

FLASHCARDS

Microservicios

206 tarjetas de repaso rápido

frente = pregunta, dorso = respuesta · numeradas · 10 subtemas



ESTRELLA = pregunta clave de examen

Microservicios · 206

abajo a la izquierda; el mazo viene ordenado de lo BÁSICO a lo AVANZADO.

CÓMO USARLO

1 · IMPRIMIR

A4 a doble cara, volteando por el BORDE LARGO, escala 100% (sin "ajustar a página"). Corta por las líneas discontinuas: cada pregunta queda con su respuesta al dorso, verificable por la numeración.

2 · CELULAR

Abre flashcards.html en el navegador (sin internet): toca para voltear, marca «La sé» o «Repasar» y el progreso queda guardado. Incluye búsqueda, filtros por subtema, modo aleatorio y lista completa.

3 · ANKI / QUIZLET

Importa cards.csv (Frente ; Reverso ; Categoría ; Subtema) para estudiar con repetición espaciada.

MUESTRA A ESCALA REDUCIDA

FRENTE (página impar)

REVERSO (página par, espejado)

MICROSERVICIOS

M005

Señales de que NO deberías usar microservicios

MICROSERVICIOS

M005

RESPUESTA

Equipo pequeño (1-5 devs), dominio sin explorar (no sabes los bounded contexts), startup buscando product-market fit, operaciones sin madurez (sin CI/CD, monitoreo, on-call).
Alternativa: MONOLITO MODULAR (módulos con fronteras claras, un despliegue) — migrar a micros después es más fácil que deshacer micros mal cortados.

Al imprimir a doble cara, cada respuesta queda exactamente detrás de su pregunta; la numeración lo verifica.

Contenido: 206 tarjetas + chuleta exprés al dorso de esta portada · 54 páginas · generado 12/09/2026

CHULETA EXPRÉS

Lo básico que siempre se pregunta: HTTP, seguridad, testing, datos, Docker/K8s y GCP.

- Verbos HTTP: GET leer · POST crear · PUT reemplaza · PATCH parcial · DELETE borra. Idempotentes: GET/PUT/DELETE. Seguro: GET.
- Códigos HTTP: 200 OK · 201 Created (+Location) · 204 sin body · 400 validación · 401 no autenticado · 403 sin permiso · 404 no existe · 409 conflicto · 429 rate limit · 500 bug · 503 caído.
- ACID: Atomicidad, Consistencia, Aislamiento, Durabilidad. CAP: ante partición (P) eliges consistencia (C) o disponibilidad (A).
- Escalera REST (Richardson): 0 RPC/POX · 1 recursos · 2 verbos + códigos HTTP · 3 HATEOAS.
- JWT: header.payload.signature; firma HS256 (simétrica) o RS256 (asimétrica). Claims clave: sub, iss, aud, exp, iat. Nunca guardar datos sensibles: es legible (base64).
- OAuth2: Authorization Code + PKCE para apps/SPA · Client Credentials para servicio-a-servicio · Refresh Token para renovar sin re-login.
- SQL rápido: índices B-tree aceleran WHERE/JOIN · EXPLAIN para ver el plan · N+1 se resuelve con JOIN FETCH / batch · paginar con keyset, no solo OFFSET.
- Concurrencia en BD: optimista = @Version + retry (mejor para poca contención) · pesimista = SELECT FOR UPDATE (contención alta).
- Pool de conexiones: HikariCP (default en Boot). Ajustar maximumPoolSize (10-20 suele bastar) y connectionTimeout.
- Caché HTTP: Cache-Control (max-age, no-cache) + ETag/If-None-Match → 304 Not Modified. Caché de app: Spring @Cacheable.
- JUnit 5: @Test, @BeforeEach, AssertJ assertThat. Mockito: @Mock + @InjectMocks. Tipos de test Spring: @WebMvcTest (capa web), @DataJpaTest (repositorios), @SpringBootTest (integración completa).
- TDD: rojo (test que falla) → verde (código mínimo) → refactor. Pirámide: muchos unit, algunos integration, pocos E2E.
- SOLID: S una razón de cambio por clase · O abierto a extensión, cerrado a modificación · L subtipos sustituibles · I interfaces pequeñas · D depender de abstracciones.
- GoF que salen siempre: Singleton, Factory Method, Builder, Strategy, Adapter, Observer, Decorator, Proxy (Spring AOP usa proxies dinámicos).
- Maven: coordenadas groupId:artifactId:version · scopes: compile, provided, runtime, test · BOM para alinear versiones · mvnw wrapper para versionar Maven.
- Docker: imagen inmutable por capas con cache · multi-stage build para imagen liviana · no correr como root · ENTRYPOINT vs CMD · healthcheck.
- Kubernetes: Pod (unidad mínima) · Deployment (réplicas + rolling update) · Service (IP estable/DNS) · Ingress (rutas HTTP) · HPA (autoescalado) · ConfigMap/Secret (config).
- Logging: niveles TRACE < DEBUG < INFO < WARN < ERROR. Estructurado (JSON) y con correlation ID. Nunca datos personales ni secretos en logs.
- CORS: el navegador lanza preflight OPTIONS; el servidor responde con Access-Control-Allow-Origin/Methods/Headers. Es un mecanismo del navegador, no del servidor.
- Git/entrega: feature branches + Pull Request · squash merge · Conventional Commits (feat/fix/chore) · SemVer MAYOR.MINOR.PATCH (mayor = rompe compatibilidad).
- Rate limiting: 429 Too Many Requests + header Retry-After. Algoritmos: token bucket, sliding window. Se aplica en gateway.
- Async en Spring: @Async + @EnableAsync (executor), @EventListener para eventos en proceso, CompletableFuture para composición.
- Programación reactiva (WebFlux): Mono (0..1) y Flux (0..N), backpressure, no bloquea hilos; útil con mucha I/O concurrente. MVC por hilos sigue siendo válido para CRUD normal.
- Microservicios: contrato primero (OpenAPI), consumer-driven contract tests (Pact), CI/CD por servicio, feature flags para desacoplar deploy de release.
- En la nube: 12-factor (config en entorno, procesos stateless, logs como stream), probes de salud, escalado horizontal primero, idempotencia en todo endpoint que escribe.
- K8s probes: liveness reinicia el pod, readiness lo saca del balanceo, startup da margen a arranques lentos. Requests/limits de CPU-memoria definen la calidad de servicio.
- JOINs SQL: INNER (solo coincidencias) · LEFT (toda la izquierda + coincidencias) · RIGHT · FULL. Subconsultas correlacionadas fila a fila = sospechar de N+1.
- HTTP/2: multiplexa streams en una conexión, comprime headers (HPACK) y permite prioridades; gRPC exige HTTP/2. HTTP/3 va sobre QUIC/UDP.
- Complejidad: HashMap get/put O(1) medio · búsqueda binaria O(log n) · ordenar O(n log n) · recorrer O(n). ArrayList contains O(n).
- Colecciones thread-safe: ConcurrentHashMap (no bloquea el mapa entero) · CopyOnWriteArrayList (muchas lecturas) · BlockingQueue (productor/consumidor).
- Bean Validation: @Valid en el parámetro + @NotNull/@Size/@Email/@Pattern en el DTO; si falla, MethodArgumentNotValidException → 400 con detalles.
- Clientes REST en Spring: RestClient (síncrono, moderno) · WebClient (reactivo) · OpenFeign (declarativo por interfaz). Configura siempre timeout y retry.
- Docker Compose levanta el stack local (app + BD + cola) con docker compose up; perfiles para servicios opcionales. En prod: Kubernetes o Cloud Run.
- GCP rápido: proyecto = unidad de facturación/IAM · región (multi-zona) vs zona · Cloud Run = contenedor serverless · Pub/Sub = at-least-once · IAM = least privilege · ADC para autenticar apps sin claves.

MICROSERVICIOS★

Fundamentos y decisiones

M001

¿Monolito o microservicios?

BASICO

MICROSERVICIOS★

Fundamentos y decisiones

M002

¿Cómo divides en microservicios? (Bounded Context)

INTERMEDIO

MICROSERVICIOS★

Fundamentos y decisiones

M003

Los 12 factores (12-factor app)

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M004

¿Qué NO es un microservicio? (desmontando mitos)

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M005

Señales de que NO deberías usar microservicios

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M006

Monolito modular: el paso intermedio que casi nadie hace

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M007

Ley de Conway aplicada: tu arquitectura imita tu organización

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M008

Event Storming: qué es y qué produce

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS



Fundamentos y decisiones

M002

RESPUESTA

Por dominio de NEGOCIO, no por capas técnicas (DDD): cada Bounded Context (Pedidos, Pagos, Inventario) tiene su modelo y lenguaje; el servicio posee sus datos.
Señales: cambios que van juntos, equipos distintos, escalado distinto.
Anti-patrón: nanoservicios CRUD que comparten BD.

INTERMEDIO

MICROSERVICIOS



Fundamentos y decisiones

M001

RESPUESTA

Monolito: un solo deployable; dev/debug/transacciones simples; escala todo junto y acopla módulos.
Microservicios: servicios pequeños, desplegables y escalables por separado; equipos independientes.
Costo: complejidad distribuida (red, consistencia eventual, observabilidad).
Empieza modular; divide cuando duela.

BASICO

MICROSERVICIOS

Fundamentos y decisiones

M004

RESPUESTA

NO es: un endpoint suelto (eso es una función), un CRUD con su propia BD compartida, un módulo desplegado aparte que necesita despliegues sincronizados, 'código más chico' por defecto.
ES: un servicio con negocio encapsulado, datos propios y despliegue independiente.
El tamaño correcto = cabe en un equipo y cambia por razones propias.

INTERMEDIO

MICROSERVICIOS



Fundamentos y decisiones

M003

RESPUESTA

Manifiesto cloud-native:
1 base de código por servicio · 2 dependencias explícitas · 3 config en variables de entorno · 4 backends como recursos adjuntos · 5 build/release/run separados · 6 procesos stateless · 7 port binding · 8 concurrencia por procesos · 9 descartabilidad · 10 paridad dev/prod · 11 logs como stream · 12 tareas admin como procesos.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M006

RESPUESTA

Un deployable, pero módulos con fronteras ENFORZADAS: paquetes cerrados, ArchUnit/Module checks, interfaces internas, BD con esquemas por módulo (sin joins cruzados).
Beneficio: la simplicidad del monolito + la disciplina de fronteras → extraer micros después es mecánico.
Es la recomendación default de Martin Fowler ('MonolithFirst').

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M005

RESPUESTA

Equipo pequeño (1-5 devs), dominio sin explorar (no sabes los bounded contexts), startup buscando product-market fit, operaciones sin madurez (sin CI/CD, monitoreo, on-call).
Alternativa: MONOLITO MODULAR (módulos con fronteras claras, un despliegue) — migrar a micros después es más fácil que deshacer micros mal cortados.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M008

RESPUESTA

Taller de discovery (Big Picture): pegas eventos de dominio en naranja en una pared (PedidoCreado, PagoConfirmado), luego actores, comandos, políticas, bounded contexts emergen visualmente.
Produce: modelo del dominio compartido, candidatos a microservicios (contextos), lenguaje común.
Barato y rápido comparado con descubrir el dominio en producción.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M007

RESPUESTA

Los sistemas reflejan los canales de comunicación de los equipos que los construyen.
Quieres microservicios por dominio? Organiza equipos por dominio (Team Topologies: stream-aligned teams con su servicio end-to-end).
Si 5 equipos tocan el mismo servicio, no importa cómo lo dividas: será un monolito distribuido.
Conway's Law inversa: organiza como quieres que sea la arquitectura.

INTERMEDIO

MICROSERVICIOS★

Fundamentos y decisiones

M009

Strangler Fig: migrar el monolito sin big bang

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M010

Distributed monolith: los síntomas del fracaso

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M011

ADR: Architecture Decision Records

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M012

Service ownership: qué significa end-to-end

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M013

Métricas DORA: las 4 que miden entrega de software

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M014

¿Cuántos microservicios son demasiados?

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M015

C4 model: documentar arquitectura en 4 niveles

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M016

API-first: qué significa de verdad

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Fundamentos y decisiones

M010

RESPUESTA

Desplegar un servicio requiere desplegar 5 (release trains), cambios que cruzan varios repos, servicios sin datos propios (todos contra la misma BD), latencias encadenadas para una operación simple, un fallo tira todo.
Causa raíz: cortaste por CAPAS TÉCNICAS (frontend-service-dao) o por conveniencia, no por dominio.
Remedio: re-cortar contextos, o retroceder a monolito modular.

INTERMEDIO

MICROSERVICIOS



Fundamentos y decisiones

M009

RESPUESTA

El gateway enruta: nuevas capacidades van a servicios NUEVOS; el monolito sigue sirviendo lo demás; cada ruta migrada se 'estrangula' del monolito hasta que muere.
Pasos: 1) gateway delante, 2) extraer funcionalidad + datos (CDC para sincronizar), 3) mover ruta, 4) repetir.
Nunca freeze del monolito: sigue evolucionando durante la migración.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M012

RESPUESTA

El equipo que construye el servicio lo OPERA: on-call, SLOs, costos, deprecación — 'you build it, you run it'.
Herramientas: catálogo de servicios (Backstage), runbooks, dashboards por servicio.
Anti-patrón: equipo de 'deploy' separado que recibe tickets de otros equipos para publicar.
Métricas DORA para medir la salud de la entrega.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M011

RESPUESTA

Documento corto por decisión: contexto, opciones consideradas, decisión, consecuencias (positivas y negativas), fecha y estado.
Versionado en el repo del servicio.
Valor: 6 meses después, '¿por qué usamos Saga orquestada y no coreografiada?' tiene respuesta escrita, no arqueología de Slack.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M014

RESPUESTA

Señales de exceso: nanoservicios CRUD que solo reenvían llamadas, servicios que SIEMPRE se despliegan juntos, equipos dedicados a 'concatenar' otros servicios, latencia por saltos que domina el P99.
Regla empírica: 1 equipo (5-9 personas) = 1-3 servicios que posee end-to-end.
Consolidar es una decisión arquitectónica tan legítima como dividir.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M013

RESPUESTA

Deployment frequency (cada cuánto despliegas), Lead time for changes (commit→prod), Change failure rate (% de deploys que causan incidente), MTTR (tiempo para recuperar).
Elite: deploy diario+, lead time < 1 día, CFR < 15%, MTTR < 1 hora.
Los microservicios bien hechos mueven estas 4; mal hechos las empeoran (medir antes/después).

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M016

RESPUESTA

El CONTRATO (OpenAPI/Proto) se diseña, revisa y aprueba ANTES del código — consumidores (frontend, otros equipos) empiezan en paralelo con mocks.
Validación: spec lint (Spectral), mock server, contract tests.
Beneficio: cambios caros se discuten en diseño, no en integración.
Herramientas: OpenAPI Generator, Prism, Spectral rules.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M015

RESPUESTA

Context (tu sistema + usuarios + sistemas externos), Containers (apps/servicios y sus tecnologías), Components (módulos dentro de un container), Code (clases — opcional).
Diagramas como código (PlantUML/Structurizr) versionados con ADRs.
Evita el diagrama PowerPoint de hace 2 años: C4 + CI mantiene docs vivas.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M017

Domain services vs application services

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M018

Anticorruption Layer: cuándo y cómo

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M019

¿Qué es un 'golden path' / platform engineering?

