ aws sqs set-queue-attributes --queue-url $URL_FACTURACION \
    --attributes VisibilityTimeout=180        # 3,5× el p99

# y en el consumidor, marca y efecto en la misma transacción
with db.transaction():
    if db.execute("INSERT INTO procesados(message_id) VALUES(%s) "
                  "ON CONFLICT DO NOTHING", [mid]).rowcount == 0:
        return            # ya procesado
    facturar(pedido)
```

Síntoma 2 — 312 pedidos sin facturar.

```
$ aws sqs get-queue-attributes --queue-url $URL_FACTURACION \
    --attribute-names RedrivePolicy
{}
```

No había cola de mensajes fallidos. Se reconstruye qué pasó con los logs:

```
$ aws logs filter-log-events --log-group-name /aws/lambda/facturacion \
    --filter-pattern 'ERROR' --query 'events[0].message' --output text
ERROR ValueError: importe negativo en pedido A-88213
```

312 pedidos de una promoción con descuento superior al importe generaban un valor negativo. El consumidor fallaba, el mensaje volvía a la cola, y así indefinidamente:

```
mensajes envenenados                    312
reintentos por mensaje en 4 horas    ~480   (cada 30 s)
invocaciones desperdiciadas       ~149.760
```

Además de perder esos pedidos, consumían capacidad que retrasaba a los buenos, lo que agravó el síntoma 1 al alargar los tiempos de procesamiento.

```
$ aws sqs create-queue --queue-name facturacion-dlq \
    --attributes MessageRetentionPeriod=1209600
$ aws sqs set-queue-attributes --queue-url $URL_FACTURACION --attributes '{
    "RedrivePolicy":"{\"deadLetterTargetArn\":\"arn:...:facturacion-dlq\",
                      \"maxReceiveCount\":\"4\"}"}'
$ aws cloudwatch put-metric-alarm --alarm-name facturacion-dlq-no-vacia \
    --metric-name ApproximateNumberOfMessagesVisible --namespace AWS/SQS \
    --dimensions Name=QueueName,Value=facturacion-dlq \
    --statistic Maximum --period 300 --threshold 0 \
    --comparison-operator GreaterThanThreshold --evaluation-periods 1
```

Y se reprocesan los 312 tras corregir la validación:

```
$ aws sqs start-message-move-task \
    --source-arn arn:aws:sqs:...:facturacion-dlq \
    --destination-arn arn:aws:sqs:...:facturacion
```

Resultado en la promoción siguiente:

```
pedidos duplicados                1.847         0
pedidos sin facturar                312         0
invocaciones desperdiciadas     149.760         0
p95 de la cola                   28,4 s     9,1 s
mensajes en cola de fallidos       n/a         7  (con alerta y reproceso)
```

Los 7 de la cola de fallidos son el resultado correcto: fallos reales, aislados, visibles y recuperables, en vez de trabajo que desaparece.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-02-aws-core-platform/033-sqs-sns-y-eventbridge/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 flujo-eventos-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 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 flujo-eventos-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
Un porcentaje de mensajes produce efecto duplicado	El procesamiento supera el tiempo de invisibilidad y el mensaje reaparece	Dimensiona la invisibilidad por encima del p99 del procesamiento, o extiéndela mientras trabajas.
Trabajo que desaparece sin dejar rastro	No hay cola de mensajes fallidos: los envenenados se reintentan hasta caducar	Configura cola de fallidos con maxReceiveCount entre 3 y 5 y alerta con umbral cero.
Un suscriptor pierde mensajes mientras está caído	SNS no persiste: si no hay quien reciba, el mensaje se pierde	Patrón tema seguido de cola por consumidor; cada uno con su ritmo y sus reintentos.
Una cola FIFO no supera 300 mensajes por segundo	Todos los mensajes usan el mismo identificador de grupo, así que no hay paralelismo	Usa el identificador de la entidad como grupo; el orden estricto es por grupo, no por cola.
Aparece un duplicado con ApproximateReceiveCount igual a 1	No es duplicado de entrega: es un fallo de idempotencia en el consumidor	Marca el mensaje como procesado en la misma transacción que el efecto.

■ Seguridad, ética y costo

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

■ Preguntas de comprobación

1. ¿Por qué el patrón robusto es tema seguido de cola por consumidor en vez de suscribir consumidores directamente al tema?
2. El procesamiento tiene un p99 de 50 s y la invisibilidad es de 30 s. ¿Qué fracción se duplicará y por qué?
3. Con invisibilidad de 120 s y maxReceiveCount de 5, ¿cuánto tarda un mensaje envenenado en aislarse?
4. ¿Por qué la deduplicación de FIFO no sustituye a la idempotencia del receptor?
5. Ves un efecto duplicado con ApproximateReceiveCount = 1. ¿Dónde está el fallo?

Referencias

- AWS (2024). Amazon SQS visibility timeout — semántica, extensión y límite de 12 horas. <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-visibility-timeout.html>
- AWS (2024). Amazon SQS dead-letter queues — política de reenvío y reproceso. <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-dead-letter-queues.html>
- AWS (2024). FIFO queues: message groups and throughput — orden por grupo y límites. <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/FIFO-queues.html>
- AWS (2024). EventBridge event patterns — filtrado por contenido y enrutamiento declarativo. <https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-event-patterns.html>
- Hohpe, G. y Woolf, B. (2003). Enterprise Integration Patterns — canal punto a punto, publicación-suscripción y canal de mensajes inválidos.
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 02 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 032 · Lambda, API Gateway y Step Functions	Parte 02 · Programa	034 · CloudWatch, CloudTrail, Config y Systems Manager →

Evaluación — 033 SQS, SNS y EventBridge

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 flujo-eventos-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.

034 — CloudWatch, CloudTrail, Config y Systems Manager

Parte: 02 — AWS: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: observability · Estado: EXECUTABLECORE

Propósito

Construir la evidencia operativa que permite responder tres preguntas distintas durante un incidente: qué está pasando, quién hizo qué y qué cambió. Cada una se responde con un servicio diferente, y confundirlos hace que se busque durante horas en el sitio equivocado. Aquí se aterriza en AWS lo que la clase 012 estableció sobre señales.

Resultados de aprendizaje

Al finalizar podrás:

1. Asignar cada pregunta forense al servicio que la responde y saber qué no responde ninguno.
2. Diseñar métricas y filtros que no multipliquen el coste por la dimensionalidad.
3. Consultar registros de auditoría para reconstruir quién ejecutó una acción y desde dónde.
4. Detectar desviaciones de configuración con reglas que se evalúan de forma continua.
5. Ejecutar una operación en una flota sin abrir puertos ni mantener claves de acceso.

Conceptos centrales

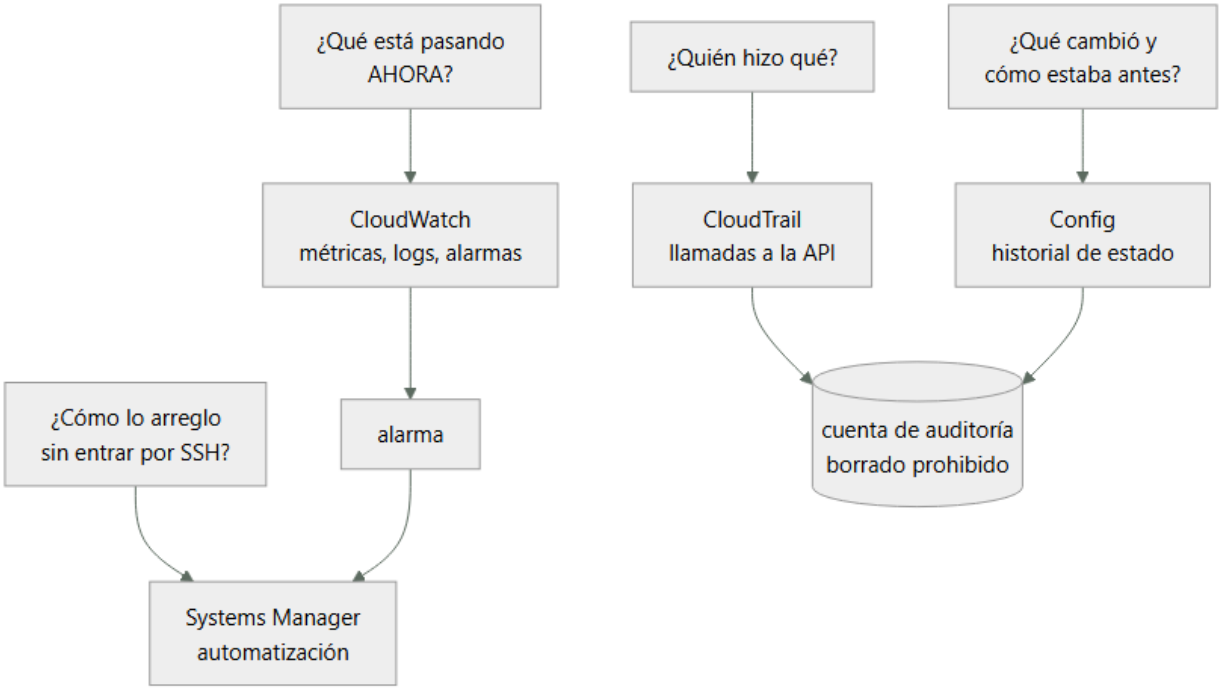
Concepto	Comprensión verificable
cardinalidad	Número de combinaciones distintas de dimensiones de una métrica. Cada combinación es una serie temporal facturada por separado, así que una dimensión con muchos valores multiplica el coste.
métrica de filtro	Métrica derivada de patrones en los logs. Convierte texto en series temporales sin instrumentar el código, a cambio de depender del formato del mensaje.
registro de auditoría	Historial de llamadas a la API con identidad, origen, parámetros y resultado. Responde «quién hizo qué», que ninguna métrica ni log de aplicación responde.
elemento de configuración	Instantánea del estado de un recurso en un momento dado. Permite comparar cómo estaba antes y después de un cambio, que es distinto de saber quién lo hizo.
documento de automatización	Procedimiento declarativo ejecutable sobre una flota, con parámetros y control de errores. Convierte un runbook escrito en uno ejecutable y auditable.

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

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

■ Flujo de razonamiento



Desarrollo

1. Cuatro preguntas, cuatro servicios

Durante un incidente se hacen preguntas distintas y cada una tiene su fuente. Buscar en la equivocada es la causa más común de que un diagnóstico tarde horas:

Pregunta	Servicio	Qué no responde
¿Qué está pasando ahora?	CloudWatch	Quién lo provocó
¿Quién hizo qué y desde dónde?	CloudTrail	Cómo estaba antes
¿Qué cambió y cuál era el estado previo?	Config	Quién lo cambió
¿Cómo actúo sobre la flota?	Systems Manager	—

Las dos filas centrales se confunden constantemente y son complementarias:

```
CloudTrail dice:  "a las 14:22, el rol ci-deploy llamó a ModifyDBInstance
                   desde 203.0.113.40 con estos parámetros"
Config dice:      "a las 14:22, la instancia pasó de BackupRetentionPeriod=7
                   a BackupRetentionPeriod=0"
```

CloudTrail da el actor y la intención; Config da el estado antes y después. Para reconstruir un incidente hacen falta los dos, y por eso ambos deben escribir en la cuenta de auditoría de la clase 025.

Y hay una pregunta que ninguno responde: por qué. Eso solo está en el ADR, el ticket o la conversación. Es la razón por la que la clase 024 insistía en registrar el razonamiento: la telemetría reconstruye el qué con precisión y nunca el motivo.

2. La cardinalidad multiplica la factura

Cada combinación distinta de dimensiones es una métrica personalizada facturada por separado, a unos 0,30 USD al mes. La aritmética se dispara rápido:

```
métrica: latencia_peticion
dimensiones: servicio (8) × endpoint (40) × código (12) × instancia (24)

series = 8 × 40 × 12 × 24 = 92.160
coste  = 92.160 × 0,30 = 27.648 USD/mes
```

Casi 28.000 dólares al mes en métricas, por añadir dos dimensiones que parecían útiles. Y la mayoría de esas series no se consulta nunca.

La dimensión culpable suele ser identificable: instancia tiene 24 valores hoy y crecerá con el autoescalado, además de generar series que mueren en cada despliegue. Un identificador efímero nunca debe ser dimensión de una métrica.

```
sin la dimensión instancia: 8 × 40 × 12 = 3.840 series → 1.152 USD/mes
agrupando códigos en clases (2xx, 4xx, 5xx): 8 × 40 × 3 = 960 → 288 USD/mes
```

Un factor de 96 respecto al diseño inicial, sin perder capacidad de diagnóstico: si hace falta saber qué instancia concreta falló, eso está en los logs, que se cobran por volumen ingerido y no por combinación.

La regla práctica:

```
métrica  → pocas dimensiones, valores acotados y estables
log      → todo el detalle, incluidos identificadores
traza    → la relación entre servicios de una petición concreta
```

Y una forma económica de obtener métricas sin instrumentar: el formato de métrica embebida, que permite emitir un log estructurado del que CloudWatch extrae métricas automáticamente, pagando la ingesta del log en vez de la publicación de cada métrica.

3. Consultar registros de auditoría para reconstruir un incidente

CloudTrail responde quién hizo qué, y la consulta directa por atributos es limitada. Para investigaciones reales conviene usar el lago de datos o consultas sobre el bucket:

```
# Búsqueda rápida por evento, últimos 90 días
$ aws cloudtrail lookup-events \
    --lookup-attributes AttributeKey=EventName,AttributeValue=DeleteBucketPolicy \
    --query 'Events[].[EventTime,Username,CloudTrailEvent]' --output text | head -3
```

Y el detalle completo del evento, que es donde está lo útil:

```
{
  "eventTime": "2026-08-01T14:22:08Z",
  "userIdentity": {
    "type": "AssumedRole",
    "arn": "arn:aws:sts::123456789012:assumed-role/ci-deploy/despliegue-4821",
    "sessionContext": {"attributes": {"mfaAuthenticated": "false"}}
  },
  "sourceIPAddress": "203.0.113.40",
  "userAgent": "aws-cli/2.15.0",
  "requestParameters": {"bucketName": "cloudshop-facturas"},
  "errorCode": null
}
```

Tres campos que suelen decidir la investigación:

- sessionContext indica si hubo MFA. Una acción sensible sin MFA es un hallazgo por sí misma.
- sourceIPAddress distingue una acción desde el pipeline de una desde un portátil.
- errorCode revela los intentos fallidos, que son la señal más útil de un ataque en curso: alguien probando permisos que no tiene.

Dos límites que conviene conocer: el historial de consulta directa cubre 90 días; más atrás hay que ir al bucket. Y los eventos de datos no se registran por defecto —lecturas de objetos, invocaciones de funciones—: si hacen falta, hay que activarlos explícitamente y su volumen puede ser enorme.

Esa segunda limitación es exactamente lo que impidió, en la brecha de la clase 010, saber cuántas veces se descargaron las facturas: los eventos de gestión estaban registrados y los de datos no.

4. Config: qué cambió y detección continua de desviaciones

Config graba el estado de cada recurso a lo largo del tiempo, lo que permite responder cómo estaba antes de un cambio:

```
$ aws configservice get-resource-config-history \
    --resource-type AWS::RDS::DBInstance --resource-id db-ABCD1234 \
    --query 'configurationItems[0:2].[configurationItemCaptureTime,
             configuration.backupRetentionPeriod]' --output text
2026-08-01T14:22:31Z    0
```

```
2026-07-15T09:11:02Z    7
```

Y sobre ese historial se evalúan reglas de forma continua:

```
$ aws configservice put-config-rule --config-rule '{
    "ConfigRuleName": "rds-retencion-minima",
    "Source": {"Owner": "AWS", "SourceIdentifier": "DB_INSTANCE_BACKUP_ENABLED"},
    "InputParameters": "{\"backupRetentionPeriod\":\"7\"}"
  }'
```

La distinción con las SCP de la clase 025 importa:

```
SCP           preventiva: impide que la acción ocurra
regla Config  detectiva: la acción ocurre y se marca como no conforme
```

Son complementarias y ninguna sustituye a la otra. Hay controles que no se pueden prevenir sin bloquear trabajo legítimo, y ahí la detección con corrección automática es la respuesta:

```
$ aws configservice put-remediation-configurations --remediation-configurations '[{
    "ConfigRuleName": "rds-retencion-minima",
    "TargetType": "SSM_DOCUMENT",
    "TargetId": "AWSConfigRemediation-ModifyRDSInstanceBackupRetention",
    "Automatic": true,
    "MaximumAutomaticAttempts": 3
  }]'
```

Una advertencia sobre el coste: Config cobra por elemento de configuración registrado. En un entorno con autoescalado agresivo, cada instancia creada y destruida genera varios elementos, y la factura puede crecer de forma inesperada. Se acota limitando los tipos de recurso grabados a los que de verdad importan.

5. Operar la flota sin puertos abiertos ni claves

Systems Manager permite ejecutar comandos y sesiones interactivas sin abrir el puerto 22, sin claves SSH y sin IP pública. El agente inicia una conexión saliente, así que la instancia puede vivir en una subred aislada —la de la clase 027—.

```
# Sesión interactiva, auditada y sin SSH
$ aws ssm start-session --target i-0a1b2c3d

# Comando en toda una flota por etiqueta
$ aws ssm send-command --document-name AWS-RunShellScript \
    --targets Key=tag:Entorno,Values=produccion \
    --parameters 'commands=["systemctl restart cloudshop"]' \
    --max-concurrency 10% --max-errors 2
```

Los dos últimos parámetros son la diferencia entre una operación y un incidente: max-concurrency limita cuántas instancias se tocan a la vez y max-errors detiene la ejecución si demasiadas fallan. Sin ellos, un comando erróneo se aplica a las 200 instancias simultáneamente.

La ventaja sobre SSH no es la comodidad, son tres propiedades:

```
1. Auditado: cada sesión y cada comando quedan en CloudTrail
2. Sin credenciales de larga vida: usa el rol de instancia
3. Sin superficie de red: ningún puerto entrante abierto
```

Y los runbooks se vuelven ejecutables en vez de escritos:

```
schemaVersion: '0.3'
parameters:
  InstanceId: {type: String}
mainSteps:
  - name: DrenarDelBalanceador
    action: aws:executeAwsApi
    inputs: {Service: elbv2, Api: DeregisterTargets, ...}
  - name: EsperarDrenado
    action: aws:sleep
    inputs: {Duration: PT90S}
  - name: Reiniciar
    action: aws:runCommand
    inputs: {DocumentName: AWS-RunShellScript, ...}
```

La diferencia con un runbook en una wiki: este se ejecuta igual a las 3 de la madrugada que en un simulacro, no depende de que alguien recuerde el orden, y su ejecución queda registrada.

Ejemplo trabajado

A las 14:31 el equipo de CloudShop recibe una alerta: la retención de copias de la base de datos de producción es cero. Se reconstruye el incidente completo con los cuatro servicios.

¿Qué está pasando? — CloudWatch dio la alerta, pero no dice más:

```
$ aws cloudwatch describe-alarms --alarm-names config-rds-retencion \
    --query 'MetricAlarms[0].[StateValue,StateUpdatedTimestamp]' --output text
ALARM   2026-08-01T14:31:12Z
```

¿Qué cambió y cómo estaba antes? — Config:

```
$ aws configservice get-resource-config-history --resource-type AWS::RDS::DBInstance \
    --resource-id db-ABCD1234 --limit 2 \
    --query 'configurationItems[].[configurationItemCaptureTime,
             configuration.backupRetentionPeriod,configuration.multiAZ]' --output text
2026-08-01T14:22:31Z    0     true
2026-07-15T09:11:02Z    7     true
```

De 7 días a 0 a las 14:22. Nueve minutos antes de la alerta.

¿Quién lo hizo? — CloudTrail:

```
$ aws cloudtrail lookup-events \
    --lookup-attributes AttributeKey=ResourceName,AttributeValue=db-ABCD1234 \
    --start-time 2026-08-01T14:00:00Z \
  | jq -r '.Events[0].CloudTrailEvent | fromjson
           | [.eventTime, .userIdentity.arn, .sourceIPAddress,
              .requestParameters.backupRetentionPeriod] | @tsv'
2026-08-01T14:22:08Z  arn:aws:sts::...:assumed-role/ci-deploy/despliegue-4821  203.0.113.40  0
```

Fue el pipeline, no una persona. Se busca el origen en el cambio de infraestructura:

```
el módulo de Terraform tenía backup_retention_period sin declarar
→ el proveedor aplicó su valor por defecto: 0
→ el plan mostraba el cambio y nadie lo leyó
```

Se comprueba si hubo más recursos afectados por la misma causa:

```
$ aws configservice select-resource-config \
    --expression "SELECT resourceId WHERE resourceType='AWS::RDS::DBInstance'
                  AND configuration.backupRetentionPeriod = 0"
{"Results": ["{\"resourceId\":\"db-ABCD1234\"}", "{\"resourceId\":\"db-EFGH5678\"}"]}
```

Dos instancias, no una. La segunda no tenía alerta porque la regla solo cubría producción.

Cuánto tiempo estuvo sin protección:

```
cambio                14:22:08
detección              14:31:12   (9 min: la regla se evalúa cada 10 min)
corrección             14:38:00
ventana sin copias      15 min 52 s
copias perdidas         ninguna: las anteriores se conservan 7 días desde su creación
```

Correcciones, separadas por tipo de control:

```
PREVENTIVO   SCP que deniega ModifyDBInstance con BackupRetentionPeriod=0
             probado con prueba negativa:
             $ aws rds modify-db-instance --backup-retention-period 0 ...
             AccessDenied ... explicit deny in a service control policy  ✓

DETECTIVO    regla de Config ampliada a todas las cuentas, no solo producción
             corrección automática que restaura 7 días

ORIGEN       el módulo de Terraform declara el valor explícitamente
             y una prueba de política rechaza el plan si es menor que 7

OPERATIVO    documento de automatización para verificar retención en la flota,
             ejecutable en simulacro
```

Y se detecta un hueco al revisar el registro: los eventos de datos no estaban activados, así que no se puede saber si alguien accedió a las copias durante la ventana. Se activan para los buckets de copias, con el coste declarado:

```
eventos de datos en 3 buckets: ~2,4 M eventos/mes × 0,10 USD/100k = 2,40 USD/mes
```

Resultado: el incidente se reconstruyó completo en 12 minutos porque cada pregunta tenía su fuente. Sin Config no se habría sabido el estado anterior; sin CloudTrail, que fue el pipeline y no una persona; y sin la consulta agregada, que había una segunda instancia afectada.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-02-aws-core-platform/034-cloudwatch-cloudtrail-config-y-systems-manager/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 evidencia-operativa-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 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 evidencia-operativa-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
La factura de métricas personalizadas es de miles de dólares	Una dimensión de alta cardinalidad multiplica las series facturadas	Saca los identificadores efímeros de las dimensiones; ese detalle va en los logs, que se cobran por volumen.
Tras un incidente no se sabe si alguien accedió a los datos	Los eventos de datos de CloudTrail no se registran por defecto	Actívalos para los recursos sensibles y asume su coste; los de gestión no cubren lecturas de objetos.
Se sabe qué cambió pero no quién, o al revés	Se consultó un solo servicio: Config da el estado y CloudTrail el actor	Usa ambos: son complementarios y ninguno responde la pregunta del otro.
Un comando erróneo se aplica a toda la flota a la vez	Se ejecutó sin limitar concurrencia ni tolerancia a errores	Usa max-concurrency y max-errors en toda ejecución sobre flota.
La factura de Config crece de forma inesperada	Se graban todos los tipos de recurso y el autoescalado genera elementos continuamente	Limita los tipos grabados a los que importan para el cumplimiento y el diagnóstico.

■ 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 servicio responde «quién lo hizo» y cuál «cómo estaba antes»? ¿Por qué hacen falta los dos?
2. Una métrica con 4 dimensiones de 8, 40, 12 y 24 valores. ¿Cuántas series genera y cuál eliminarías primero?
3. ¿Qué campo de un evento de auditoría revela un ataque en curso mejor que las acciones exitosas?
4. ¿Qué diferencia hay entre una SCP y una regla de Config, y por qué ninguna sustituye a la otra?
5. ¿Qué tres propiedades aporta una sesión gestionada frente a SSH, más allá de la comodidad?

Referencias

- AWS (2024). CloudWatch: publishing custom metrics and dimensions — cardinalidad y facturación por serie. <https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/publishingMetrics.html>
- AWS (2024). CloudWatch Embedded Metric Format — métricas desde logs estructurados. <https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatchEmbeddedMetricFormat.html>
- AWS (2024). CloudTrail: logging data events — qué se registra por defecto y qué hay que activar. <https://docs.aws.amazon.com/awscloudtrail/latest/userguide/logging-data-events-with-cloudtrail.html>
- AWS (2024). AWS Config rules and remediation — evaluación continua y corrección automática. <https://docs.aws.amazon.com/config/latest/developerguide/evaluate-config.html>
- AWS (2024). Systems Manager Session Manager — acceso sin puertos abiertos ni claves, con auditoría. <https://docs.aws.amazon.com/systems-manager/latest/userguide/session-manager.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 02 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 033 · SQS, SNS y EventBridge	Parte 02 · Programa	035 · KMS, Secrets Manager, WAF y controles de seguridad →

Evaluación — 034 CloudWatch, CloudTrail, Config y Systems Manager

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 evidencia-operativa-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.

035 — KMS, Secrets Manager, WAF y controles de seguridad

Parte: 02 — AWS: plataforma esencial Nivel: intermedio · Horas estimadas: 4 Laboratorio: security · Estado: EXECUTABLECORE

Propósito

Aplicar defensa en profundidad a la plataforma AWS: cifrado con control de claves propio, secretos que rotan sin desplegar, y filtrado en el borde. La clase 026 cubrió quién puede hacer qué; esta cubre qué ocurre cuando esa barrera falla, que es la única suposición razonable a la hora de diseñar seguridad.

Resultados de aprendizaje

Al finalizar podrás:

1. Distinguir clave gestionada por el proveedor, gestionada por el cliente y material importado, con sus consecuencias.
2. Escribir una política de clave que impida el acceso incluso a quien tenga permisos sobre el recurso cifrado.
3. Rotar un secreto sin desplegar la aplicación, entendiendo la fase de dos versiones.
4. Configurar reglas de borde que corten ataques volumétricos y de aplicación sin bloquear tráfico legítimo.
5. Verificar cada control con prueba negativa, no con inspección de la configuración.

Conceptos centrales

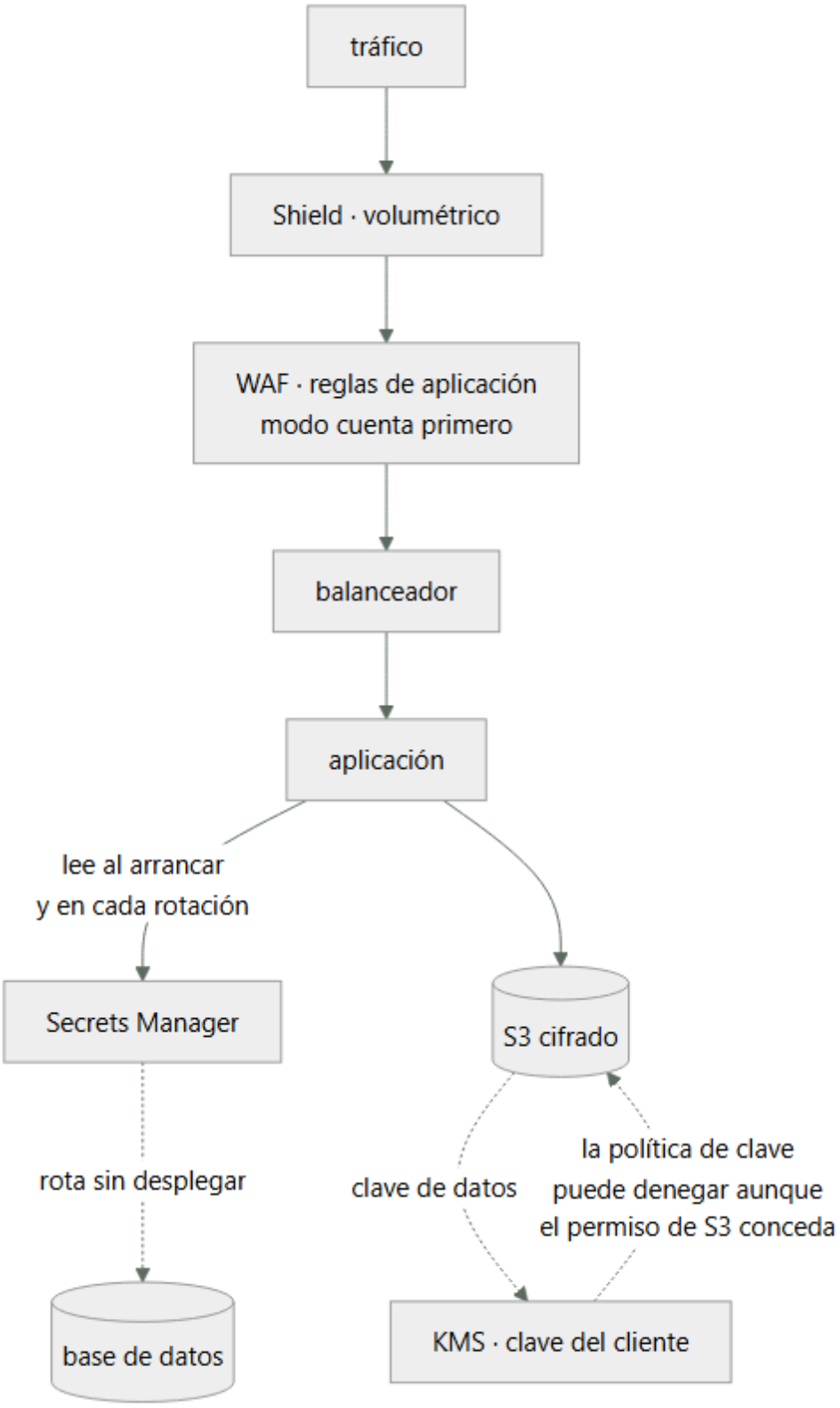
Concepto	Comprensión verificable
cifrado de sobre	Cifrar los datos con una clave de datos y esa clave con una clave maestra. Permite cifrar volúmenes grandes sin llamar al servicio de claves por cada operación y rotar la maestra sin recifrar los datos.
política de clave	Documento que gobierna quién puede usar y administrar una clave. Es independiente de los permisos sobre el recurso cifrado: sin acceso a la clave, el acceso al dato no sirve.
rotación de secretos	Sustitución periódica de una credencial sin intervención humana. Exige una fase en la que dos versiones son válidas, o la rotación corta las conexiones en curso.
regla gestionada	Conjunto de reglas de filtrado mantenido por el proveedor o un tercero. Ahorra escribirlas y exige medirlas en modo cuenta antes de bloquear, porque generan falsos positivos.
modo cuenta	Configuración en la que una regla registra las coincidencias sin bloquear. Es el único modo seguro de introducir reglas nuevas en un servicio con tráfico real.

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

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


tráfico
Shield · volumétrico
WAF · reglas de aplicación
modo cuenta primero
balanceador
aplicación
lee al arrancar
y en cada rotación
Secrets Manager
rota sin desplegar
base de datos
S3 cifrado
clave de datos
la política de clave
puede denegar aunque
el permiso de S3 conceda
KMS · clave del cliente

Desarrollo

1. Tres tipos de clave, tres niveles de control

Tipo	Quién controla	Se puede	Coste
Gestionada por AWS	AWS	Nada: no se ve ni se audita su uso por separado	Gratis
Gestionada por el cliente	Tú	Política propia, rotación, desactivación, auditoría	1 USD/mes + uso
Material importado	Tú, fuera de AWS	Todo lo anterior más control del material	1 USD/mes + uso

La diferencia práctica entre la primera y la segunda es mayor de lo que sugiere la tabla:

```
clave gestionada por AWS
  no puedes escribir su política
  no puedes desactivarla
  no puedes impedir que otro principal de la cuenta la use
  → el cifrado protege del disco físico y de poco más

clave del cliente
  su política es una segunda barrera independiente de la del recurso
  se puede desactivar y con ello hacer ilegibles los datos al instante
  su uso se audita como evento propio
```

Esa segunda barrera es lo que convierte el cifrado en un control real. Con clave del cliente, alguien con permiso de lectura sobre el bucket pero sin permiso sobre la clave no puede leer nada:

```
{
  "Sid": "Solo el rol de la aplicación descifra",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::123456789012:role/cloudshop-app"},
  "Action": ["kms:Decrypt", "kms:GenerateDataKey"],
  "Resource": "*",
  "Condition": {"StringEquals": {"kms:ViaService": "s3.sa-east-1.amazonaws.com"}}
}
```

La condición kms:ViaService añade una restricción útil: la clave solo se puede usar a través de S3, no directamente. Alguien que consiga el permiso no puede descifrar objetos que haya copiado a otro sitio.

Y el cifrado de sobre explica por qué esto no arruina el rendimiento: KMS no cifra los datos, cifra la clave de datos que sí los cifra. Un objeto de 5 GB implica una llamada a KMS, no cinco mil.

2. Rotación de claves: lo que rota y lo que no

La rotación automática de una clave de KMS crea material criptográfico nuevo y conserva el anterior:

```
los datos cifrados con el material antiguo NO se recifran
KMS guarda todas las versiones y usa la correcta al descifrar
las escrituras nuevas usan el material nuevo
```

Eso tiene dos consecuencias que sorprenden:

1. La rotación no invalida el material antiguo. Si alguien lo obtuvo, sigue sirviendo para lo cifrado antes. Rotar reduce la exposición futura, no la pasada.
2. No hay que hacer nada al rotar. No hay migración, no hay ventana, no hay recifrado. Por eso activarla no tiene coste operativo y no activarla no tiene excusa.