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M020

Cuándo EXTRAER un servicio del monolito: señales concretas

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M021

Product over project: el cambio de mentalidad

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M022

«¿Un microservicio puede llamar a otro directamente?» — la respuesta completa

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M023

Escenario: «cada despliegue de X rompe Y» — diagnóstico arquitectónico

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M024

Escenario: «el P99 explotó tras mover a microservicios» — causas top

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Fundamentos y decisiones

M018

RESPUESTA

Cuando consumiste un sistema EXTERNO con modelo ajeno/feo (ERP legado, API de tercero): capa que traduce su modelo al tuyo en la FRONTERA (adapter + translator).

Sin ACL: el modelo del ERP (campos raros, estados en mayúsculas) se contagia a todo tu dominio.

Costo: más código; valor: tu dominio queda limpio y el swap del externo es local.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M017

RESPUESTA

Domain service: lógica de negocio que no pertenece a un solo agregado (calcularTarifa(remate, destino)).

Application service: orquesta CASO DE USO — carga agregados, llama dominio, persiste, publica eventos, maneja tx (el 'service' de Spring suele ser esto).

Confundirlos = lógica de negocio fugándose a capas técnicas (anemia).

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M020

RESPUESTA

1) Un módulo escala de forma diferente (picos aislados).

2) Ritmos de release chocan (equipo X frena a Y).

3) Tecnología distinta necesaria (ML en Python, streaming).

4) Compliance aísla datos.

NO extraer por: 'está feo el código' (refactor), moda, o porque el dominio no está claro.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M019

RESPUESTA

Plataforma interna que da a los equipos el camino pavimentado: template de servicio (CI/CD, observabilidad, seguridad incluidos), catálogo, infraestructura self-service.

El equipo de producto codifica negocio, no YAML.

Medida del éxito: tiempo de 'idea a producción' de un nuevo servicio en días, no semanas.

Antipatrón: plataforma obligatoria sin feedback de los equipos.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M022

RESPUESTA

Sí, con condiciones: relación cliente-proveedor estable (contract tests), timeouts+breakers (no cascadas), deadline propagado, y control de la cadena (A→B→C→D de más de 2 saltos = smell — usar eventos o agregación).

Prohibido: llamadas circulares (A llama B que llama A).

Alternativa siempre presente: async por eventos si el llamador no necesita respuesta inmediata.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M021

RESPUESTA

Proyecto: equipo se arma, entrega, se desarma → el servicio queda huérfano.

Producto: equipo estable posee el servicio y su roadmap continuo.

En microservicios es crítico: cada servicio necesita dueño permanente (on-call, SLOs, evolución).

Si tu organización solo sabe hacer proyectos, microservicios producirán servicios abandonados.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M024

RESPUESTA

1) N llamadas encadenadas que antes eran 1 (in-process → network): arreglar con agregación/eventos/caché. 2) Serialización JSON duplicada: protobuf/caché. 3) Timeouts mal puestos que esperan el peor caso. 4) Falta de paralelismo en scatter-gather. 5) Cold calls sin connection pooling.

Medir con tracing el waterfall — el número indica el tramo culpable, no se adivina.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M023

RESPUESTA

Síntoma de acoplamiento: contrato compartido implícito (schema de BD o payload SIN versionar), o dependencia síncrona dura.

Remedios: contrato explícito + versionado, consumidor tolerante (ignora campos nuevos), migrar la dependencia dura a eventos, y contract testing que ROMPA el CI antes que producción.

Si persiste: los dos 'servicios' son uno — unificar o cortar bien.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M025

Escenario: «necesitamos un reporte que cruza 6 servicios» — opciones

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M026

Escenario: «dos equipos editan el mismo servicio» — soluciones

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M027

Escenario: «migrar 200 endpoints del monolito» — priorización

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M028

Escenario final: «convence al CTO (en contra) de microservicios»

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M029

Escenario: «startups rápido: ¿empezamos con microservicios?» — respuesta honesta

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M030

Microservice chassis: la base común de todo servicio

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M031

Agregados DDD: la unidad de consistencia

AVANZADO

MICROSERVICIOS

Fundamentos y decisiones

M032

Context Mapping: relaciones entre bounded contexts

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Fundamentos y decisiones

M026

RESPUESTA

Corto plazo: separar módulos claros + CODEOWNERS + checks de fronteras (ArchUnit).
 Mediano: extraer lo que el segundo equipo necesita a SU servicio (el motivo de roce indica frontera mal puesta).
 Alternativa: módulos internos deployables (modulith) con releases independientes.
 El roce recurrente entre equipos es LA señal clásica para dividir.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M025

RESPUESTA

Malo: API composition en vivo (lento y frágil para analítica).
 Bueno: read model analítico — eventos de los 6 servicios → warehouse (BigQuery) → reporte/dashboards con datos de minutos de antigüedad.
 Operativo-urgente: materialized view alimentada por CDC.
 La regla: analítica NO cruza servicios en vivo; se replica con propósito.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M028

RESPUESTA

Presenta AMBOS lados: costo real (plataforma, on-call, latencia, complejidad de datos) vs beneficio (velocity de equipos, escala, resiliencia).
 Propón el camino intermedio: monolito modular ahora, fronteras por DDD, extracción por señales concretas (escala/equipos).
 Muestra métricas: DORA actuales y objetivo.
 El arquitecto senior no vende microservicios: vende el diseño que RESUELVE el problema de la organización.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M027

RESPUESTA

Por valor/riesgo: 1) primero casos de uso AUTOCONTENIDOS y de alto cambio (más beneficio, menos riesgo). 2) Frontend en momentos de pico (escala distinta). 3) Rutas write-heavy con necesidades de escala. 4) Lo último: núcleo transaccional entrelazado.
 Nunca empezar por lo más difícil para 'demostrar valor' — empieza por lo que libera velocidad al negocio.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M030

RESPUESTA

Plantilla/framework que resuelve lo TRANSVERSAL para que cada servicio solo programe su negocio: config externalizada, health checks, logging/metrics/tracing, seguridad, registro.
 Spring Boot ES un chasis de facto; otros: Micronaut, Quarkus.
 En la práctica: un starter/template interno de la organización (golden path).
 Sin chasis: cada servicio inventa su propio logging y su on-call sufre.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M029

RESPUESTA

No. Empieza monolito modular: 1 deploy, 1 BD, equipo pequeño va rapidísimo.
 Disciplina barata que paga después: módulos por dominio, sin joins cruzados entre módulos, eventos internos desde el inicio, CI por módulo.
 Cuando un módulo CRECE y tiene dueño propio → extracción mecánica (esa ruta ya se anduvo).
 La velocidad inicial manda: el dominio no descubierto se corta mal siempre.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M032

RESPUESTA

Shared Kernel (código común), Customer/Supplier (aguas arriba atiende al downstream), Conformist (aceptas el modelo del otro), Anticorruption Layer (traduces su modelo al tuyo — para sistemas legados/externos), Open Host Service + Published Language (tu API es el estándar).
 Decide por relación de poder y estabilidad del otro contexto.

AVANZADO

MICROSERVICIOS

Fundamentos y decisiones

M031

RESPUESTA

Un agregado agrupa entidades bajo una raíz que protege INVARIANTES (Pedido con sus Items: total = suma de items).
 Reglas: referencia otros agregados por ID (no por objeto), una transacción modifica UN agregado, eventos publicados al cambiar.
 Microservicio ≠ agregado, pero los agregados guían qué va junto y qué se comunica por eventos.

AVANZADO

MICROSERVICIOSFundamentos y decisionesM033

Hexagonal/Ports & Adapters dentro de un microservicio

AVANZADO

MICROSERVICIOSFundamentos y decisionesM034

Arquitectura evolutiva: fitness functions

AVANZADO

MICROSERVICIOSFundamentos y decisionesM035

Branch by abstraction: refactor sin big bang

AVANZADO

MICROSERVICIOS★ComunicaciónM036

Comunicación síncrona vs asíncrona

INTERMEDIO

MICROSERVICIOS★ComunicaciónM037

REST vs gRPC

INTERMEDIO

MICROSERVICIOS★ComunicaciónM038

Service Discovery: ¿por qué y cómo?

INTERMEDIO

MICROSERVICIOSComunicaciónM039

Event notification vs event-carried state transfer

INTERMEDIO

MICROSERVICIOSComunicaciónM040

Request-reply sobre mensajería: cómo y cuándo

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Fundamentos y decisiones

M034

RESPUESTA

Métricas automatizadas que validan UNA decisión arquitectónica continuamente: ArchUnit (dependencias entre módulos), contract tests (acoplamiento de APIs), performance test en CI (presupuesto de latencia), security scans (postura).
Si la función falla, la deriva arquitectónica se detiene en CI, no en producción.
Es como tener un arquitecto revisando cada commit.

AVANZADO

MICROSERVICIOS

Fundamentos y decisiones

M033

RESPUESTA

Dominio en el centro (entidades, reglas) SIN dependencias de Spring/JPA/HTTP.
Puertos: interfaces que el dominio define (RepositoryPort, NotificationPort).
Adapters: implementaciones de infraestructura (JpaAdapter, KafkaAdapter, RestController).
Beneficio: dominio testeable sin levantar nada, swap de infraestructura, y la prueba de que tu lógica no está en los adapters.

AVANZADO

MICROSERVICIOS



Comunicación

M036

RESPUESTA

Síncrona (REST, gRPC): el llamador espera; acopla en el tiempo (si el destino cae, falla).
Asíncrona (Pub/Sub, Kafka): eventos/mensajes; desacopla en el tiempo, absorbe picos, más resiliente; pero consistencia eventual.
Mezcla habitual: comandos críticos sync + eventos async para propagar cambios.

INTERMEDIO

MICROSERVICIOS

Fundamentos y decisiones

M035

RESPUESTA

Cuando un cambio ESTRUCTURAL es grande (cambiar una librería clave, dividir un módulo):
1) Creas una ABSTRACCIÓN (interfaz) sobre el código a cambiar.
2) El código viejo la implementa; todo el resto del sistema usa la abstracción.
3) En un branch desarrollas la NUEVA implementación.
4) La activas por feature flag/config y migras llamada por llamada.
Permite refactors largos sin congelar el trunk.

AVANZADO

MICROSERVICIOS



Comunicación

M038

RESPUESTA

En la nube las instancias suben/bajan con IPs dinámicas: hay que localizarlas en runtime.
Client-side: el cliente consulta un registro (Eureka, Consul) y balancea él.
Server-side: Service + DNS de Kubernetes o un LB ocultan las instancias.
En K8s nativo no necesitas Eureka: el DNS del Service resuelve.

INTERMEDIO

MICROSERVICIOS



Comunicación

M037

RESPUESTA

REST: recursos + verbos HTTP, JSON legible, cacheable, tooling universal — ideal para APIs públicas.
gRPC: HTTP/2 + Protobuf binario, contratos tipados (.proto), streaming bidireccional, mucho más rápido — ideal interno servicio-a-servicio.
REST se documenta con OpenAPI; gRPC genera código cliente/servidor.

INTERMEDIO

MICROSERVICIOS

Comunicación

M040

RESPUESTA

Necesitas respuesta async: publicas comando con reply_to + correlation_id; el responder publica la respuesta al topic de reply; el cliente correlaciona por id.
Casos: backend-for-backend async donde sync acoplaría.
Complejidad: timeouts, polling de respuesta, dos colas.
Si puedes, REST/gRPC directo es más simple — async-reply solo con justificación.

INTERMEDIO

MICROSERVICIOS

Comunicación

M039

RESPUESTA

Notification: 'pasó X, id=42' — el consumidor debe llamar de vuelta para datos (acoplamiento a tu API, riesgo de N+1 remoto).
Event-carried state: el evento TRAE los datos necesarios (PedidoCreado con items y cliente) — consumidor construye su vista sin llamarte (desacoplado, datos posiblemente viejos).
Regla: notificación mínima + API para detalles, o state transfer si el consumidor es estable.

INTERMEDIO

MICROSERVICIOS★

Comunicación

M041

Deadlines y presupuesto de latencia en cascadas de servicios

INTERMEDIO

MICROSERVICIOS

Comunicación

M042

Schema evolution: compatibilidad forward y backward

INTERMEDIO

MICROSERVICIOS

Comunicación

M043

JSON vs Avro vs Protobuf para mensajería

INTERMEDIO

MICROSERVICIOS

Comunicación

M044

Consumer groups y particiones (Kafka): cómo escala un topic

INTERMEDIO

MICROSERVICIOS

Comunicación

M045

Idempotent producer y transacciones en Kafka

INTERMEDIO

MICROSERVICIOS

Comunicación

M046

Retry amplification: cómo los reintentos escalan una caída

INTERMEDIO

MICROSERVICIOS

Comunicación

M047

Correlation ID vs message ID vs causation ID

INTERMEDIO

MICROSERVICIOS

Comunicación

M048

CloudEvents: el sobre estándar de eventos

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Comunicación

M042

RESPUESTA

Backward: schema NUEVO lee datos VIEJOS (añadir campo con default — seguro). Forward: schema viejo lee datos nuevos (no romper al añadir).

Full compatibility: ambos.

Reglas prácticas: no renombrar (añadir + deprecarse), no cambiar tipos, campos nuevos OPCIONALES.

Protobuf/Avro + schema registry validan en CI.

INTERMEDIO

MICROSERVICIOS



Comunicación

M041

RESPUESTA

Cada servicio tiene un presupuesto del tiempo total: A(50ms) llama B(200ms) que llama C(150ms)...

Propagar deadline: si a A le quedan 80ms y B necesita 200, B ni intenta (fail fast) — header de deadline (gRPC lo trae nativo).

Sin deadlines, los reintentos amplifican la latencia (retry storm) y el P99 explota.

Medir por tramo con tracing.

INTERMEDIO

MICROSERVICIOS

Comunicación

M044

RESPUESTA

Topic dividido en N particiones (orden SOLO dentro de partición, por key).

Consumer group: cada partición se asigna a UN consumidor del grupo → paralelismo máx = #particiones.

Rebalanceo cuando entran/salen consumidores (pausa breve).

Offsets: posición por consumidor, commiteada — replay posible re-seateando.

INTERMEDIO

MICROSERVICIOS

Comunicación

M043

RESPUESTA

JSON: legible, universal, verbose, sin schema fuerte (JSON Schema opcional).

Protobuf: binario compacto, tipado, código generado, requiere .proto compartido — el estándar gRPC.

Avro: binario + schema en registry, evolución sofisticada — estándar Kafka.

Decisión: ecosistema (gRPC→protobuf), herramientas y habilidad del equipo pesan más que benchmarks.

INTERMEDIO

MICROSERVICIOS

Comunicación

M046

RESPUESTA

A reintenta a B 3 veces; B reintenta a C 3 veces → un clic del usuario = 27 llamadas a C. Servicio caído recibe 27x carga = nunca se recupera.