```
$ aws kms enable-key-rotation --key-id 1a2b3c4d --rotation-period-in-days 365
$ aws kms get-key-rotation-status --key-id 1a2b3c4d --query 'KeyRotationEnabled'
true
```

La acción que sí invalida el acceso es desactivar la clave:

```
$ aws kms disable-key --key-id 1a2b3c4d
# a partir de aquí, ningún descifrado funciona
```

Es la respuesta correcta ante un compromiso confirmado, y por eso la clave debe estar en una cuenta donde el atacante no tenga permisos. Si la clave y los datos comparten cuenta y credenciales, desactivarla no es una opción disponible durante el incidente.

Y el borrado de una clave tiene una salvaguarda deliberada: un periodo de espera de entre 7 y 30 días, irreversible después. Borrar una clave hace ilegibles todos los datos cifrados con ella, para siempre. No hay recuperación, ni por soporte. Es el equivalente criptográfico de un borrado seguro y hay que tratarlo con el mismo cuidado.

3. Rotar secretos sin cortar conexiones

Un secreto que rota mal corta las conexiones activas. La estrategia de cuatro pasos existe para evitarlo:

```
createSecret   genera la credencial nueva, la guarda como AWSPENDING
setSecret      la aplica en el servicio destino (crea el usuario nuevo)
testSecret     comprueba que funciona
finishSecret   mueve la etiqueta AWSCURRENT a la nueva versión
```

Durante los pasos 1 a 3 ambas credenciales son válidas. Esa ventana es lo que permite que las conexiones en curso terminen con la antigua mientras las nuevas usan la nueva.

La estrategia de usuario alterno lo lleva más lejos y es la recomendada para bases de datos:

```
usuario_a  activo, contraseña vigente
usuario_b  se rota este; cuando termina, se conmuta la etiqueta
siguiente rotación: se rota usuario_a
```

Así nunca se toca la credencial que la aplicación está usando en ese momento.

El error que anula todo esto: cachear el secreto indefinidamente en la aplicación.

```
# mal: se lee una vez al arrancar y nunca más
SECRETO = obtener_secreto("cloudshop/db")

# bien: caché con caducidad y reintento ante fallo de autenticación
_cache = {"valor": None, "expira": 0}
def secreto():
    if time.time() > _cache["expira"]:
        _cache.update(valor=obtener_secreto("cloudshop/db"), expira=time.time() + 300)
    return _cache["valor"]

def conectar():
    try:
        return db.connect(**secreto())
    except AuthenticationError:
        _cache["expira"] = 0          # forzar relectura: quizá rotó
        return db.connect(**secreto())
```

El bloque except es la pieza que hace la rotación transparente: si la autenticación falla, se relee el secreto antes de rendirse. Sin él, la aplicación falla hasta que alguien la reinicie, y la rotación automática se convierte en una fuente de incidentes programados.

4. Filtrado en el borde: medir antes de bloquear

Las reglas gestionadas de WAF cubren categorías conocidas —inyección SQL, cruce de sitios, entradas maliciosas— y generan falsos positivos con tráfico real. Activarlas en modo bloqueo directamente es una forma fiable de cortar usuarios legítimos.

El procedimiento seguro:

```
1. Añadir la regla en modo cuenta
2. Observar 7-14 días de tráfico real, incluido un cierre de mes
3. Revisar las coincidencias: ¿son ataques o tráfico propio?
4. Excluir las reglas concretas que producen falsos positivos
5. Pasar a bloqueo
```

El paso 3 es donde aparecen las sorpresas: un campo de texto libre donde los usuarios escriben comillas o guiones dispara reglas de inyección; una carga de fichero grande dispara reglas de tamaño de cuerpo.

```
$ aws wafv2 get-sampled-requests --web-acl-arn $ACL --rule-metric-name AWSManagedRulesCommonRuleSet \
    --scope REGIONAL --time-window StartTime=...,EndTime=... --max-items 100 \
  | jq -r '.SampledRequests[] | [.Request.URI, .RuleNameWithinRuleGroup] | @tsv' \
  | sort | uniq -c | sort -rn | head -5
```

La regla más útil y la más barata de todas no es de contenido, es de tasa:

```
{
  "Name": "limite-por-ip",
  "Priority": 1,
```

```
  "Statement": {"RateBasedStatement": {"Limit": 2000, "AggregateKeyType": "IP"}},
  "Action": {"Block": {}}
}
```

Corta la fuerza bruta y el rastreo agresivo sin necesidad de entender el contenido. Dos matices: la ventana de evaluación es de 5 minutos, así que un pico corto puede pasar; y agregando por IP se penaliza a redes con NAT compartida —una oficina entera sale por una IP—, para lo que conviene agregar por otra clave, como una cabecera de sesión.

Y la capa anterior: la protección volumétrica estándar cubre ataques de red sin coste. El nivel avanzado añade respuesta gestionada y protección de costes frente al escalado provocado por un ataque, que es un riesgo real: un ataque que no tumba el servicio pero dispara el autoescalado produce una factura, no una caída.

5. Cada control necesita su prueba negativa

Igual que en la clase 025, la configuración no demuestra el efecto. Cada control de esta clase tiene su comprobación:

```
# 1. La política de clave deniega a quien no debe
$ aws s3api get-object --bucket cloudshop-facturas --key f-1042.pdf /tmp/x \
    --profile rol-sin-permiso-de-clave
An error occurred (AccessDenied) ... not authorized to perform: kms:Decrypt   ✓

# 2. El secreto rotado sigue funcionando sin desplegar
$ aws secretsmanager rotate-secret --secret-id cloudshop/db
$ sleep 60 && curl -s -o /dev/null -w '%{http_code}\n' https://api.cloudshop.cl/health/ready
200                                                                            ✓

# 3. La regla de tasa corta el exceso
$ for i in $(seq 1 2500); do curl -s -o /dev/null https://api.cloudshop.cl/productos; done
$ curl -s -o /dev/null -w '%{http_code}\n' https://api.cloudshop.cl/productos
403                                                                            ✓

# 4. El tráfico legítimo NO se bloquea
$ curl -s -o /dev/null -w '%{http_code}\n' -X POST https://api.cloudshop.cl/opiniones \
    -d '{"texto":"El envío llegó rápido; muy bien -- lo recomiendo"}'
200                                                                            ✓
```

La cuarta es tan importante como las tres primeras y casi nunca se hace: ese texto contiene -- y ;, que las reglas de inyección marcan. Un control que bloquea tráfico legítimo es un incidente igual que uno que deja pasar un ataque, con la diferencia de que se descubre cuando los usuarios se quejan.

Y todas deben ser automáticas y periódicas. Una política de clave correcta hoy puede quedar anulada mañana por otra concesión; una exclusión de WAF puede volverse innecesaria o insuficiente cuando cambia el tráfico.

Ejemplo trabajado

Una revisión de seguridad de CloudShop encuentra tres huecos: cifrado con clave gestionada por AWS, la contraseña de la base de datos en una variable de entorno desde hace 14 meses, y ninguna protección de aplicación en el borde.

Hueco 1 — el cifrado no protegía de nada útil.

```
$ aws s3api get-bucket-encryption --bucket cloudshop-facturas \
    --query 'ServerSideEncryptionConfiguration.Rules[0].ApplyServerSideEncryptionByDefault'
{"SSEAlgorithm": "AES256"}          # clave gestionada por AWS
```

Con esa configuración, cualquiera con s3:GetObject lee los objetos: no hay segunda barrera. Se migra a clave del cliente:

```
$ aws kms create-key --description "cloudshop facturas" --policy file://politica-clave.json
$ aws s3api put-bucket-encryption --bucket cloudshop-facturas \
    --server-side-encryption-configuration '{"Rules":[{
      "ApplyServerSideEncryptionByDefault":{"SSEAlgorithm":"aws:kms",
        "KMSMasterKeyID":"arn:aws:kms:...:key/1a2b3c4d"},
      "BucketKeyEnabled":true}]}'
```

BucketKeyEnabled reduce las llamadas a KMS reutilizando una clave a nivel de bucket:

```
sin clave de bucket: 1 llamada por objeto → 8,4 M llamadas/mes × 0,03/10k = 25,20 USD
con clave de bucket: ~99 % menos          →                                 0,28 USD
```

Prueba negativa con un rol que tiene permiso sobre S3 y no sobre la clave:

```
AccessDenied ... not authorized to perform: kms:Decrypt   ✓
```

Hueco 2 — la contraseña en variable de entorno.

```
$ aws ecs describe-task-definition --task-definition cloudshop-api:41 \
    --query 'taskDefinition.containerDefinitions[0].environment[?name==`DB_PASSWORD`].name' --output
text
DB_PASSWORD
```

Una variable de entorno es visible en la definición de tarea, en los logs de despliegue y para cualquiera que pueda describirla. Se migra:

```
$ aws secretsmanager create-secret --name cloudshop/db \
    --secret-string '{"username":"cloudshop_a","password":"..."}'
$ aws secretsmanager rotate-secret --secret-id cloudshop/db \
    --rotation-lambda-arn arn:...:function:SecretsManagerRDSPostgreSQLRotationMultiUser \
    --rotation-rules AutomaticallyAfterDays=30
```

La primera rotación cortó el servicio 4 minutos. La causa:

```
SECRETO = obtener_secreto("cloudshop/db")   # leído una vez al arrancar
```

La aplicación cacheaba el secreto indefinidamente. Se corrige con caducidad y relectura ante fallo de autenticación, y se repite la prueba:

```
rotación #2:  0 s de indisponibilidad, 0 errores    ✓
```

Hueco 3 — nada en el borde. Se añaden reglas gestionadas en modo cuenta:

```
tras 10 días en modo cuenta:
  coincidencias totales                   18.412
  ataques reales (inyección, rastreo)     17.903
  FALSOS POSITIVOS                           509
```

Los 509 se concentran en un sitio:

```
$ aws wafv2 get-sampled-requests ... | jq -r '.SampledRequests[].Request.URI' \
  | sort | uniq -c | sort -rn | head -2
    486 /opiniones
     23 /soporte/adjuntar
```

Opiniones de clientes con guiones y comillas disparaban SQLiBODY. Se excluye esa regla solo para esa ruta, no globalmente:

```
scope-down statement: NOT (URI = "/opiniones")
→ la regla sigue activa en todo lo demás
```

Y se añade la regla de tasa, que resultó ser la más efectiva:

```
límite 2.000 peticiones / 5 min por IP
bloqueos en la primera semana: 41 IP
tráfico malicioso cortado: 63 % del total de bloqueos
```

Resultado, con las cuatro pruebas:

```
cifrado con segunda barrera                no        sí       kms:Decrypt denegado ✓
secreto en variable de entorno             sí        no       rotación sin caída ✓
rotación automática                        no        30 días  ejecutada 2 veces ✓
filtrado de aplicación                     no        sí       exceso bloqueado ✓
tráfico legítimo con caracteres especiales n/a       pasa     opinión con -- : 200 ✓
coste añadido                               —      +38 USD/mes
```

Los 509 falsos positivos son el dato que justifica el modo cuenta: activadas en bloqueo desde el principio, habrían impedido escribir opiniones a 486 clientes sin que nadie relacionara la causa.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-02-aws-core-platform/035-kms-secrets-manager-waf-y-controles-de-seguridad/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 controles-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 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 controles-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
El cifrado está activado y cualquiera con permiso de lectura ve los datos	Clave gestionada por AWS: no hay política propia que actúe como segunda barrera	Usa clave del cliente con política que restrinja el descifrado a los roles previstos.
La primera rotación automática corta el servicio	La aplicación lee el secreto una vez al arrancar y lo cachea indefinidamente	Caché con caducidad y relectura ante fallo de autenticación.
Activar reglas de WAF bloquea a usuarios legítimos	Se pasó a bloqueo sin medir falsos positivos con tráfico real	Modo cuenta 7-14 días, excluye reglas concretas por ruta y solo entonces bloquea.
La factura de KMS crece con el volumen de objetos	Una llamada a KMS por objeto sin clave de bucket	Activa la clave de bucket: reduce las llamadas en torno al 99 %.
Durante un compromiso no se puede revocar el acceso a los datos	La clave vive en la misma cuenta que el atacante controla	Sitúa la clave en una cuenta separada; desactivarla es la palanca de emergencia.

■ 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 aporta una clave gestionada por el cliente que una gestionada por AWS no puede aportar?
2. ¿Recifra la rotación de una clave de KMS los datos existentes? ¿Qué implica eso si el material antiguo se filtró?
3. ¿Por qué la rotación de secretos necesita una fase con dos versiones válidas?
4. ¿Qué código debe tener una aplicación para que una rotación automática le resulte transparente?

5. ¿Por qué una prueba de que el tráfico legítimo pasa es tan necesaria como la de que el ataque se bloquea?

Referencias

- AWS (2024). AWS KMS: key policies — segunda barrera independiente de los permisos sobre el recurso. <https://docs.aws.amazon.com/kms/latest/developerguide/key-policies.html>
- AWS (2024). Rotating AWS KMS keys — qué rota, qué se conserva y por qué no hay recifrado. <https://docs.aws.amazon.com/kms/latest/developerguide/rotate-keys.html>
- AWS (2024). Secrets Manager rotation — los cuatro pasos y la estrategia de usuario alterno. <https://docs.aws.amazon.com/secretsmanager/latest/userguide/rotating-secrets.html>
- AWS (2024). AWS WAF managed rule groups — modo cuenta, exclusiones y sentencias de reducción de alcance. <https://docs.aws.amazon.com/waf/latest/developerguide/waf-managed-rule-groups.html>
- AWS (2024). S3 Bucket Keys — reducción de llamadas a KMS y su efecto en el coste. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/bucket-key.html>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 02 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 034 · CloudWatch, CloudTrail, Config y Systems Manager	Parte 02 · Programa	036 · Proyecto: aplicación de tres capas en AWS →

Evaluación — 035 KMS, Secrets Manager, WAF y controles de seguridad

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 controles-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.

036 — Proyecto: aplicación de tres capas en AWS

Parte: 02 — AWS: plataforma esencial Nivel: intermedio · Horas estimadas: 8 Laboratorio: capstone · Estado: EXECUTABLECORE

Propósito

Integrar las once clases anteriores en una plataforma que funcione, se observe, falle de forma controlada y cueste lo que se decidió que costara. No es un ejercicio de montar servicios: es demostrar con evidencia que cada decisión de las clases 025 a 035 produce el efecto que se le atribuyó, incluidas las que resulten estar mal.

Resultados de aprendizaje

Al finalizar podrás:

1. Ensamblar una aplicación de tres capas con las decisiones de las once clases previas, cada una justificada.
2. Demostrar con prueba negativa que los controles de identidad, red y cifrado deniegan lo que deben.
3. Medir una línea base de latencia y coste unitario que sirva de referencia para las partes siguientes.
4. Provocar tres fallos —zona, dependencia y credencial— y documentar el comportamiento observado.
5. Entregar un paquete de evidencia que otra persona pueda verificar sin conocimiento tácito.

Conceptos centrales

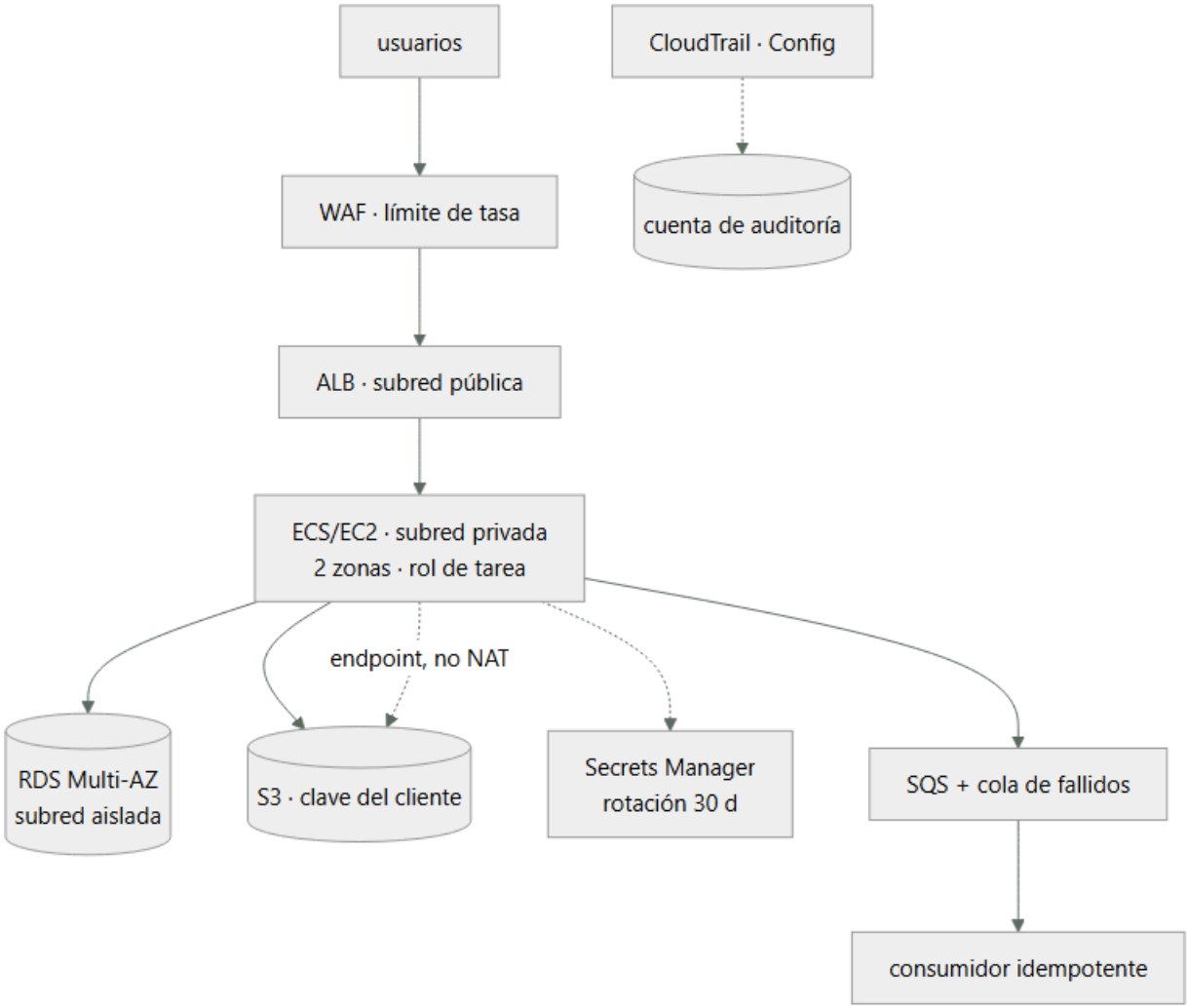
Concepto	Comprensión verificable
paquete de evidencia	Conjunto de salidas reproducibles que respaldan cada afirmación del diseño. Sustituye a «está bien configurado» por el comando ejecutado y su resultado.
prueba de fallo	Inyección deliberada de una avería para observar el comportamiento real. Un diseño resiliente que nunca se probó es una hipótesis, no una propiedad.
línea base	Medición inicial de latencia, rendimiento y coste unitario contra la que se comparará todo cambio posterior. Sin ella, «mejoró» es una opinión.
criterio de aceptación	Condición verificable que decide si el trabajo está terminado. Debe poder evaluarla alguien distinto de quien lo construyó.
riesgo residual	Lo que sigue siendo vulnerable tras aplicar los controles. Nombrarlo es parte de la entrega; omitirlo convierte un límite conocido en una sorpresa futura.

Modelo mental

Una cuenta AWS es una frontera de seguridad y facturación; los servicios se combinan mediante contratos, no como una lista de productos aislados.

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

■ Flujo de razonamiento



Desarrollo

1. La arquitectura y de dónde sale cada decisión

Nada en este capstone es una elección nueva: todo procede de una clase anterior, y esa trazabilidad es lo que hay que poder defender.

Componente	Decisión	Viene de
Cuentas separadas por entorno	Cuotas y radio de impacto	017, 025
Rol federado para el pipeline	Cero credenciales de larga vida	018, 026
Subred aislada para la base	Sin ruta a internet	027
Endpoints en vez de NAT	86 % del tráfico no va a internet	027
m7g en vez de t3	La media supera la línea base	028
Seguimiento de objetivo	Histéresis automática	016, 029
Drenado de 60 s	Mayor que la petición más larga	029
Bloqueo de objetos	El versionado no frena un borrado con permisos	010, 030
Índice compuesto	El orden estaba fuera del índice	031
Cola con fallidos e idempotencia	Entrega al menos una vez	009, 033
Clave del cliente	Segunda barrera independiente	035
Modo cuenta antes de bloquear	509 falsos positivos medidos	035

La columna derecha es el entregable real. Una arquitectura sin esa columna es una lista de servicios, y es exactamente lo que la clase 021 identificaba como diseño sin compromisos declarados.

Y hay decisiones que se toman en contra de lo que parece mejor, y también se registran: dos zonas en vez de tres porque el requisito es 21,6 min/mes y con dos ya sobran 20; RDS en vez de DynamoDB porque la agregación mensual no tiene equivalente. Justificar lo que no se hizo es tan parte de la defensa como lo que sí.

2. Cada control necesita su prueba negativa

La configuración no demuestra el efecto. El paquete de evidencia se construye ejecutando intentos que deben fallar y intentos que deben funcionar:

```
# Identidad: un repositorio ajeno no puede asumir el rol
$ (desde repo externo) aws sts assume-role-with-web-identity ...
AccessDenied                                                    ✓

# Red: la base de datos no alcanza internet
$ aws ssm start-session --target $ID_BASTION
$ curl -m 5 https://example.com
curl: (28) Connection timed out                                  ✓
$ aws secretsmanager get-secret-value --secret-id cloudshop/db --query Name
"cloudshop/db"                                                  ✓ el endpoint sí

# Cifrado: sin permiso de clave no se lee, aunque haya permiso de S3
$ aws s3api get-object --bucket cloudshop-facturas --key f.pdf /tmp/x --profile sin-kms
not authorized to perform: kms:Decrypt                           ✓

# Almacenamiento: no se puede reabrir al público
$ aws s3api put-bucket-acl --bucket cloudshop-facturas --acl public-read
AccessDenied ... blocked by the public access block              ✓

# Guardarraíl: no se puede desactivar la auditoría
$ aws cloudtrail stop-logging --name org-trail
AccessDenied ... explicit deny in a service control policy       ✓
```

Y las positivas, que son las que se olvidan:

```
$ curl -s -o /dev/null -w '%{http_code}\n' https://api.cloudshop.cl/health/ready
200                                                              ✓
$ curl -s -X POST https://api.cloudshop.cl/opiniones \
    -d '{"texto":"llegó rápido -- muy bien"}' -o /dev/null -w '%{http_code}\n'
200                                                              ✓ WAF no bloquea
```

Cinco negativas y dos positivas. Un control que deniega todo también «pasa» las cinco primeras, y solo las positivas distinguen seguridad de indisponibilidad.

3. La línea base es el entregable que más dura

Las partes 03 a 23 compararán contra estos números, así que hay que medirlos bien y guardarlos versionados.

```
$ hey -z 120s -c 50 https://api.cloudshop.cl/productos/A-1042

Summary:
  Requests/sec: 987.42
Latency distribution:
  50% in 0.0384 secs
  95% in 0.0912 secs
  99% in 0.1847 secs
Status code distribution:
  [200] 118496 responses
```

Se verifica la coherencia con la ley de Little de la clase 011:

```
L = λ × W = 987 × 0,0384 = 37,9 concurrentes
solicitados: 50 → hay holgura; la medida no está limitada por el generador  ✓
```

Si L hubiera dado 50 exactos, el número mediría el generador de carga y no el servicio.

Y el coste unitario, con el denominador de la clase 011:

```
cómputo (3 × m7g.large)             139,00
base de datos (Multi-AZ)            412,00
almacenamiento y endpoints           58,80
```

```
balanceador y WAF                     47,20
observabilidad                        31,40
                                     -------
total                                688,40

pedidos/mes                          980.000
coste unitario            688,40 / 980.000 = 0,000702 USD/pedido
```

Se registra como contrato comparable, con el escenario incluido:

```
{"escenario":"productos-lectura","concurrencia":50,"duracion_s":120,
 "rps":987.4,"p50_ms":38.4,"p95_ms":91.2,"p99_ms":184.7,"errores":0,
 "coste_mensual_usd":688.40,"coste_unitario_usd":0.000702,
 "zonas":2,"instancias":3,"fecha":"2026-08-01"}
```

La relación p99/p50 = 4,8× es la métrica a vigilar entre versiones: si crece, la cola se alarga aunque la media no se mueva.

4. Tres fallos provocados y lo que enseña cada uno

Fallo 1 — pérdida de una zona.

```
$ aws ec2 modify-network-acl-entry --network-acl-id $ACL_ZONA_B \
    --rule-number 100 --egress --protocol -1 --rule-action deny --cidr-block 0.0.0.0/0
```

```
t+0    se aísla la zona B
t+31   el balanceador marca insanos sus destinos (10 s × 3)
t+31   el tráfico se concentra en la zona A
t+34   RDS conmuta a la réplica de la zona A
t+96   el autoescalado añade 2 instancias en A
p95 durante la ventana:  91 ms → 214 ms → 97 ms
errores:  0
```

Los 31 segundos de detección son el coste de los parámetros de la clase 029. Bajarlos aumentaría los falsos positivos.

Fallo 2 — dependencia opcional degradada. No caída, sino lenta, que es el caso que rompe sistemas:

```
$ aws ssm send-command --targets Key=tag:Servicio,Values=recomendaciones \
    --document-name AWS-RunShellScript \
    --parameters 'commands=["tc qdisc add dev eth0 root netem delay 800ms"]'
```

```
p95 sin degradación   91,2 ms
p95 con la dependencia a 800 ms   94,7 ms
```

El plazo de 150 ms corta la llamada. Sin él, cada petición ocuparía un trabajador 800 ms y por la ley de Little la concurrencia necesaria se multiplicaría por ocho.

Fallo 3 — credencial comprometida. Se simula qué obtiene un atacante con el rol de tarea:

```
$ aws iam list-users                    AccessDenied     ✓
$ aws rds delete-db-instance ...        AccessDenied     ✓
$ aws s3 rm s3://cloudshop-facturas/... AccessDenied     ✓ (bloqueo de objetos)
$ aws s3 ls s3://cloudshop-facturas/    lista objetos    ← SÍ puede
$ aws s3 cp s3://cloudshop-facturas/f-1042.pdf .   descarga    ← SÍ puede
```

El rol puede leer y exfiltrar las facturas. No es un fallo de configuración: es el permiso que la aplicación necesita. El control que queda es detectivo —alerta por volumen anómalo de lecturas— y hay que declararlo como riesgo residual, no ocultarlo.

5. La entrega: sin conocimiento tácito

El criterio de aceptación no es que funcione, sino que otra persona lo verifique sin preguntarte nada.