Defensas: retry budget global (máx % de tráfico como retry), solo reintentar en el borde, circuit breakers que cortan reintentos, deadlines que acaban cadenas.

Regla: reintentar en UNA capa, no en todas.

INTERMEDIO

MICROSERVICIOS

Comunicación

M045

RESPUESTA

Idempotent producer: enable.idempotence=true → broker deduplica reintentos por (PID, secuencia) — exactamente-once POR PARTIÓN en escritura.

Transacciones: escribir a varias particiones + offset commit atómicamente (consume-transform-produce) — exactly-once end-to-end del pipeline.

Costo: throughput algo menor, complejidad de aislamiento (read_committed).

INTERMEDIO

MICROSERVICIOS

Comunicación

M048

RESPUESTA

Spec de CNCF: id, source, type (com.empresa.pedido.creado.v1), specversion, time, datacontenttype + extensiones (traceparent, tenantid).

Beneficio: herramientas neutrales (routers, validators, sinks) entienden TODOS tus eventos; el type versionado evita magic strings.

Adoptar en el wrapper del outbox desde el día 1 — migrar después duele.

INTERMEDIO

MICROSERVICIOS

Comunicación

M047

RESPUESTA

Message ID: identidad ÚNICA del mensaje (dedup).

Correlation ID: agrupa toda una petición de negocio end-to-end (tracing).

Causation ID: qué mensaje PRECISO provocó este (cadena de eventos: PedidoCreado causa EmailEnviado).

En headers/wrapper CloudEvents: id, correlationid, causationid — logs y trazas se vuelven navegables.

INTERMEDIO

MICROSERVICIOS

Comunicación

M049

Event granularity: fino, grueso y la regla práctica

INTERMEDIO

MICROSERVICIOS

Comunicación

M050

Versionado de eventos: compatibilidad en la práctica

INTERMEDIO

MICROSERVICIOS

Comunicación

M051

Long polling, streaming y gRPC bidi: para datos en vivo

INTERMEDIO

MICROSERVICIOS

Comunicación

M052

Escenario: «el consumidor necesita datos de 3 eventos para actuar» — patrón

INTERMEDIO

MICROSERVICIOS

Comunicación

M053

Escenario: «la cola crece sin parar en Black Friday» — playbook

INTERMEDIO

MICROSERVICIOS

Comunicación

M054

Escenario: «el gateway es un SPOF» — diseños alternativos

INTERMEDIO

MICROSERVICIOS

Comunicación

M055

Message Router / Content-based routing: qué es

INTERMEDIO

MICROSERVICIOS

Comunicación

M056

Backpressure en sistemas de mensajería: quién frena

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Comunicación

M050

RESPUESTA

type con versión (pedido.creado.v2). V2 añade campos OPCIONALES (consumidores viejos ignoran).
Cambios incompatibles: nuevo type v2 CONVIVE con v1 (publicas ambos durante transición), migras consumidores, matas v1 con fecha.
Nunca: cambiar semántica del mismo type en silencio — los consumidores viejos seguirán funcionando MAL sin error.

INTERMEDIO

MICROSERVICIOS

Comunicación

M049

RESPUESTA

Muy fino (CampoActualizado por cada campo): tormenta de eventos, consumidores engordando.
Muy grueso (NegocioCambiado con todo): consumidores filtrando y parseando de más.
Regla: un evento por HECHO DE NEGOCIO significativo (PedidoCreado, PagoRechazado), payload con las entidades afectadas + id + versión.

INTERMEDIO

MICROSERVICIOS

Comunicación

M052

RESPUESTA

Problema: eventos llegan desordenados/parciales.
Opciones: 1) event-carried state (cada evento trae snapshot completo del agregado — consumidor siempre tiene lo último). 2) consumidor mantiene estado incremental y actúa cuando la clave está completa. 3) orquestador que consulta APIs al trigger.
Preferir 1: elimina dependencia de orden entre eventos distintos.

INTERMEDIO

MICROSERVICIOS

Comunicación

M051

RESPUESTA

gRPC bidirectional streaming: canal persistente tipo-duplex con backpressure nativo — el más eficiente entre servicios de tu dominio.
SSE/WebSocket en el edge para clientes.
Long polling: fallback universal, simple pero desperdicia conexiones.
Criterio: frecuencia de datos y tamaño de payload — eventos espaciados → SSE; continuo/binario → gRPC bidi.

INTERMEDIO

MICROSERVICIOS

Comunicación

M054

RESPUESTA

Múltiples instancias del gateway detrás de un LB anycast (nunca 1 pod).
Dos gateways con rutas distintas (uno para APIs públicas, otro interno) — blast radius.
DNS failover entre regiones con gateway regional + LB global.
Cliente con reintentos + caché de descubrimiento si el gateway cae.
El gateway puede degradar, no puede desaparecer: capacidad N+1 siempre.

INTERMEDIO

MICROSERVICIOS

Comunicación

M053

RESPUESTA

- 1) Confirmar: es lag del consumidor o throughput del productor legítimo.
- 2) Escalar horizontal consumidores (KEDA por lag).
- 3) Bajar trabajo por mensaje (batching), subir paralelismo por partición.
- 4) Shedding: degradar trabajo no crítico (deferir emails).
- 5) Si los datos lo toleran: ralentizar productores (429) — el usuario entiende 'lento' mejor que 'perdido'.

INTERMEDIO

MICROSERVICIOS

Comunicación

M056

RESPUESTA

Productor más rápido que consumidor = colas creciendo sin fin (latencia + OOM).
Estrategias: colas con límite (reject/persist en DLQ), consumer escala horizontal (KEDA por lag), batching del consumidor, y si el origen puede frenarse: HTTP 429 backpressure real.
Monitoreo clave: consumer lag — el indicador de salud #1 de un pipeline async.

AVANZADO

MICROSERVICIOS

Comunicación

M055

RESPUESTA

Decidir el destino de un mensaje por su CONTENIDO (tipo, atributos): 'eventos de pago → topic pagos; alertas → topic ops'.
Implementado con filters de subscription (Pub/Sub), routing keys (Rabbit) o el gateway de eventos (Eventarc).
Mantén las reglas declarativas y versionadas — routers con lógica oculta se vuelven indebuggeables.

INTERMEDIO

MICROSERVICIOS

Comunicación

M057

Fan-out y scatter-gather: patrones de agregación

AVANZADO

MICROSERVICIOS

Comunicación

M058

Splitter / Aggregator (EIP): dividir y recomponer trabajo

AVANZADO

MICROSERVICIOS

Comunicación

M059

Process Manager: el primo serio del orquestador

AVANZADO

MICROSERVICIOS

Comunicación

M060

Dead letter processing: automatizar la reparación

AVANZADO

MICROSERVICIOS

Datos y consistencia

M061

Proyecciones CQRS: rebuild y testing

BASICO

MICROSERVICIOS

★

Datos y consistencia

M062

Database per service

INTERMEDIO

MICROSERVICIOS

★

Datos y consistencia

M063

Consistencia eventual

INTERMEDIO

MICROSERVICIOS

★

Datos y consistencia

M064

Integración directa a la BD de otro servicio: por qué está prohibido

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Comunicación

M058

RESPUESTA

Splitter: un pedido grande → N items como mensajes independientes (paralelismo por ítem).

Aggregator: recolecta los N resultados (correlation por pedido_id) y emite el resultado completo cuando todos llegan (o timeout con parciales).

Estado del aggregator: Redis/DB con TTL.

Ejemplo clásico: procesar factura línea a línea y consolidar.

AVANZADO

MICROSERVICIOS

Comunicación

M057

RESPUESTA

Fan-out simple: publicar a topic, N suscriptores actúan independientes.

Scatter-gather: llamar a N servicios en PARALELO y combinar respuestas (búsqueda que consulta inventario+precios+envío) — con timeout por rama y degradación parcial (mostrar lo que respondió).

Herramientas: CompletableFuture.allof, orquestadores (Workflows), GraphQL.

Riesgo: latencia del más lento → bulkheads por rama.

AVANZADO

MICROSERVICIOS

Comunicación

M060

RESPUESTA

El DLQ no es un vertedero: pipeline de reparación — tooling que inspecciona el mensaje + error, permite editar payload y re-publicar (replay) con un click/endpoint.

Métrica: edad del mensaje más viejo en DLQ; alerta cuando > umbral.

Auditoría: quién re-publicó qué y cuándo (los mensajes re-procesados marcan retried=true).

AVANZADO

MICROSERVICIOS

Comunicación

M059

RESPUESTA

Máquina de estados distribuida: recibe eventos/comandos, decide próximo paso según estado persistido, emite comandos.

Diferencia con orquestador simple: maneja MÚLTIPLES instancias del flujo concurrentes, timeouts, reintentos y compensaciones como reglas declarativas.

Implementaciones: Workflows (GCP), Temporal, Camunda — o tabla de saga propia.

AVANZADO

MICROSERVICIOS



Datos y consistencia

M062

RESPUESTA

Cada microservicio es DUEÑO de su esquema/BD; nadie más lo consulta directo.

Beneficio: acoplamiento cero, tecnología propia, escalado independiente.

Consecuencia: no hay joins entre servicios → compartir por APIs o eventos; consultas cruzadas con API composition o CQRS.

INTERMEDIO

MICROSERVICIOS

Datos y consistencia

M061

RESPUESTA

Proyección = función determinista (estado + evento → nuevo estado) con posición guardada (offset/version por proyección).

Rebuild: borrar read store y reprocesar desde evento 0 — corre como deploy (verificando conteos antes de cutover).

Testing: fixture de eventos → assert de estado final; test de idempotencia (mismo evento 2 veces = mismo estado).

Versiona la proyección junto al evento.

BÁSICO

MICROSERVICIOS



Datos y consistencia

M064

RESPUESTA

El dueño del esquema no puede cambiar nada sin romper a los demás (acoplamiento por schema), no hay control de acceso por negocio, no hay boundary de transacciones, dificulta refactor total.

Correcto: API (para queries puntuales) o eventos (para replicar datos que el otro necesita localmente).

La BD de un servicio es un detalle de implementación PRIVADO.

INTERMEDIO

MICROSERVICIOS



Datos y consistencia

M063

RESPUESTA

Con mensajería asíncrona y réplicas, los datos convergen con el tiempo: un usuario puede leer datos ligeramente viejos.

Se diseña con: lectura de tu propia escritura, versiones/ETags, compensación en UI, procesos idempotentes.

Si el negocio exige fuerte → transacción local o BD distribuida consistente (Spanner).

INTERMEDIO

MICROSERVICIOS★

Datos y consistencia

M065

API composition: el agregador simple y su límite

INTERMEDIO

MICROSERVICIOS

Datos y consistencia

M066

Weak vs strong consistency por tipo de dato: tabla de decisiones

INTERMEDIO

MICROSERVICIOS

Datos y consistencia

M067

Consistencia en UI: patrones de UX para eventual

INTERMEDIO

MICROSERVICIOS

Datos y consistencia

M068

Soft delete vs hard delete entre servicios

INTERMEDIO

MICROSERVICIOS

Datos y consistencia

M069

Sagas y validación de negocio: rechazo asincrónico

INTERMEDIO

MICROSERVICIOS

Datos y consistencia

M070

Data ownership ambiguity: quién es dueño de 'cliente'

INTERMEDIO

MICROSERVICIOS★

Datos y consistencia

M071

Patrón Saga (transacciones distribuidas)

AVANZADO

MICROSERVICIOS★

Datos y consistencia

M072

¿Qué es CQRS?

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Datos y consistencia

M066

RESPUESTA

Fuerte requerido: saldos, stock vendible, permisos, pagos.
 Eventual aceptable: feeds, notificaciones, contadores de vistas, rankings, perfiles.
 Gray zone (strong en escritura del dueño, eventual para lecturas de otros): catálogo, inventario mostrado.
 Elegir por dato, no por sistema — mezclar en el mismo micro es normal.

INTERMEDIO

MICROSERVICIOS



Datos y consistencia

M065

RESPUESTA

Orquestador llama a N servicios y junta la respuesta (página de producto = catálogo + precio + stock).
 Simple y consistente-en-tiempo-real.
 Límite: latencia = suma/máx de llamadas, disponibilidad = producto de las partes, JOINS complejos (filtros cross-service) imposibles.
 Escala mal para queries analíticas → ahí entra CQRS con read model materializado.

INTERMEDIO

MICROSERVICIOS

Datos y consistencia

M068

RESPUESTA

Hard delete en el dueño → consumidores con referencia muerta (FK fantasma). Mejor: eventos de tipo 'eliminado' + soft delete (status=deleted, filtro en query) durante la ventana de sincronización.
 Purga real (GDPR): job que propaga el borrado a todos los read models + logs con tokenización previa.
 Los backups: definir retención y exclusiones desde el diseño.

INTERMEDIO

MICROSERVICIOS

Datos y consistencia

M067

RESPUESTA

Optimistic UI: mostrar el estado esperado inmediatamente ('Tu pedido está en proceso'), sincronizar al confirmar.
 Pendiente explícito: estados visibles PENDING/PROCESANDO con refresh automático (polling/websocket).
 Read-your-writes: tras escribir, las consultas del usuario van al write model o a réplica actualizada.
 NEVER: pantalla de error por lag eventual que el usuario percibe como fallo.

INTERMEDIO

MICROSERVICIOS

Datos y consistencia

M070

RESPUESTA

Típico bloqueo: pedidos necesita datos del cliente, marketing también, soporte también.
 Regla: la ENTIDAD maestra (fuente de verdad del registro) tiene UN dueño (servicio identidad/clientes); los demás mantienen VISTAS locales (nombre, email) actualizadas por eventos.
 Cambios al modelo maestro: propuesta al dueño, no 'un updatecillo' directo a su tabla.

INTERMEDIO

MICROSERVICIOS

Datos y consistencia

M069

RESPUESTA

En saga, el 'no' llega tarde (pago rechazado 3s después de crear el pedido).
 Diseño: estados intermedios visibles (PENDING_PAYMENT), el cliente ve el progreso, compensación automática al rechazo, y notificación final (push/email).
 Prohibido: crear el pedido COMPLETO y 'borrarlo' si falla — estados y compensaciones, no deshacer silencioso.

INTERMEDIO

MICROSERVICIOS



Datos y consistencia

M072

RESPUESTA

Command Query Responsibility Segregation: separar el modelo de ESCRITURA (comandos, invariantes, normalizado) del de LECTURA (queries, denormalizado, optimizado para buscar).
 Se sincronizan por eventos.
 Útil con cargas asimétricas o vistas complejas; añade duplicación y eventualidad.

AVANZADO

MICROSERVICIOS



Datos y consistencia

M071

RESPUESTA

Transacción que cruza servicios = secuencia de transacciones LOCALES; si un paso falla, se ejecutan acciones COMPENSATORIAS que deshacen los anteriores.
 Coreografía: cada servicio reacciona a eventos, sin coordinador.
 Orquestación: un orquestador central dirige los pasos.
 Evita el 2PC.

AVANZADO

MICROSERVICIOS★

Datos y consistencia

M073

¿Qué es Event Sourcing?

AVANZADO

MICROSERVICIOS

Datos y consistencia

M074

Idempotencia: ¿por qué es clave en microservicios?

AVANZADO

MICROSERVICIOS★

Datos y consistencia

M075

Replicación de datos entre servicios: read model local

AVANZADO

MICROSERVICIOS★

Datos y consistencia

M076

Outbox pattern: implementación paso a paso

AVANZADO

MICROSERVICIOS

Datos y consistencia

M077

Inbox pattern: el complemento del outbox

AVANZADO

MICROSERVICIOS

Datos y consistencia

M078

2PC/XA: por qué los microservicios lo evitan

AVANZADO

MICROSERVICIOS

Datos y consistencia

M079

Saga coreografiada: ejemplo pedido→pago→inventario

AVANZADO

MICROSERVICIOS

Datos y consistencia

M080

Saga orquestada: cuándo el coordinador central gana

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Datos y consistencia

M074

RESPUESTA

Con reintentos y entrega at-least-once, la misma petición/mensaje puede procesarse 2+ veces.
Idempotente = aplicarlo N veces da el mismo resultado que una vez.
Técnicas: clave de idempotencia (header Idempotency-Key), upsert/MERGE, constraints únicos, deduplicación de IDs de mensaje.

AVANZADO

MICROSERVICIOS



Datos y consistencia

M073

RESPUESTA

Guardar el estado como secuencia INMUTABLE de eventos (no el estado actual): se reconstruye con replay.
Da auditoría completa, time travel y encaja con CQRS.
Costos: versionado de eventos, snapshots por rendimiento, complejidad.
Ej.: cuenta bancaria = lista de movimientos.

AVANZADO

MICROSERVICIOS



Datos y consistencia

M076

RESPUESTA

1) En la MISMA tx de negocio: INSERT entidad + INSERT outbox(event_id, tipo, payload, status=PENDING).
2) Relay: poller SELECT ... WHERE pending FOR UPDATE SKIP LOCKED, o CDC (Debezium leyendo el WAL).
3) Publicar a broker → marcar SENT (o dejar que el offset/CDC haga de marcador).
Garantiza: evento sale si y solo si la tx confirmó. Nunca dual-write directo.

AVANZADO

MICROSERVICIOS



Datos y consistencia

M075

RESPUESTA

Patrón: B necesita datos de A frecuentemente (nombre del cliente en 10k queries/min).
En vez de llamar a A siempre: A publica eventos de cambio → B mantiene COPIA LOCAL (tabla cliente_basico) actualizada por eventos.
Trade-off: consistencia eventual + almacenamiento duplicado a cambio de latencia local e independencia de A.
Es CQRS sin nombrarlo.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M078

RESPUESTA

Two-phase commit bloquea recursos entre fases (prepare/commit): latencia y locks largos, el coordinador es SPOF, muchos recursos modernos (Kafka, HTTP APIs) no lo soportan. Escala mal horizontalmente.
Alternativa: Saga (tx locales + compensaciones) — disponible eventualmente pero disponible siempre.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M077

RESPUESTA

El consumidor también tiene el problema: procesar mensaje + guardar resultado deben ser atómicos.
Solución: en la tx de negocio, INSERT INTO inbox(message_id) — si viola unique, ya procesado (dedup atómica).
Outbox (emisor) + Inbox (receptor) = entrega efectivamente exacta entre servicios con brokers at-least-once.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M080

RESPUESTA

Un orquestador (PedidoSaga) dirige: cobrar → reservar → crear envío, con estados y timeout por paso.
Ventaja: flujo visible en UN lugar, fácil de auditar/modificar, timeouts y compensaciones explícitas.
Desventaja: coordinador puede engordar ('god service'), acople al conocer los pasos.
Regla: flujos con >3 pasos o ramas → orquestada; simples 2-3 reacciones → coreografiada.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M079

RESPUESTA

PedidoService crea pedido(PENDING) → publica PedidoCreado.
PagoService escucha, cobra → PagoConfirmado / PagoRechazado.
InventarioService reserva → StockReservado.
PedidoService escucha todo → COMPLETADO (todo ok) o compensa (CANCELADO + reembolso si pago ya había ido).
Sin coordinador: cada servicio reacciona. Difícil de rastrear → tracing obligatorio.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M081

Compensaciones de saga: transacciones que no son reversibles

AVANZADO

MICROSERVICIOS

Datos y consistencia

M082

CQRS: read model materializado en la práctica

AVANZADO

MICROSERVICIOS

Datos y consistencia

M083

Event sourcing: aggregates, streams y snapshots

AVANZADO

MICROSERVICIOS

Datos y consistencia

M084

Event sourcing: upcasting y GDPR (crypto-shredding)

AVANZADO

MICROSERVICIOS

★

Datos y consistencia

M085

Anomalías de consistencia eventual (y sus defesas)

AVANZADO

MICROSERVICIOS

Datos y consistencia

M086

Conflict resolution: LWW, versiones y CRDTs en una pincelada

AVANZADO

MICROSERVICIOS

Datos y consistencia

M087

Migraciones entre servicios: expand-contract a través de límites

AVANZADO

MICROSERVICIOS

Datos y consistencia

M088

Caché distribuida en microservicios: dónde y con qué TTL

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Datos y consistencia

M082

RESPUESTA

Write side: Cloud SQL normalizado, validación de invariantes.
 Eventos (outbox) → proyección (worker) → read store:
 Elasticsearch (búsqueda), Redis (hot), Firestore (por usuario).
 Reglas: proyecciones IDEMPOTENTES (reprocesar no rompe),
 versionadas (rebuild desde evento 0), lag monitoreado.
 Consulta nunca escribe: comandos van al write side.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M081

RESPUESTA

Compensar NO es rollback: un pago compensable es reembolso
 (no 'deshacer'), un email enviado no se des-envía (compensación
 = enviar aviso de corrección).
 Clasificar pasos: compensables (reembolso), pivot (el punto de no
 retorno — el pago capturado), retrieable (envíos con retry).
 Diseño: poner el pivot LO MÁS TARDE posible.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M084

RESPUESTA

Upcasting: eventos versionados (PedidoCreado_v1, _v2) —
 transformer de versión vieja a nueva al leer; NUNCA reescribir
 historia.
 GDPR/borrar usuario: los eventos son inmutables →
 crypto-shredding: datos personales cifrados con clave POR
 USUARIO; borrar la clave = datos irrecuperables, eventos intactos.
 Alternativa: tokenización (el evento guarda token, los datos reales
 en vault borrrable).

AVANZADO

MICROSERVICIOS

Datos y consistencia

M083

RESPUESTA

Stream por agregado (pedido-42): eventos ordenados con
 version/sequence (optimistic lock natural).
 Estado actual = fold de eventos; para performance: snapshot
 cada N eventos y aplicar solo los posteriores.
 Comandos validan contra el estado reconstruido y APPENDIAN
 nuevos eventos (nunca UPDATE).
 Concurrencia: dos comandos sobre versión vieja → conflicto,
 reintentar.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M086

RESPUESTA

Last-Write-Wins (timestamp): simple, pierde writes
 concurrentes — solo si la pérdida es aceptable (perfil de usuario).
 Versiones/tickets: rechazar writes sobre versión vieja (optimistic)
 → cliente resuelve.
 CRDT (counters, sets que convergen): cuando múltiples editores
 deben converger sin coordinación (carrito offline móvil).
 Evitar 'resolver en la BD' sin definir semántica de negocio.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M085

RESPUESTA

Stale read (saldo viejo), read-your-writes roto (creo pedido, no lo
 veo), monotonic read roto (veo versión nueva, luego vieja).
 Defensas: read-your-writes → leer del write model o sticky
 routing tras escritura; versiones/ETags; UI optimista (mostrar
 'pendiente'); UX que tolera lag ('procesando...').
 Documentar QUÉ tan eventual es cada dato (p50/p99 de lag) —
 el negocio decide si acepta.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M088

RESPUESTA

Niveles: caché de app local (Caffeine, microsegundos, por pod) +
 caché distribuida (Redis, ms, compartida) + HTTP/CDN (bordes).
 TTL por tolerancia al stale: precios 30-60s, catálogo 5-15min,
 config 1min.
 Invalidación activa por eventos cuando el stale duele (precio
 cambió → publish invalidation).
 Jamás cachear autorización por largo TTL.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M087

RESPUESTA

Mover 'cliente' del monolito a su servicio: 1) sincronizar por CDC
 a la nueva BD (dual write implícito), 2) monolito Lee de nuevo
 servicio vía API, 3) escrituras migradas (feature flag), 4)
 monitorear paridad, 5) cortar el flujo viejo, 6) drop tabla en
 monolito.
 Cada paso reversible; cutover con flag, no con deploy.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M089

Locks distribuidos: Redis SET NX y el fencing token

AVANZADO

MICROSERVICIOS

Datos y consistencia

M090

Materialized views: cuándo pre-construir la respuesta

AVANZADO

MICROSERVICIOS

Datos y consistencia

M091

Sharding de datos: cuándo y por qué casi nunca al inicio

AVANZADO

MICROSERVICIOS

Datos y consistencia

M092

CDC (Change Data Capture): Debezium y su rol moderno

AVANZADO

MICROSERVICIOS

Datos y consistencia

M093

Exactly-once en la práctica: el stacking de garantías

AVANZADO

MICROSERVICIOS

Datos y consistencia

M094

CAP aplicado (de verdad): qué eliges cuando la red se parte

AVANZADO

MICROSERVICIOS

Datos y consistencia

M095

Saga log: persistir el estado de la saga y reanudar

AVANZADO

MICROSERVICIOS



Datos y consistencia

M096

Dual write: el bug silencioso más común

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Datos y consistencia

M090

RESPUESTA

Query costosa y frecuente (dashboard de ventas por región por mes) → materializar en tabla de lectura actualizada por eventos/agendado.

Cloud SQL: vistas materializadas; BigQuery: scheduled queries/MVs.

Trade-off: espacio + staleness vs velocidad de lectura.

Perfecto para reportes donde 'hace 5 minutos' es aceptable.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M089

RESPUESTA

SET lock:key value NX PX 30000 → solo un cliente obtiene el lock; liberar con script Lua (comparar value para no borrar el lock ajeno).

Advertencia de Kleppmann: Redis async replication puede perder el lock (dos dueños) → usar FENCING TOKENS (número creciente verificado por el recurso) o etcd/ZooKeeper/Consul para correctividad estricta.

Preguntarse siempre: ¿de verdad necesito el lock o basta optimismo (versiones)?

AVANZADO

MICROSERVICIOS

Datos y consistencia

M092

RESPUESTA

Captura cambios del WAL/binlog de la BD y los publica como eventos en Kafka — SIN tocar el código de la app.

Usos: outbox relay, sincronizar monolito → nuevos servicios (strangler), alimentar analítica/elastic en tiempo real.

Cuidados: eventos capturan CAMBIOS de fila (no eventos de negocio semánticos), schema evolution, orden por tabla.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M091

RESPUESTA

Partir datos por clave (tenant, región) en múltiples BDs para escalar escrituras más allá de una máquina.

Costo: queries cross-shard, rebalanceo, clave de shard mal elegida = hotspots, joins imposibles.

Antes de sharding: índices, particionado, read replicas, caché, hardware mayor, CQRS.

Postgres/Cloud SQL escala muchísimo antes de necesitar shards.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M094

RESPUESTA

Ante partición, DEBES elegir: disponibilidad (aceptar writes que divergen — eventual, CRDTs) o consistencia (rechazar writes — locks, quorums).

CP: Spanner, ZooKeeper/etcd (consenso). AP: Cassandra, Dynamo-style, DNS.

PACELC añade: aún sin partición, eliges latencia vs consistencia (sync replication = latencia).

En la entrevista: di qué hace TU sistema por dato, no recites la teoría.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M093

RESPUESTA

Ningún sistema solo da exactly-once end-to-end; se compone: Emisor: outbox + (broker con dedup/idempotent producer).

Broker: at-least-once con orden por clave o exactly-once (Kafka tx).

Consumidor: inbox/dedup por message-id en la misma tx del efecto.

El 'exactly-once' real = at-least-once + idempotencia verificada en cada salto.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M096

RESPUESTA

Código: guardar en BD + publicar a cola, sin transacción entre ambos.

Fallas: crashea entre ambos (dato sin evento o evento sin dato), o tx rollback pero mensaje ya enviado.

Síntomas: datos que 'a veces' no aparecen downstream, imposible de reproducir.

Única solución sería: outbox (mismo tx) o broker transaccional en la misma BD.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M095

RESPUESTA

La saga misma es un agregado: tabla saga_log(saga_id, step, status, payload) — cada paso registra antes de ejecutar.

Crash recovery: al arrancar el orquestador, reanuda sagas en estados intermedios (reintenta paso pendiente, idempotente).

Timeout por paso → compensación automática.

Sin persistencia de saga, un restart pierde la mitad de las operaciones en curso.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M097

Snapshot y purge en event store:
rendimiento a largo plazo

AVANZADO

MICROSERVICIOS

Datos y consistencia

M098

Idempotencia con caducidad: el TTL del dedup

AVANZADO

MICROSERVICIOS

★

Resiliencia

M099

Patrones de resiliencia: timeout, retry,
circuit breaker, bulkhead

INTERMEDIO

MICROSERVICIOS

★

Resiliencia

M100

Timeout defaults sensatos por tipo de llamada

INTERMEDIO

MICROSERVICIOS

Resiliencia

M101

Graceful shutdown en microservicios:
checklist

INTERMEDIO

MICROSERVICIOS

Resiliencia

M102

Escenario: «la BD del catálogo se cae 30s
cada hora» — diseño defensivo

INTERMEDIO

MICROSERVICIOS

Resiliencia

M103

Escenario: «un cliente abuse 10x su
cuota» — protección progresiva

INTERMEDIO

MICROSERVICIOS

Resiliencia

M104

Circuit breaker + caché: el combo de lectura
resiliente

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Datos y consistencia

M098

RESPUESTA

La tabla de dedup crece para siempre si no expira: guarda (message_id, processed_at) con purga a 7-30 días (mayor periodo máximo de re-entrega del broker).
 Ventana peligrosa: un mensaje re-entregado DESPUÉS de purgar su id se procesa dos veces.
 Mitigación: hacer la operación naturalmente idempotente (upsert por entity_id) — el dedup es optimización, no la única defensa.

AVANZADO

MICROSERVICIOS

Datos y consistencia

M097

RESPUESTA

Streams de millones de eventos → reconstruir estado tarda segundos.
 Snapshot cada N eventos (ej. 100): estado serializado + versión → cargar snapshot + eventos posteriores.
 Purge: eventos viejos archivados a almacenamiento frío (el snapshot conserva el estado legalmente necesario).
 Monitorear: tiempo medio de reconstrucción por agregado.