```
capstone-aws/
├── README.md          qué es, cómo se despliega, cómo se comprueba, cómo se retira
├── infra/             la infraestructura declarada, no clicada
├── evidencia/
│   ├── negativas.md   los 5 intentos que deben fallar, con su salida
│   ├── positivas.md   los 2 que deben funcionar
│   ├── baseline.json  la línea base con su escenario
│   ├── fallo-zona.md  cronología medida
│   ├── fallo-dep.md   p95 con y sin degradación
│   └── fallo-cred.md  qué obtiene un atacante con el rol de tarea
├── adr/               las decisiones con sus alternativas descartadas
└── verify.sh          ejecuta todas las comprobaciones y sale != 0 si alguna falla
```

verify.sh es la pieza que convierte la evidencia en algo vivo:

```
$ ./verify.sh
✓ rol federado deniega desde repositorio ajeno
✓ base de datos sin salida a internet
✓ objeto no legible sin permiso de clave
✓ bucket no reabrible al público
✓ CloudTrail no desactivable
✓ salud del servicio: 200
✓ opinión con caracteres especiales: 200
7/7 comprobaciones correctas
```

Un script que imprime «OK» y siempre sale 0 no verifica nada: el código de salida es el contrato, como en la clase 012. Y con él, estas mismas comprobaciones se ejecutan en CI en la parte 08 sin cambiar una línea.

Riesgos residuales que se entregan declarados:

```
1. El rol de tarea puede leer y exfiltrar facturas. Mitigación detectiva
   (alerta por volumen), no preventiva. Aceptado por el responsable de datos.
2. No sobrevive a un fallo regional completo. Aceptado: no es requisito hoy.
3. La línea base se midió con datos sintéticos; la distribución real de
   tamaños de pedido puede diferir.
4. El WAF excluye la regla de inyección en /opiniones. Compensado con
   validación y consultas parametrizadas en esa ruta.
```

Esa lista es la parte más valiosa de la entrega. Un capstone sin riesgos residuales declarados no es que no los tenga: es que no los buscó.

Ejemplo trabajado

Entrega del capstone de la parte 02, con las cifras que se llevan a la parte 03.

Verificación completa:

```
$ ./verify.sh
✓ rol federado deniega desde repositorio ajeno       AccessDenied
✓ base de datos sin salida a internet                timeout a los 5 s
✓ objeto no legible sin permiso de clave             kms:Decrypt denegado
✓ bucket no reabrible al público                     bloqueado
✓ CloudTrail no desactivable                         deny de SCP
✓ salud del servicio                                 200
✓ opinión con " -- " y ";"                           200
7/7 correctas
```

Línea base:

```
rps 987,4 · p50 38,4 ms · p95 91,2 ms · p99 184,7 ms · errores 0
L = 987 × 0,0384 = 37,9 de 50 solicitados → medida válida
p99/p50 = 4,8×
coste 688,40 USD/mes · 0,000702 USD/pedido
```

Los tres fallos, con lo aprendido en cada uno:

```
zona aislada               31 s        214 ms        0       el coste de detección
                                                               son los parámetros de salud
dependencia a 800 ms        —          94,7 ms       0       el plazo hizo el trabajo;
                                                               sin él, ×8 de concurrencia
credencial comprometida     —            —           —       lee facturas: riesgo residual
```

Un hallazgo que la prueba de fallo destapó y la configuración no. Durante el aislamiento de la zona B:

```
$ aws logs filter-log-events --log-group-name /aws/ecs/cloudshop \
    --filter-pattern 'ERROR' --start-time ... --query 'length(events)'
0
$ aws sqs get-queue-attributes --queue-url $URL_DLQ \
    --attribute-names ApproximateNumberOfMessagesVisible
{"ApproximateNumberOfMessagesVisible": "14"}
```

14 mensajes en la cola de fallidos sin ningún error en los logs. El consumidor vivía solo en la zona B; al aislarla, sus mensajes agotaron reintentos mientras el servicio HTTP —que sí estaba en dos zonas— seguía en verde.

```
síntoma observable   ninguno: cero errores, p95 aceptable, alarmas en verde
consecuencia real    14 pedidos sin facturar
causa                el consumidor asíncrono no estaba en dos zonas
```

Se corrige y se añade lo que faltaba:

```
1. consumidor desplegado en dos zonas
2. alerta sobre la cola de fallidos con umbral cero (clase 033)
3. la prueba de fallo de zona pasa a incluir la ruta asíncrona,
   no solo la síncrona
```

El valor del capstone fue este hallazgo. Todo estaba correctamente configurado, todas las pruebas negativas pasaban, y había un camino —el asíncrono— que nadie había probado porque el fallo no era visible desde fuera.

Se entrega a la parte 03 con:

```
línea base       para comparar la misma aplicación en Azure
ADR              9 decisiones con sus alternativas descartadas
4 riesgos residuales declarados y aceptados por su responsable
verify.sh        reutilizable, con código de salida
```

Y la pregunta que abre la parte siguiente: de estas nueve decisiones, ¿cuáles son de arquitectura y cuáles son de proveedor? Las primeras deberían reaparecer en Azure con otro nombre; las segundas, no reaparecer en absoluto.

Laboratorio guiado

Ejecuta desde la raíz:

```
python classes/part-02-aws-core-platform/036-proyecto-aplicacion-de-tres-capas-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 plataforma-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 plataforma-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
El capstone es una lista de servicios desplegados	Falta la trazabilidad de cada decisión a la clase y al requisito que la motivó	Cada componente debe citar qué requisito satisface y qué alternativa se descartó.
Todas las pruebas negativas pasan y el sistema no funciona	Un control que deniega todo también las pasa; faltan las pruebas positivas	Empareja cada prueba negativa con una positiva del acceso legítimo.
Un fallo de zona deja trabajo asíncrono sin procesar y nada lo detecta	Solo se probó el camino síncrono, que sí estaba en dos zonas	Incluye colas y consumidores en la prueba de fallo, y alerta sobre la cola de fallidos.
No se puede afirmar si un cambio posterior mejoró o empeoró	No hay línea base con escenario, percentiles y coste unitario	Registra la medición versionada junto al código, con la carga y la duración usadas.
La entrega depende de que su autor explique algo	Conocimiento tácito no escrito y verificación no automatizada	Exige que otra persona ejecute verify.sh sin ayuda; lo que pregunte es lo que falta en el README.

■ 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. Elige tres componentes de tu capstone y di de qué clase y de qué requisito procede cada decisión.
2. ¿Por qué cinco pruebas negativas correctas no demuestran que el sistema sea seguro Y usable?
3. Tu medición da L = 50 con concurrencia solicitada de 50. ¿Qué estás midiendo en realidad?
4. Durante un fallo de zona no hay errores ni alarmas, pero se pierden pedidos. ¿Dónde buscarías?
5. De las decisiones de tu capstone, ¿cuáles esperas volver a tomar en Azure y cuáles no reaparecerán?

Referencias

- AWS (2024). Well-Architected Framework: the review process — cómo estructurar la revisión de una carga. <https://docs.aws.amazon.com/wellarchitected/latest/framework/the-review-process.html>
- AWS (2024). Fault Injection Service — inyección controlada de fallos de zona, red y recursos. <https://docs.aws.amazon.com/fis/latest/userguide/what-is.html>
- Beyer, B. et al., eds. (2018). The Site Reliability Workbook, cap. 5 — pruebas de fiabilidad y game days. <https://sre.google/workbook/testing-reliability/>
- Nygard, M. (2018). Release It!, 2.ª ed., cap. 5 — plazos, mamparos y degradación bajo dependencia lenta.
- Nygard, M. (2011). Documenting Architecture Decisions — formato de los ADR que acompañan la entrega. <https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions>
- Documentación oficial vigente del servicio implementado; registra URL y fecha de consulta.

Descargar: Parte 02 en PDF · Recorrido de AWS en PDF · Manual integral

Anterior	Índice	Siguiente
← 035 · KMS, Secrets Manager, WAF y controles de seguridad	Parte 02 · Programa	037 · Tenant, management groups, suscripciones y resource groups →

Evaluación — 036 Proyecto: aplicación de tres capas 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 plataforma-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.