AVANZADO

MICROSERVICIOS



Resiliencia

M100

RESPUESTA

DB queries: 1-5s (OLTP). Cache: 50-100ms. API interna crítica: 500ms-2s. API interna batch: 10-30s. API externa pagos: 5-10s.
 Email/webhook: async (nunca en el request path).
 Connect timeout SEPARADO del read timeout (2-3x menor).
 Documentar el presupuesto heredado: tu timeout ≤ presupuesto del llamador.

INTERMEDIO

MICROSERVICIOS



Resiliencia

M099

RESPUESTA

Timeout: nunca esperes infinito.
 Retry: backoff exponencial + jitter, solo fallos transitorios y siendo idempotente.
 Circuit breaker: corta llamadas a un servicio caído y responde con fallback.
 Bulkhead: aísla pools de hilos/conexiones por dependencia para que un fallo no hunda todo.
 Resilience4j los implementa.

INTERMEDIO

MICROSERVICIOS

Resiliencia

M102

RESPUESTA

Buscar causa raíz paralela (failover? OOM? migraciones?).
 Mientras tanto, aislar: caché de catálogo con serve-stale (TTL largo), breaker + fallback a catálogo embebido/mínimo, retry con backoff en lecturas.
 La app NO debe caerse por una BD de LECTURA: degradar el módulo, no el servicio.
 Post-mortem: la degradación es permanente hasta arreglar la causa.

INTERMEDIO

MICROSERVICIOS

Resiliencia

M101

RESPUESTA

1) Recibir SIGTERM → dejar de aceptar nuevos (readiness false).
 2) Terminar requests en curso (timeout). 3) Parar consumidores (ack/nack pendientes). 4) Flush de métricas/logs/buffers. 5) Cerrar pools. 6) exit 0.
 En K8s: terminationGracePeriodSeconds > suma de todo lo anterior.
 Sin esto: cada deploy pierde requests y mensajes.

INTERMEDIO

MICROSERVICIOS

Resiliencia

M104

RESPUESTA

Lectura crítica (precios): 1) caché L1 local, 2) Redis, 3) BD. Si todo falla → último valor conocido con timestamp ('precio al 14:32').
 El breaker protege la BD de la tormenta; el serve-stale mantiene la app útil.
 Header X-Cache: STALE para trazabilidad.
 Resultado: la caída de una dependencia de lectura se vuelve invisible para el usuario final.

INTERMEDIO

MICROSERVICIOS

Resiliencia

M103

RESPUESTA

Capa 1: rate limit por API key (429 rápido y barato).
 Capa 2: detección de patrón (abuse score) → degradación (menor prioridad de procesamiento, no bloqueo).
 Capa 3: política comercial (contactar, cobrar tier superior).
 Logs de abuso como evidencia. NUNCA bloqueos manuales 'a mano' — reglas versionadas y auditadas.

INTERMEDIO

MICROSERVICIOS

Resiliencia

M105

Budget de error en operaciones: cuándo congelar releases

INTERMEDIO

MICROSERVICIOS

Resiliencia

M106

Latency budget: diseñar el P99 antes de codificar

AVANZADO

MICROSERVICIOS

Resiliencia

M107

Circuit breaker: métricas de apertura y tuning fino

AVANZADO

MICROSERVICIOS

Resiliencia

M108

Fallbacks con criterio: qué responder cuando todo falla

AVANZADO

MICROSERVICIOS

Resiliencia

M109

Bulkhead: aislar por dependencia y por prioridad

AVANZADO

MICROSERVICIOS

Resiliencia

M110

Load shedding: soltar carga antes de caer

AVANZADO

MICROSERVICIOS

Resiliencia

M111

Graceful degradation: la matriz de degradación

AVANZADO

MICROSERVICIOS

Resiliencia

M112

Chaos engineering: el experimento bien planteado

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Resiliencia

M106

RESPUESTA

Define el objetivo: buscar producto < 300ms P99.
 Presupuesto por tramo: gateway 20ms, auth 15ms, servicio 150ms, DB 80ms, margen 35ms.
 Consecuencias prácticas: cada dependencia tiene su timeout ($\leq$ presupuesto), el agregador degrada ramas lentas, el caché existe donde el presupuesto no alcanza.
 Sin presupuesto, cada equipo optimiza a ciegas.

AVANZADO

MICROSERVICIOS

Resiliencia

M105

RESPUESTA

Error budget del mes consumido > 80% → congelar features no críticas, solo fixes de confiabilidad.
 Automatizar: el dashboard muestra el presupuesto; la política está escrita (quién decide, qué cuenta).
 Evita la discusión eterna '¿seguimos lanzando?' en pleno incidente — la matemática decide.

INTERMEDIO

MICROSERVICIOS

Resiliencia

M108

RESPUESTA

Buen fallback: valor default razonable (recomendaciones → 'populares'), caché stale (últimos precios conocidos), función reducida (checkout sin sugerencias).
 Mal fallback: exception genérica 500 (para eso no hacías breaker), respuesta VACÍA que la UI interpreta como 'no hay nada'.
 Documentar el comportamiento degradado en el diseño, no improvisarlo en el incidente.

AVANZADO

MICROSERVICIOS

Resiliencia

M107

RESPUESTA

Cuenta fallos en ventana (últimos N o últimos T segundos): ratio > umbral → OPEN.
 Tuning: failing calls mínimas antes de evaluar (evitar abrir por 2 fallos de tráfico bajo), slow calls (latencia > umbral) cuentan como fallos, half-open con pocas llamadas de prueba.
 Expón el estado como métrica y dashboard — un breaker abierto es un incidente de dependencia visible.

AVANZADO

MICROSERVICIOS

Resiliencia

M110

RESPUESTA

Cuando la saturación llega (cola llena, CPU alta): responder 503 + Retry-After a lo NUEVO y prioritario, o al revés (proteger pago, degradar búsqueda).
 Requiere clasificar requests (endpoint, usuario, prioridad).
 Mejor 90% de requests OK que 100% con timeout.
 K8s: HPA + PDB + queue limits; app: límite de cola + criterio de descarte.

AVANZADO

MICROSERVICIOS

Resiliencia

M109

RESPUESTA

Pools separados: conexiones/hilos para 'pagos' no pueden ser consumidos por 'analytics'.
 Por prioridad: clientes premium / free con capas distintas de rate limit y capacidad.
 Consecuencia: el colapso de una dependencia solo degrada SU compartimiento.
 En Resilience4j: bulkhead semáforo (in-process) o thread pool (aislamiento duro).

AVANZADO

MICROSERVICIOS

Resiliencia

M112

RESPUESTA

Hipótesis medible: 'si cae Redis, el P99 solo sube a X y el error rate queda < Y'.
 Experimento: definir blast radius (solo staging, % pods), inyectar fallo (network delay, kill pod, llenar disco), observar métricas vs hipótesis, documentar y arreglar lo roto.
 Herramientas: Chaos Mesh/Litmus en k8s, Gremlin.
 Nunca en producción sin kill switch y ventanas acordadas.

AVANZADO

MICROSERVICIOS

Resiliencia

M111

RESPUESTA

Tabla: dependencia caída → qué dejo de hacer → qué muestro.
 Ejemplos: recomendaciones caídas → ocultar sección; reviews caídas → 'temporalmente no disponible' (no página rota); pagos caídos → modo mantenimiento SOLO de checkout.
 Se diseña en el contrato del frontend (estados vacíos/degradados).
 Se prueba: chaos engineering apaga dependencias en staging.

AVANZADO

MICROSERVICIOS

Resiliencia

M113

Health checks que no mienten

AVANZADO

MICROSERVICIOS

Resiliencia

M114

Serve-stale: la caché como red de seguridad

AVANZADO

MICROSERVICIOS

Resiliencia

M115

Thundering herd en reintentos: jitter y token bucket

AVANZADO

MICROSERVICIOS

Resiliencia

M116

Anatomía de una cascada de fallos (y dónde cortarla)

AVANZADO

MICROSERVICIOS

Resiliencia

M117

Resilience4j en Spring: configuración YAML típica

AVANZADO

MICROSERVICIOS

Resiliencia

M118

Rate limiting por niveles de servicio (gold/platinum)

AVANZADO

MICROSERVICIOS

Resiliencia

M119

Self-healing automático: qué permite K8s y qué no

AVANZADO

MICROSERVICIOS

Resiliencia

M120

El patrón request hedging: duplicar llamadas para ganar P99

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Resiliencia

M114

RESPUESTA

Al fallar la BD/upstream, responder el valor cached aunque expirado (con header/etiqueta de staleness).
Implementación: caché con soft-TTL (fresco) y hard-TTL (aceptable si el origen cae).
En Redis/locales: guardado con versión + timestamp; el cliente decide si tolera datos viejos.
Transforma un outage en un degrade silencioso.

AVANZADO

MICROSERVICIOS

Resiliencia

M113

RESPUESTA

Liveness: solo 'el proceso responde' — NUNCA dependencias (BD caída ≠ reiniciar todos los pods: tormenta).
Readiness: incluye dependencias críticas CON cache/timeout corto (BD lenta ≠ muerta).
Startup: para apps que tardan (JVM + migraciones).
Cachear resultados de health (1-5s) para no convertir el health check en ataque.

AVANZADO

MICROSERVICIOS

Resiliencia

M116

RESPUESTA

C lento → B acumula hilos esperando → B agota pool → B rechaza a A → A también se ahoga → cascada total.
Puntos de corte: timeout en C (B no espera eterno), bulkhead en B (pool aislado), circuit breaker A→B (deja de insistir), load shedding en A (suelta tráfico), autoscaling como salida lenta.
Cada patrón corta UNA propagación — necesitas varios.

AVANZADO

MICROSERVICIOS

Resiliencia

M115

RESPUESTA

Todos los clientes reintentan al mismo tiempo → sincronización que tumba al que se recupera.
Jitter: backoff = $\text{base} * 2^n * \text{random}(0.5, 1.5)$ — desincroniza.
Token bucket server-side: el servidor acepta N reintentos/min totales (retry budget), excedente falla inmediato.
Client-side: circuit breaker evita reintentar contra servicio en OPEN.

AVANZADO

MICROSERVICIOS

Resiliencia

M118

RESPUESTA

Clasificar clientes (plan, prioridad) → presupuesto por clase: gold sin límite práctico, free con N rps.
En gateway: atributo del token (claim plan) → política correspondiente.
Bajo presión extrema: shedding empieza por free (proteger revenue-critical).
Comunicado claro al cliente: límites documentados + métrica de cuota consumida.

AVANZADO

MICROSERVICIOS

Resiliencia

M117

RESPUESTA

resilience4j:
circuitbreaker.instances.pagos: slidingWindowSize: 20,
failureRateThreshold: 50, waitDurationInOpenState: 30s,
slowCallDurationThreshold: 2s, slowCallRateThreshold: 80
retry.instances.pagos: maxAttempts: 3, waitDuration: 200ms,
enableExponentialBackoff: true
timelimiter.instances.pagos: timeoutDuration: 3s
Métricas automáticas en /actuador/metrics
(resilience4j.circuitbreaker.state).

AVANZADO

MICROSERVICIOS

Resiliencia

M120

RESPUESTA

Lanzar la misma llamada a 2 réplicas tras un pequeño delay (ej. 50ms) y tomar la primera respuesta — reduce tail latency dramáticamente en servicios con colas variadas.
Costo: 2x tráfico en los casos lentos; SOLO para operaciones read-only/idempotentes.
No aplicar globalmente: en dependencias con costos por llamada (APIs pagas) es tirar dinero.

AVANZADO

MICROSERVICIOS

Resiliencia

M119

RESPUESTA

K8s sana automáticamente: pod muerto → restart, nodo caído → reschedule, health check fallido → reinicio.
NO sana: dependencias externas caídas (necesitas breakers/fallbacks), bugs (necesitas rollback), saturación sostenida (necesitas autoscaling + shedding).
El equipo diseña PARA el restart: idempotencia, arranque rápido, estado externo.
Game days validan el self-healing real.

AVANZADO

MICROSERVICIOS

★

Observabilidad

M121

Los 3 pilares de la observabilidad

INTERMEDIO

MICROSERVICIOS

Observabilidad

M122

Distributed tracing: ¿cómo funciona?

INTERMEDIO

MICROSERVICIOS

Observabilidad

M123

Instrumentación con OpenTelemetry: qué se auto-detecta

INTERMEDIO

MICROSERVICIOS

Observabilidad

M124

Tracing: sampling head vs tail y su costo

INTERMEDIO

MICROSERVICIOS

Observabilidad

M125

SLOs por microservicio: quién define qué y cómo se negocia

INTERMEDIO

MICROSERVICIOS

Observabilidad

M126

Logging en microservicios: estructura y muestreo

INTERMEDIO

MICROSERVICIOS

Observabilidad

M127

Dependencias externas: cómo observar lo que no controlas

INTERMEDIO

MICROSERVICIOS

Observabilidad

M128

Postmortem blameless: estructura que funciona

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Observabilidad

M122

RESPUESTA

Cada petición lleva un `traceId` propagado en headers (W3C `traceparent`) por todos los saltos; cada servicio crea SPANS con timings enlazados al mismo trace.

Resultado: waterfall de toda la cadena con latencia por servicio. OpenTelemetry es el estándar (instrumentación + SDK).

INTERMEDIO

MICROSERVICIOS



Observabilidad

M121

RESPUESTA

LOGS (qué pasó, con contexto): centralizados (ELK, Cloud Logging) con correlation ID.

METRICS (números, tendencias): Micrometer →

Prometheus/Grafana; RED (rate, errors, duration) por servicio.

TRACING (recorrido entre servicios): OpenTelemetry → Jaeger/Tempo.

Las tres juntas: con solo logs no ves el sistema.

INTERMEDIO

MICROSERVICIOS

Observabilidad

M124

RESPUESTA

Head sampling: decide al INICIO (1% de traces) — barato, pero pierdes los errores raros.

Tail sampling: decides al FINAL con criterios (guardar 100% de errores y lentos, 1% del resto) — necesita collector con buffer.

Recomendación prod: head bajo + tail policy (errors, latencia p99, endpoints críticos).

Baggage: datos de negocio viajando con el trace (`tenant_id`) con cuidado en cardinalidad.

INTERMEDIO

MICROSERVICIOS

Observabilidad

M123

RESPUESTA

Agent/SDK auto-instrumenta: HTTP servers/clients, gRPC, JDBC (queries como spans), Kafka/PubSub, Redis, log correlation.

Manual: spans de lógica de negocio (reservar stock), atributos de negocio (`tenant`, `order_id`) — respetando cardinales bajos.

Propagación: context extractor/injector (W3C `traceparent`) automático en protocolos comunes; manual en colas custom.

Exporter: OTLP → collector → backend.

INTERMEDIO

MICROSERVICIOS

Observabilidad

M126

RESPUESTA

Estructurado JSON: timestamp, level, service, `trace_id`, `span_id`, `tenant`, `msg`, y contexto como campos (no string concatenado).

Muestreo de logs exitosos (1 de 10) si el volumen mata el presupuesto; SIEMPRE 100% de errores y warnings.

PII: mask en el `append` (regex o librería) — compliance no es opcional.

Correlación: mismo `trace_id` en logs de todos los servicios = `grep` end-to-end.

INTERMEDIO

MICROSERVICIOS

Observabilidad

M125

RESPUESTA

El equipo dueño define SLOs por servicio basados en EXPECTATIVA DEL USUARIO, no en el valor default 99.9.

Dependencia en cadena: tu SLO requiere que el upstream cumpla (documento el requisito: 'necesito pagos con 99.95 y p99 300ms').

Error budget como contrato de riesgo: si lo agotas, se prioriza confiabilidad sobre features.

Revisar trimestralmente con datos, no con intuición.

INTERMEDIO

MICROSERVICIOS

Observabilidad

M128

RESPUESTA

1) Resumen ejecutivo (impacto: usuarios/dinero/duración).

2) Timeline con datos de métricas.

3) Causa raíz (contributing factors, no culpables).

4) Qué funcionó, qué falló (detección, mitigación, comunicación).

5) Action items con OWNER y FECHA (fixes, alertas faltantes, runbook).

Blameless: el sistema falló, las personas operaron el sistema que existía.

INTERMEDIO

MICROSERVICIOS

Observabilidad

M127

RESPUESTA

Métricas por dependencia: latencia p95, tasa de error, rate — separadas (tag provider).

Timeouts + breakers por dependencia; dashboard con estado de cada una.

Status pages de terceros enlazadas; SLA documentado.

Contratos: si el tercero no da SLO, asume el peor caso y diseña fallback — el riesgo es tuyo.

INTERMEDIO

MICROSERVICIOS

Observabilidad

M129

Dashboards por servicio: el layout mínimo viable

INTERMEDIO

MICROSERVICIOS

Observabilidad

M130

Oncall: qué debe poder hacer un respondiente SIN conocimiento del código

INTERMEDIO

MICROSERVICIOS

Observabilidad

M131

Synthetic monitoring: detectar el problema antes que el usuario

INTERMEDIO

MICROSERVICIOS

Observabilidad

M132

Escenario: «usuarios reportan errores que nuestras métricas no ven» — dónde mirar

INTERMEDIO

MICROSERVICIOS

Observabilidad

M133

Métricas con cardinalidad: el error que cuesta dinero

AVANZADO

MICROSERVICIOS

Observabilidad

M134

Alertas multi-ventana multi-burn-rate (el estándar SRE)

AVANZADO

MICROSERVICIOS

★

APIs y edge

M135

¿Qué es un API Gateway y qué responsabilidades tiene?

INTERMEDIO

MICROSERVICIOS

★

APIs y edge

M136

Versionado de APIs sin romper clientes

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Observabilidad

M130

RESPUESTA

Ver salud (dashboard), leer qué falló (alerta → runbook), ejecutar runbook (rollback, drain, flag off, escalar cola), ver quién es dueño de dependencias (catálogo), escalar.

Si requiere 'entrar al código y debuggear en vivo' para mitigar, el servicio no está listo para on-call → más flags de kill, más runbooks, más automatización.

INTERMEDIO

MICROSERVICIOS

Observabilidad

M129

RESPUESTA

Fila 1: salud (SLO compliance, error rate, p95/p99). Fila 2: tráfico (req/s por endpoint top, throughput de colas). Fila 3: dependencias (latencia/errores por servicio externo). Fila 4: infra (CPU/mem/replicas/cola profundidad).

Annotate deploys (líneas verticales) — el 80% de incidentes arranca 10 min después de un deploy.

Un dashboard por servicio, linkeado desde el catálogo y las alertas.

INTERMEDIO

MICROSERVICIOS

Observabilidad

M132

RESPUESTA

1) El path roto puede no estar instrumentado (endpoint nuevo sin métricas).

2) Falla en el cliente/borde (CDN, DNS, certificado) antes de llegar.

3) Sample del tracing esconde el caso (tail sampling sin regla de ese endpoint).

4) Errores de negocio (HTTP 200 con lógica fallida — metrics de códigos no captan).

Fix: RUM + synthetic del flujo reportado + log con código de negocio.

INTERMEDIO

MICROSERVICIOS

Observabilidad

M131

RESPUESTA

Transacciones sintéticas: script que ejecuta el flujo crítico (login→agregar→checkout) cada 1-5 min desde las regiones de usuarios.

Detecta: certificados vencidos, DNS roto, degradación regional, flujos que nadie probó tras el deploy.

Alerta si falla 2 de 3 corridas (evita flapping).

Barato comparado con un solo SEV1 no detectado.

INTERMEDIO

MICROSERVICIOS

Observabilidad

M134

RESPUESTA

Rápido: burn 14.4x en 1h (página — 'esto se cae ya').

Lento: burn 6x en 6h (página/aviso — 'consumirás el budget del mes').

Windows combinadas (1h + 5m) para confirmar y evitar flapping.

Todo sobre el error budget del SLO, no sobre CPU.

Resultado: páginas POCA pero RELEVANTES.

AVANZADO

MICROSERVICIOS

Observabilidad

M133

RESPUESTA

Cardinalidad = combinaciones de labels: path=/users/{id} (no el id real!), status, método — miles de series ok.

Fatal: labels con user_id, order_id, session → millones de series → Prometheus/Cloud Monitoring se ahoga y la factura explota.

Regla: cada label debe tener decenas-centenas de valores máximos.

Datos de alta cardinalidad → logs o traces, no métricas.

AVANZADO

MICROSERVICIOS



APIs y edge

M136

RESPUESTA

Estrategias: en la URL (/api/v1/orders), por header (Accept versionado) o query param.

Reglas: no romper contratos (añadir campos sí; quitar/renombrar no), deprecate avisando, mantener versiones en paralelo, OpenAPI por versión, tests de contrato (Pact).

INTERMEDIO

MICROSERVICIOS



APIs y edge

M135

RESPUESTA

Puerta única de entrada para clientes externos:

routing, autenticación/autorización (JWT), rate limiting, TLS, caché, agregación, CORS, versionado.

Implementaciones: Spring Cloud Gateway, Kong, Apigee.

Mantiene los cross-cutting concerns fuera de los microservicios.

INTERMEDIO

MICROSERVICIOS

APIs y edge

M137

Gateway anti-patrones: lo que NUNCA debe hacer

INTERMEDIO

MICROSERVICIOS

★

APIs y edge

M138

BFF (Backend for Frontend): el patrón que resuelve el disagreement

INTERMEDIO

MICROSERVICIOS

APIs y edge

M139

GraphQL entre microservicios: cuándo y las trampas

INTERMEDIO

MICROSERVICIOS

APIs y edge

M140

Contratos en el edge: schema-first y validación en runtime

INTERMEDIO

MICROSERVICIOS

★

APIs y edge

M141

API composition vs CQRS read model: tabla de decisión

INTERMEDIO

MICROSERVICIOS

★

APIs y edge

M142

Idempotency-Key: el estándar para POSTs críticos

INTERMEDIO

MICROSERVICIOS

APIs y edge

M143

Webhooks salientes: entrega confiable a terceros

AVANZADO

MICROSERVICIOS

Seguridad

M144

Flujo Authorization Code + PKCE: por qué es EL estándar web/móvil

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS



APIs y edge

M138

RESPUESTA

Un backend POR tipo de cliente (BFF-web, BFF-mobile): cada uno agrega/da forma a los datos como el cliente los necesita.
 Resuelve: 'la API genérica no sirve a nadie' y el choca-de-equipo frontend/backend.
 Dueño: el EQUIPO DEL CLIENTE (no un equipo central).
 Costo: duplicación controlada — compartida por libs comunes de dominio.

INTERMEDIO

MICROSERVICIOS

APIs y edge

M137

RESPUESTA

Lógica de negocio en el gateway (descuentos, reglas) → cambio de negocio = deploy del gateway (cuello).
 Transformaciones complejas de payload en cada request (latencia + imposibles de debuggear).
 Base de datos propia del gateway.
 El gateway: routing, auth token validation, rate limit, TLS, telemetría — NADA más.

INTERMEDIO

MICROSERVICIOS

APIs y edge

M140

RESPUESTA

OpenAPI/Proto ES el contrato: genera server stubs, client SDKs, mocks y documentación desde UNA fuente.
 Runtime: validar request/response contra el schema (springdoc + validator, o gateway) — desviaciones fallan rápido.
 CI: spectral lint (naming, errores consistentes), oasdiff para breaking changes entre versiones.
 El contrato vive, no es un PDF.

INTERMEDIO

MICROSERVICIOS

APIs y edge

M139

RESPUESTA

Valor: el cliente pide exactamente lo que quiere, un endpoint, menos round-trips.
 Federación (Apollo): cada micro publica su parte del supergraph — composición de schema.
 Trampas: resolver N+1 (dataloader obligatorio), autorización por campo, caché HTTP perdida, consultas destructivas (deep queries → depth/complexity limits).
 Úsalo en el edge (BFF), no como capa interna entre servicios.

INTERMEDIO

MICROSERVICIOS



APIs y edge

M142

RESPUESTA

Cliente genera UUID por INTENTO LÓGICO (no por retry) y lo manda en header.
 Servidor: guarda key → resultado (tabla/Redis con TTL 24h).
 Reintento con misma key = devuelve el resultado original SIN re-ejecutar.
 Stripe lo popularizó; también para consumidores de colas.
 Detalle: si la primera petición está EN CURSO, segunda espera o 409 — no duplicar.

INTERMEDIO

MICROSERVICIOS



APIs y edge

M141

RESPUESTA

Composition: pocos servicios (2-4), latencias bajas, filtros simples, datos frescos requeridos.
 Read model (CQRS): muchos servicios, filtros/agregaciones complejas, volumen alto de lecturas, staleness de segundos tolerable.
 Híbrido real: composition para detalle (fresco), read model para listados/búsqueda (rápido).
 No uses CQRS para todo: costo de mantener proyecciones.

INTERMEDIO

MICROSERVICIOS

Seguridad

M144

RESPUESTA

- 1) App redirige al IdP con code_challenge (hash del verifier).
- 2) Usuario loguea EN EL IDP (credenciales nunca tocan tu app).
- 3) Vuelve con code de un solo uso → app intercambia code+verifier por tokens (backend o SPAs públicos).

PKCE evita que un code interceptado sirva sin el verifier.
 Implicit flow: DEPRECIADO (tokens en URL). Client Credentials: solo service-to-service.

INTERMEDIO

MICROSERVICIOS

APIs y edge

M143

RESPUESTA

Firma: HMAC-SHA256 del body + timestamp en header (el receptor valida — anti-replay con ventana).
 Entrega: cola con reintentos exponenciales (horas, no segundos), circuit breaker por endpoint, DLQ visible.
 Versionado del payload (header X-Event-Type) y docs de verificación.
 Replay tool para el tercero (reenviar eventos de un rango) — reduce soporte 10x.

AVANZADO

MICROSERVICIOSSeguridadM145

OIDC: qué añade sobre OAuth2 (y los 3 tokens)

INTERMEDIO

MICROSERVICIOSSeguridadM146

Secrets en un ecosistema de microservicios: patrón completo

INTERMEDIO

MICROSERVICIOSSeguridadM147

PII entre microservicios: minimización y marcado

INTERMEDIO

MICROSERVICIOSSeguridadM148

Rate limiting por identidad vs por IP: cuándo cada uno

INTERMEDIO

MICROSERVICIOSSeguridadM149

Zero trust en una frase aplicado a microservicios

INTERMEDIO

MICROSERVICIOS★SeguridadM150

JWT en la práctica: los 5 errores que ves en producción

INTERMEDIO

MICROSERVICIOSSeguridadM151

On-behalf-of y token exchange: propagar identidad entre servicios

AVANZADO

MICROSERVICIOSSeguridadM152

Autorización centralizada vs por servicio (OPA/sidecar)

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Seguridad

M146

RESPUESTA

Fuente única: Secret Manager/Vault (nunca git, nunca imagen).
 Entrega: sidecar/CSI que inyecta como archivo (rotación sin restart) o variable al arranque.
 Acceso: identidad del servicio (SA/workload identity) con rol mínimo por secreto.
 Auditoría: quién leyó qué cuándo. Rotación: automatizada con doble-validación (ver tarjeta de rotación).

INTERMEDIO

MICROSERVICIOS

Seguridad

M145

RESPUESTA

OIDC = OAuth2 + capa de IDENTIDAD: id_token (JWT con quién es el usuario: sub, email), userinfo endpoint, discovery (/well-known/openid-configuration), scopes (openid profile email).
 access_token: para llamar APIs (autorización). refresh_token: renovar sin re-login.
 En microservicios: validas access_token (JWT) por servicio; el id_token es para el CLIENTE, no para APIs.

INTERMEDIO

MICROSERVICIOS

Seguridad

M148

RESPUESTA

Por IP: anónimos, protección DDoS gruesa (y castiga NAT corporativos).
 Por identidad/API key: clientes autenticados — el límite justo y negocio-relevante (plan free/pro).
 Combinado típico: IP en el edge (Armor/WAF), identidad en el gateway/servicio.
 Headers informativos (X-RateLimit-*) y 429 con Retry-After — el cliente que respeta es el cliente que no rompe.

INTERMEDIO

MICROSERVICIOS

Seguridad

M147

RESPUESTA

Envía el MÍNIMO: el servicio de pedidos no necesita la fecha de nacimiento del cliente (necesita cliente_id y nombre).
 Eventos con PII: cifrar campos sensibles (crypto-shredding para GDPR), o tokenizar.
 Marcado de esquemas: anotación/classification en el schema registry (pii=email) → pipelines aplican mask automáticamente.
 Inventario de PII por servicio (data mapping) — requisito de auditorías.

INTERMEDIO

MICROSERVICIOS

Seguridad

M150

RESPUESTA

1) Validar solo firma sin exp/aud/iss. 2) Aceptar alg 'none' o negociado. 3) Secret compartido en N servicios (HS256) sin rotación. 4) Guardar datos sensibles EN el payload (es Base64 legible). 5) Tokens de vida larga sin refresh ni revocación (logout que no desloguea).
 Fixes: librería que valide todo, JWKS+RS256, vidas cortas (15-60min) + refresh.

INTERMEDIO

MICROSERVICIOS

Seguridad

M149

RESPUESTA

Nunca confíes en la red: cada llamada se autentica y autoriza explícitamente (mTLS + tokens + políticas por servicio), minimiza privilegios, asume brecha.
 Contrario al 'castillo y foso' (VPN = dentro confío).
 Implementación progresiva: identidad de servicios → mTLS → políticas por método → auditoría continua.

INTERMEDIO

MICROSERVICIOS

Seguridad

M152

RESPUESTA

Por servicio: cada uno implementa sus reglas (simple, se dispersa).
 Central policy engine (OPA/Rego como sidecar o librería): políticas unificadas, testables, auditables; servicio pregunta '¿puede X hacer Y sobre Z?'.
 Datos necesarios para decidir (jerarquías, roles finos) → servicio de autorización con caché.
 Balance: reglas de dominio en el micro, políticas transversales centralizadas.

AVANZADO

MICROSERVICIOS

Seguridad

M151

RESPUESTA

Problema: servicio A llama a B — ¿token del USUARIO o del servicio?
 Token exchange (RFC 8693): A intercambia el token del usuario por uno delegado para B (audience B, act={sub de A}) — B ve QUIÉN es el usuario y QUIÉN llamó.
 Alternativa simple: propagar el token original + header interno firmado.
 Auditoría correcta requiere la cadena completa.

AVANZADO

MICROSERVICIOS

Seguridad

M153

mTLS entre servicios: qué resuelve y su costo

AVANZADO

MICROSERVICIOS

Seguridad

M154

Supply chain de microservicios: de commit a despliegue confiable

AVANZADO

MICROSERVICIOS

Seguridad

M155

Pentest-ready: los checks de seguridad de un microservicio nuevo

AVANZADO

MICROSERVICIOS

Seguridad

M156

Threat modeling de un flujo de pago en microservicios

AVANZADO

MICROSERVICIOS

Plataforma y despliegue

M157

¿Qué es DevOps?

BASICO

MICROSERVICIOS

★

Plataforma y despliegue

M158

¿Qué es CI/CD?

BASICO

MICROSERVICIOS

★

Plataforma y despliegue

M159

¿Qué es Kubernetes (K8s) y para qué sirve?

BASICO

MICROSERVICIOS

★

Plataforma y despliegue

M160

Contenedor vs máquina virtual: diferencia clave

BASICO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Seguridad

M154

RESPUESTA

SLSA niveles: build reproducible y provable (provenance firmada), dependencies escaneadas (Dependabot/Snyk), imágenes distroless escaneadas (Trivy/AR scanning), firma cosign, admisión solo firmadas (Binary Auth/K8s admission).
SBOM por release (syft) para responder '¿tenemos log4j?' en minutos.
Base images propias actualizadas semanalmente.

AVANZADO

MICROSERVICIOS

Seguridad

M153

RESPUESTA

Ambos se autentican con certificados → sin credenciales robadas no hay llamadas (zero trust interno); cifrado integral; identidad para el audit log.
Costo: emisión/rotación automática (service mesh lo hace: Istio SPIFFE), latencia de handshake (keep-alive mitiga).
Sin mesh: alternativa pragmática = red privada + token de servicio firmado.

AVANZADO

MICROSERVICIOS

Seguridad

M156

RESPUESTA

Diagrama: app → gateway → pedidos → pagos → banco. Por confianza: ¿puede un usuario repetir un cobro? (idempotency) ¿ver pagos de otro? (authz por recurso) ¿token reusado? (jti/nonce) ¿MITM interno? (mTLS) ¿secret leak? (vault) ¿reply attack del webhook? (firma+timestamp).
STRIDE por elemento: spoofing, tampering, repudiation, info disclosure, DoS, elevation.
Salida: riesgos rankeados + mitigaciones asignadas.

AVANZADO

MICROSERVICIOS

Seguridad

M155

RESPUESTA

Auth en TODOS los endpoints (test que falla sin token), autorización por recurso (user A no lee de user B — IDOR), input validation estricta, rate limits, TLS only, secrets fuera del repo (gitleaks en CI), headers de seguridad en responses, dependencias sin CVEs críticos, logs sin PII.
Checklist en el template — el auditor lo pedirá igual.

AVANZADO

MICROSERVICIOS ★

Plataforma y despliegue

M158

RESPUESTA

CI (integración continua): cada push compila + testea automáticamente (todos integran a un trunk varias veces al día) → los conflictos se detectan temprano.
CD (entrega/despliegue continuo): el artefacto verificado se DESPLIEGA automáticamente (entrega = listo para producción con 1 click; despliegue = va a producción solo).
Pipeline típico: build → unit tests → análisis estático → imagen → deploy staging → smoke → prod.
En GCP: Cloud Build + Cloud Deploy.

BASICO

MICROSERVICIOS

Plataforma y despliegue

M157

RESPUESTA

Cultura + prácticas que unen DESARROLLO y OPERACIONES: los mismos equipos que construyen despliegan y operan (you build it, you run it).
Prácticas: CI/CD, infraestructura como código, monitoreo, postmortems sin culpa.
No es un puesto ni una herramienta: es acortar el ciclo idea → producción CON calidad.
Se mide con las métricas DORA (deploy frequency, lead time, MTTR, change failure rate).

BASICO

MICROSERVICIOS ★

Plataforma y despliegue

M160

RESPUESTA

VM: virtualiza HARDWARE — cada una lleva SO completo (GBs), arranque en minutos, aislamiento fuerte.
Contenedor: virtualiza solo el SO (comparte kernel del host) — imagen de MBs, arranque en ms, proceso aislado con su filesystem propio (Docker).
Consecuencia: en una VM caben pocos servicios; con contenedores empaquetas un proceso por contenedor y Kubernetes orquesta cientos.
El contenedor es la unidad natural de un microservicio.

BASICO

MICROSERVICIOS ★

Plataforma y despliegue

M159

RESPUESTA

Orquestador de CONTENEDORES: despliega, escala, cura y expone tus contenedores en un clúster de máquinas.
Funciones clave: autorreparación (pod muerto → reinicia), escalado (HPA), despliegues sin downtime (rolling), balanceo, config/secretos.
Vocabulario: Pod (contenedor+), Deployment (réplicas+updates), Service (IP/DNS estable), Ingress (entrada HTTP).
En GCP: GKE (gestionado); si no quieres operar nodos → Autopilot.

BASICO

MICROSERVICIOS

Plataforma y despliegue

M161

Configuración centralizada en microservicios

INTERMEDIO

MICROSERVICIOS

★

Plataforma y despliegue

M162

Health checks: liveness vs readiness

INTERMEDIO

MICROSERVICIOS

★

Plataforma y despliegue

M163

Blue/green vs canary deployment

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M164

Plantilla de microservicio (service template): qué incluye

INTERMEDIO

MICROSERVICIOS

★

Plataforma y despliegue

M165

Requests y limits en K8s: cómo definirlos bien

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M166

PodDisruptionBudget y surges: deploys que no degradan

INTERMEDIO

MICROSERVICIOS

★

Plataforma y despliegue

M167

Feature flags: liberar ≠ desplegar

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M168

Ambientes efímeros (preview environments)

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS



Plataforma y despliegue

M162

RESPUESTA

Liveness: ¿el proceso vive? Si falla, K8s lo REINICIA.
 Readiness: ¿puedo atender tráfico? Si falla, me quitan del load balancer sin reiniciar.
 Spring Boot Actuator: /actuator/health/liveness y /readiness.
 Nunca hagas que liveness dependa de una BD caída (reiniciaría en bucle).

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M161

RESPUESTA

Config externalizada por entorno, fuera del jar: Spring Cloud Config (backend Git), Consul, o K8s ConfigMaps/Secrets.
 Beneficios: un solo lugar para cambiar, refresco sin redeploy (@RefreshScope), historial y rollback.
 Secretos NUNCA en el repo: Secret Manager / Vault.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M164

RESPUESTA

Estructura lista con: health endpoints, métricas/tracing, logging JSON, config externalizada, resiliencia por defecto (timeouts), Dockerfile distroless, CI/CD pipeline, tests de ejemplo, README/runbook, chart de Helm/Terraform module.
 Generada con cookiecutter/backstage template.
 Beneficio: un servicio nuevo nace OBSERVABLE y seguro en día 1; equipos no reinventan el cableado.

INTERMEDIO

MICROSERVICIOS



Plataforma y despliegue

M163

RESPUESTA

Blue/green: dos entornos idénticos; el router cambia de blue a green de golpe; rollback instantáneo.
 Canary: el release nuevo recibe un % creciente de tráfico (1% → 10% → 50% → 100%) mientras vigila métricas; limita el impacto.
 K8s + Istio/Gateway API lo automatizan.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M166

RESPUESTA

PDB: mínimo de réplicas disponibles durante drenajes (nodos updates, autoscaler) — minAvailable: 2 o maxUnavailable: 1.
 Rolling: maxSurge (cuántos pods extra al desplegar) + maxUnavailable (cuántos pueden faltar) — típico 25/25.
 Sin PDB, un upgrade de nodos puede tumbar tu servicio completo.
 readiness gate correcta = prerequisite de todo esto.

INTERMEDIO

MICROSERVICIOS



Plataforma y despliegue

M165

RESPUESTA

Requests: lo que el POD asegura (planifica el scheduler, base del HPA) — usa p50-p75 real.
 Limits: techo (CPU se throttles, memoria OOMKilled) — p99 con margen ~20-30%.
 Medir con VPA en modo recomendación antes de fijar.
 Requests altos = dinero desperdiciado (nodos sobredimensionados); bajos = throttling y OOM.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M168

RESPUESTA

Por PR: despliega el servicio + dependencias simuladas en namespace/URL efímero (pr-1234.tuapp.com).
 Validación real del reviewer: click, no imaginar.
 Ciclo: crear al abrir PR, destruir al cerrar (costo 0 en reposo).
 Con datos: subsets anonimizados o sintéticos — nunca PII de prod en previews.

INTERMEDIO

MICROSERVICIOS



Plataforma y despliegue

M167

RESPUESTA

Deploy = código en prod; release = usuarios lo ven. Flags separan los dos.
 Tipos: release (apagar feature nueva), ops (kill switch), experiment (A/B %), permission (beta users).
 Gobernanza: dueño, expiración, limpieza periódica — flags muertos son deuda técnica invisible.
 Testing: CI corre con flags en AMBOS estados (el flag es una rama lógica).

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M169

Sidecar, ambassador, adapter: patrones de contenedor auxiliar

INTERMEDIO

MICROSERVICIOS



Plataforma y despliegue

M170

Stateless de verdad: los 5 estados escondidos que rompen el scaling

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M171

Progressive delivery: el continuo de estrategias

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M172

Estructura de un repo de microservicio (Java/Spring) en 2025

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M173

Pipeline CI/CD de un microservicio: etapas y puertas

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M174

Secrets y config por entorno sin duplicar YAML

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M175

Versionado de imágenes y trazabilidad de deploy

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M176

Consumidores de colas: despliegue y compatibilidad

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS



Plataforma y despliegue

M170

RESPUESTA

1) Sesiones en memoria (sticky sessions). 2) Cachés locales por instancia (respuestas inconsistentes tras scale). 3) Archivos en disco local. 4) Contadores/timers en memoria. 5) Scheduled jobs corriendo en todas las réplicas.

Fixes: JWT/Redis sessions, caché distribuida, GCS/S3 para archivos, DB sequences, ShedLock/scheduler externo.

Test: matar una réplica en caliente — ¿se pierde algo?

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M169

RESPUESTA

Sidecar: acompaña al principal compartiendo su vida (proxy, collector de logs, sync de secrets).

Ambassador: proxy a dependencias (cloud-sql-proxy, connector) — la app habla 'localhost'.

Adapter: expone la app en formato estándar (exporter de métricas legacy).

K8s: contenedores del mismo pod, localhost compartido, lifecycle juntos.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M172

RESPUESTA

src/main/java (api: controllers/DTOs, domain: entidades+lógica, app: services/casos de uso, infra: repos/config/clients), src/test (unit + integration).

Root: Dockerfile (multi-stage), cloudbuild.yaml/.github workflows, openapi.yaml o generado, README + runbook, chart/Terraform, ADRs en docs/.

Convención > configuración: mismo layout en todos los servicios = onboarding instantáneo.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M171

RESPUESTA

Recreate (downtime) → rolling (default k8s) → blue/green (switch instantáneo, doble costo temporal) → canary (% creciente con métricas) → shadow/mirror (tráfico real copiado sin respuesta — probar con riesgo 0).

Elegir por riesgo del cambio y madurez de métricas.

Shadow para cambios de alto riesgo (nuevo motor de pricing) — valida con tráfico real sin exponer usuarios.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M174

RESPUESTA

Base común: application.yml del jar (defaults seguros).

Por entorno: ConfigMap/Config Server con overrides (solo lo que cambia).

Secrets: NUNCA en el YAML de entorno — Secret Manager montado/inyectado por el runtime.

Convención de nombres: SPRING_DATASOURCE_URL etc. — relaxed binding une YAML y env vars.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M173

RESPUESTA

PR: compile + unit + lint + contract tests del consumer + build imagen (no deploy) + security scan.

Merge a main: integration tests (Testcontainers) + push imagen tag SHA + deploy dev automático + smoke + contract verify si soy provider.

Release: tag → deploy stage → canary prod con análisis de métricas → 100%.

Todo falla rápido y es visible en el PR.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M176

RESPUESTA

El esquema del mensaje es un contrato: consumer DEBE tolerar campos extra (ignorar desconocidos) y validar lo esencial.

Deploy de consumer antes que del producer (backward compatible primero).

Durante un deploy hay 2 versiones consumiendo: una transacción atómica por mensaje + offset/persistencia de posición.

Blue/green de consumers: nueva versión a grupo separado hasta validar.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M175

RESPUESTA

Tag SIEMPRE con: git SHA (app:9f3c2ab) + semántico opcional (app:2.4.1). Never 'latest' en prod.

Deployment anotado con SHA, build URL, quién aprobó.

Rollback = redeploy del SHA anterior (imagen inmutable en registry con retention).

La pregunta '¿qué versión corre en prod AHORA?' se responde en 5 segundos.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M177

Escenario: «regulatorio exige saber dónde vive cada dato» — mapa de datos

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M178

Escenario: «deploy requiere 15 pasos manuales» — automatización mínima viable

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M179

Escenario: «el mismo config en 5 servicios se desincroniza» — arquitectura de config compartida

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M180

gRPC interno: contratos y mejores prácticas entre servicios

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M181

API interna vs externa: por qué NO exponer tus protos/entidades

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M182

HPA con métricas custom: escalar por negocio, no solo CPU

AVANZADO

MICROSERVICIOS

Plataforma y despliegue

M183

Migraciones de schema compatibles: el contrato entre deploys

AVANZADO

MICROSERVICIOS

Plataforma y despliegue

M184

Canary automated: qué métricas deciden promover

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Plataforma y despliegue

M178

RESPUESTA

Semana 1: script único (deploy.sh) con los pasos — ya elimina error humano.
 Semana 2: mismo script en CI con parámetros entorno/versión.
 Luego: gates (tests, aprobación) en el pipeline, rollback automático en fallo de smoke.
 La automatización incremental vale más que esperar la plataforma perfecta.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M177

RESPUESTA

Inventario por servicio: qué datos personales procesa, en qué almacenes (SQL, Redis, logs, colas, backups), regiones.
 Herramientas: DLP scan sobre almacenes + catálogo con clasificación por servicio.
 Controles: residency por región, retención automática, borrado trazable (right to erfection como job auditable).
 Se demuestra con evidencia, no con promesas.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M180

RESPUESTA

Protos en repos compartido (api-definitions) con versioning semántico por paquete (pedidos.v1).
 Deadline SIEMPRE propagado; keepalive configurado; status codes gRPC mapeados a errores de negocio (FAILED_PRECONDITION vs INTERNAL).
 Interceptors: auth (JWT), tracing, recovery — infra común en librería interna.
 Reflection habilitada en staging para debug con grpcurl.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M179

RESPUESTA

Propiedad compartida → biblioteca común de config (starter propio con defaults) o config central con herencia (defaults org + override por servicio).
 Cambios auditados en un solo repo; versiones de la librería pinneadas.
 Config runtime compartida (timeouts de plataforma) via config server/ConfigMap común referenciado.
 Anti-patrón: copiar-pegar la misma propiedad en 5 YAMLS.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M182

RESPUESTA

CPU engaña en I/O-bound (hilos bloqueados ≠ CPU alta).
 HPA v2 con métricas externas: longitud de cola (Pub/Sub backlog), p99 latencia, QPS por pod (via Prometheus Adapter/KEDA).
 KEDA: ScaledObject por consumidor — escala a 0 con colas vacías, replica por lag.
 Objetivo tipo: '1 pod por 100 mensajes pendientes'.

AVANZADO

MICROSERVICIOS

Plataforma y despliegue

M181

RESPUESTA

API externa: REST/JSON estable, versionada para terceros, auth de usuarios.
 API interna: gRPC/protobuf, puede evolucionar más rápido, auth de servicios (mTLS/tokens).
 Compartir el mismo contrato = los cambios internos rompen a externos (o external congelan tu evolución).
 Traducción en el edge (gateway/BFF) — el costo extra compra libertad interna.

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M184

RESPUESTA

Comparar canary vs baseline (stable) en: error rate, p99 latencia, throughput esperado, y señales de negocio (conversion, checkout success).
 Reglas de decisión: si canary empeora > X% con significancia estadística → rollback automático.
 Herramientas: Argo Rollouts (analysis template), Flagger (promote/rollback automático), Istio traffic split.
 Sin métricas automáticas, 'canary' es solo un deploy lento a mano.

AVANZADO

MICROSERVICIOS

Plataforma y despliegue

M183

RESPUESTA

Durante un deploy conviven versión N y N+1 → el schema debe servir a AMBAS.
 Orden: ADD nullable → backfill por batches → deploy código que usa la nueva → (release siguiente) DROP/RENAME viejo.
 Prohibido: renombrar columnas directas (use rename + vista o doble escritura temporal).
 Herramientas: Flyway/Liquibase + CI que valida contra versión previa del código.

AVANZADO

MICROSERVICIOS

Plataforma y despliegue

M185

¿Qué es un service mesh y cuándo NO vale la pena?

AVANZADO

MICROSERVICIOS

Plataforma y despliegue

M186

Multi-region de un microservicio: el checklist concreto

AVANZADO

MICROSERVICIOS

Testing y calidad

M187

Testcontainers: BDs reales en tests de integración

INTERMEDIO

MICROSERVICIOS

Testing y calidad

M188

E2E en microservicios: la pirámide invertida a evitar

INTERMEDIO

MICROSERVICIOS

Testing y calidad

M189

Escenario: «la suite de integración tarda 40 minutos» — plan de reducción

INTERMEDIO

MICROSERVICIOS

Testing y calidad

M190

Escenario: «no podemos probar el flujo completo antes de producción»

INTERMEDIO

MICROSERVICIOS

Testing y calidad

M191

Contract testing consumer-driven (Pact): el flujo

AVANZADO

MICROSERVICIOS

Testing y calidad

M192

Testing en producción: formas seguras

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Plataforma y despliegue

M186

RESPUESTA

Datos: estrategia por BD (replica global o por región + partición de usuarios).
 Tráfico: LB global, deployment en ambas regiones.
 Estado: stateless (sesiones/caché compartidos o regionales).
 Deploy: pipeline despliega ambas con canary independiente por región.
 DR probado: drill que apaga una región y mide el impacto real.

AVANZADO

MICROSERVICIOS

Plataforma y despliegue

M185

RESPUESTA

Capa de infra (Istio/Linkerd) que inyecta proxies para mTLS, retries, timeouts, traffic split, telemetría — SIN tocar el código.
 Vale: 10+ servicios, multi-equipo, zero-trust exigido, canary sofisticado.
 No vale: pocos servicios, equipo chico (operar Istio es un trabajo), latencia sensible al extremo.
 Alternativa media: librerías in-app (Resilience4j) + gateway.

AVANZADO

MICROSERVICIOS

Testing y calidad

M188

RESPUESTA

Anti-patrón: suite E2E gigante que cruza 10 servicios — frágil (cualquier deploy rompe tests ajenos), lenta, flaky, difícil de atribuir.
 Reemplazo: contract tests entre pares + component tests por servicio (stubs de dependencias) + pocos E2E críticos (login, compra).
 Cada equipo ejecuta sus tests contra pacts, no contra deploys de otros.

INTERMEDIO

MICROSERVICIOS

Testing y calidad

M187

RESPUESTA

Docker + testcontainers levanta Postgres/Kafka/Redis reales por sesión de test (o singleton compartido).
 Pruebas lo que verás en prod: migraciones Flyway reales, dialecto real, transacciones reales.
 Patrón singleton container para velocidad (una BD por suite, limpieza por test).
 Alternativa en CI: contenedores sidecar del pipeline.

INTERMEDIO

MICROSERVICIOS

Testing y calidad

M190

RESPUESTA

Realidad de microservicios: el 'ambiente completo' es una ilusión costosa.
 Sustitutos: contract tests por par (cada dependencia verificada localmente), staging con stubs de terceros + servicios reales clave, canary/shadow en prod como validación final con reversión automática.
 El entorno de integración compartido eterno se sustituye por confianza CONTRACTUAL verificada por máquinas.

INTERMEDIO

MICROSERVICIOS

Testing y calidad

M189

RESPUESTA

- 1) Medir: qué tests son lentos y por qué (startup de contexto, BD, sleeps).
- 2) Slices sobre @SpringBootTest (WebMvcTest/DataJpaTest) donde no se necesita todo.
- 3) Contenedores singleton + reutilización, datos mínimos por test.
- 4) Parallelización con fuentes de datos aisladas.
- 5) Sleeps → Awaitility; contextos duplicados → unificar configuración de test (cache de contexto).

INTERMEDIO

MICROSERVICIOS

Testing y calidad

M192

RESPUESTA

Feature flags para experimentos controlados, canary como test de tráfico real, synthetic probes (transacciones sintéticas constantes), chaos experiments acotados, shadow traffic (duplicar tráfico sin responder).
 Pre-requisitos: aislamiento de datos de test, IDs marcados (test-user), alertas que reconozcan tests.
 Nunca 'probemos en prod' sin blast radius definido.

AVANZADO

MICROSERVICIOS ★

Testing y calidad

M191

RESPUESTA

- 1) Consumer escribe expectativas → genera pact file (JSON de interacciones).
- 2) Pact se publica al broker.
- 3) Provider verifica el pact contra su implementación real.
- 4) can-i-deploy: CI del provider verifica que TODOS los consumidores activos funcionan antes de desplegar.

Elimina el ambiente de integración compartido como cuello de botella.

AVANZADO

MICROSERVICIOS

Operación y organización

M193

Runbook: el documento que salva el on-call

INTERMEDIO

MICROSERVICIOS

Operación y organización

M194

On-call sostenible: rotación, severidades y handoff

INTERMEDIO

MICROSERVICIOS

Operación y organización

M195

Incident management: roles en un SEV1

INTERMEDIO

MICROSERVICIOS

Operación y organización

M196

FinOps por microservicio: atribuir costo y crear incentivos

INTERMEDIO

MICROSERVICIOS

Operación y organización

M197

Deprecar y matar un microservicio: el proceso que nadie documenta

INTERMEDIO

MICROSERVICIOS

Operación y organización

M198

Catalog de servicios (Backstage): para qué sirve de verdad

INTERMEDIO

MICROSERVICIOS

Operación y organización

M199

Capacity planning: de la predicción al presupuesto

INTERMEDIO

MICROSERVICIOS

Operación y organización

M200

Cortar un servicio de otro equipo: la transición

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Operación y organización

M194

RESPUESTA

Rotación primario/backup semanal (mínimo 6 personas para no quemar), compensación o reducción de tareas, severidades definidas (SEV1 despierta, SEV3 ticket).
 Handoff documentado: qué pasó, qué está pendiente, qué alertas están silenciadas y por qué.
 Métrica de salud: páginas por semana por persona (<2 sostenible).
 On-call que nadie quiere = señal de deuda de confiabilidad.

INTERMEDIO

MICROSERVICIOS

Operación y organización

M193

RESPUESTA

Por servicio: qué hace (1 párrafo), arquitectura (diagrama), dependencias y sus dashboards, alertas explicadas (qué significa + qué hacer), procedimientos comunes (restart seguro, drain, backfill), escalado a quién (escala de contacto), límites y cuotas. Enlazado desde CADA alerta.
 Test real: alguien ajeno al equipo lo sigue en un game day.

INTERMEDIO

MICROSERVICIOS

Operación y organización

M196

RESPUESTA

Labels/tags por servicio + billing export → costo por servicio/equipo visible mensualmente.
 Métrica de eficiencia: costo por transacción/por usuario (tendencia, no absoluto).
 Presupuesto por equipo con alertas; costos del shared (nats, logging) repartidos por uso.
 Efecto: el equipo que paga por over-provisioning, lo arregla — sin mesa de discusión.

INTERMEDIO

MICROSERVICIOS

Operación y organización

M195

RESPUESTA

Incident Commander (coordina, no arregla), Comms (stakeholders/status page), Operations (mitiga), Scribe (timeline). Canal dedicado + puente de voz; timeline con timestamps desde el minuto 0.
 Mitigación primero (rollback/flag/drain), causa raíz DESPUÉS.
 Postmortem en 48h, blameless, action items con fecha.

INTERMEDIO

MICROSERVICIOS

Operación y organización

M198

RESPUESTA

Inventario: quién es dueño de qué, repos, dashboards, runbooks, SLOs, dependencias (mapa vivo), on-call actual.
 Scorecards: madurez de seguridad/observabilidad por servicio — empuja estándares sin perseguir equipos.
 Onboarding: 'nuevo servicio' genera template completo.
 Sin catálogo, preguntar '¿quién es dueño de X?' toma días.

INTERMEDIO

MICROSERVICIOS

Operación y organización

M197

RESPUESTA

1) Anunciar consumidores (el catálogo lista quién te llama) con fecha límite.
 2) Métrica de tráfico residual → contactar rezagados.
 3) 410 Gone con aviso en la respuesta, luego apagar.
 4) Archivar datos según retención legal, borrar el resto.
 5) Documentar en el catálogo: 'absorbido por X'.
 Servicios zombis pagando infra y parches = deuda silenciosa.

INTERMEDIO

MICROSERVICIOS

Operación y organización

M200

RESPUESTA

1) Catálogo actualiza ownership; 2) transferencia de conocimiento (sesiones + docs), no solo git; 3) nuevo equipo empieza on-call acompañado (shadow 2 semanas); 4) SLOs/runbooks revisados; 5) accesos (IAM, dashboards, repos) migrados; 6) período de hypercare 30 días.
 Servicio sin dueño claro en la transición = incidente sin respondiente.

INTERMEDIO

MICROSERVICIOS

Operación y organización

M199

RESPUESTA

Datos: crecimiento histórico de QPS/datos, eventos estacionales (Black Friday), margen de seguridad 2x.
 Costos por réplica → presupuesto requerido por trimestre.
 Load tests que VALIDEN la predicción (¿cada pod realmente aguanta 100 rps?).
 Plan de contingencia: qué se degrada primero si duplica el tráfico (prioridades ya definidas).

INTERMEDIO

MICROSERVICIOS

Operación y organización

M201

El costo real de los microservicios (para defender la decisión)

INTERMEDIO

MICROSERVICIOS

Operación y organización

M202

Escenario: «presupuesto de observabilidad disparado» — optimización

INTERMEDIO

MICROSERVICIOS

Operación y organización

M203

Documentación viva de un servicio: el set mínimo

INTERMEDIO

MICROSERVICIOS

Operación y organización

M204

Estimación de carga para un servicio nuevo (metodología)

INTERMEDIO

MICROSERVICIOS

Operación y organización

M205

Game days: ensayar incidentes con guion

INTERMEDIO

MICROSERVICIOS

Operación y organización

M206

Cómo justificar inversión en confiabilidad con números

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

MICROSERVICIOS

Operación y organización

M202

RESPUESTA

Logging: exclusion sinks de ruido, sampling de successes, retención escalonada (hot 7d, archivo GCS).
 Tracing: bajar head sampling, tail solo errores/lentos.
 Métricas: matar series de alta cardinalidad (top offender report).
 Cada optimización con validación: '¿siguió detectándose el incidente X?' — la observabilidad no se recorta a ciego.

INTERMEDIO

MICROSERVICIOS

Operación y organización

M201

RESPUESTA

Costos: infra distribuida (más VMs/pods), observabilidad (logging/tracing mensual), plataforma (equipo que la mantiene), latencia de red, complejidad de datos (sagas/CQRS).
 Beneficios: despliegue independiente (time-to-market), escalado selectivo, equipos desacoplados, fallos acotados.
 Decisión honesta: solo si los beneficios pesan para TU escala — 5 devs raramente justifica 30 servicios.

INTERMEDIO

MICROSERVICIOS

Operación y organización

M204

RESPUESTA

1) Volumen esperado del negocio (usuarios, transacciones/día). 2) Pico: promedio x factor horario (10-50x en flashes). 3) Por request: tamaño y count de queries/llamadas. 4) Capacidad: throughput por réplica medido en load test (no asumido). 5) Réplicas = pico/throughput x margen 2x.
 Revisar con datos reales al mes 1 — toda estimación inicial está mal en algo.

INTERMEDIO

MICROSERVICIOS

Operación y organización

M203

RESPUESTA

README (qué hace, cómo correr), runbook (alertas y acciones), ADRs (decisiones), diagrama C4 container, contrato API (spec generado), SLOs y dashboards linkeados, ownership (equipo + canal).
 Todo EN EL REPO o linkeado del catálogo.
 Test de vida: si un alerta enlaza al runbook correcto, la doc vive; si nadie la actualizó en un año, está muerta.

INTERMEDIO

MICROSERVICIOS

Operación y organización

M206

RESPUESTA

Traduce a dinero: incidentes del último año x duración x revenue/minute afectado = costo de NO invertir.
 Comparar: precio de la mejora (equipo/infra) vs pérdida evitada + churn de clientes grandes.
 Agrega riesgo regulatorio si aplica (multas).
 El SRE budget se aprueba con historia de incidentes cuantificada, no con 'sentimos que faltan pruebas'.

AVANZADO

MICROSERVICIOS

Operación y organización

M205

RESPUESTA

Escenario planificado (caída de Redis, pérdida de región, pico 5x), fecha agendada, roles reales de on-call, observadores que anotan tiempos de detección/mitigación.
 Resultado: gaps concretos (alerta faltante, runbook inexistente, acceso sin permiso) → backlog de confiabilidad.
 Frecuencia: trimestral por equipo de on-call. La primera vez SIEMPRE se descubre algo vergonzoso — mejor en un game day.

INTERMEDIO